Ruby

Radek Hnilica
hnilica.cz

Radek DOT Hnilica AT gmail DOT com

Sestavil
Radek Hnilica

Ruby

Radek Hnilica

Sestavil Radek Hnilica

Working Vydání
Vydáno pubdate:

Tato „kniha“ je v jistém ohledu mým pracovním sešitem o Ruby. Pokud mám čas, zapisuji sem poznatky které jsem získal při práci s Ruby a Ruby on Rails. Nejsem profesionální „písmák“, a tak to místy připomíná haldu starého harampádí ve které jak doufám se sem tam zablýskne perla.

Tento dokument je k dispozici v několika různých formátech. Jako vícestránkový HTML dokument (index.html), postscriptový (ruby.ps) či PDF (ruby.pdf) soubor formátovaný na velikost papíru A4. Pokud některý z těchto formátů nenaleznete, nebo bude neaktuální dejte mi vědět, připravím jej pro vás.

Aktuální verze knihy je vystavena na www.hnilica.cz (http://www.hnilica.cz/radek/book/ruby/index.html), www2.hnilica.cz (http://www2.hnilica.cz/radek/book/ruby/index.html) a na penguin.cz/~radek (http://www.penguin.cz/~radek/book/ruby/index.html). Některé z těchto webů nemusí být dosažitelné.

Pokud budu mít možnost vystavit tento dokument i jinde, rád tak učiním.

Počet stran v Postscriptové (ruby.ps) a PDF (ruby.pdf) verzi: 487 .

Příšerně žluťoučký kůň úpěl ďábelské ódy.

Hleď, toť přízračný kůň v mátožné póze šíleně úpí.

Přehled revizí

Revize 0.0 2002-09-29
První publikovaná, pracovní, verze.
Revize 0.x 2002-10-18
Pracovní verze.
Revize 0.1 2002-12-09
Pracovní výtisk.
Revize 0.2 2003-10-10
Pracovní výtisk.
Revize 0.3 2008-03-05
Po dlouhé době oficiální revize. Pracuji především na kapitole Ruby on Rails, a zanedbávám kapitoly věnované samotnému Ruby.
Revize 0.4 2009-06-24
Po obnově.
Revize 0.5 2010-03-18
Zavedeno do Git serveru.
Revize 0.6 2010-10-07
Změna licence na Creative Commons BY-NC-SA.

Věnování tag dedication/title

* *Rozmyslet si komu tuto knihu věnuji a dopsat věnování.*

    BLOCKQUOTE

Obsah

Seznam tabulek

FIXME: colophon/title

FIXME: colophon/title

Tento dokument je psán s pomocí značkovacího jazyka DocBook (http://www.docbook.org/tdg/en/html/) editorem Emacs (http://www.gnu.org/software/emacs/) a transformován do různých formátů nsátroji: DocBook XSL Stylesheets (http://wiki.docbook.org/topic/DocBookXslStylesheets), OpenJade (http://openjade.sourceforge.net/) a množstvím skriptů v Bashi (http://www.gnu.org/software/bash/) a Ruby (http://www.ruby-lang.org/en/) na operačním systému Debian (www.debian.org).

Předmluva

* *preface id="preface"*

In my eyes it is never a crime to steal knowledge.
It as i good theft.

The pirate of knowledge is a good pirate.

Michel Serres

Nektěří lidé preferují nechávat si všechny znalosti pro sebe, někteří v nich vidí možnost prodat je za velkou cenu. Já v informacích vidím obrovské bohatství celého společenství. V mých očích má každá informace sdělená druhému velkou cenu. I proto, zveřejňuji své poznámky, abych je sdělil vám.

* *Poznámky k dokumentu. Tento dokument je v celé šíři věnován programovacímu a skriptovacímu jazyku Ruby. Všechny zde uvedené kapitoly a jiné části jsou v přímé souvislosti s jazykem. Je probírána verze 1.6.7 instalovaná z debianovského balíčku z Debian Woody, z řady 1.7 pak verze 1.7.2 instalované z balíčků Debian Sarge a 1.7.3 či novější kompilovaná ze zdroju z CVS stromu. Dále 1.8.5 z Debian Etch a možná i další verze pokud jsem tento odstavec zapoměl opravit.*

1. Historie

* *Použité zdroje: A little bit of Ruby history (http://www.ruby.ch/en/rubyhistory.shtml), Comparing and introducing Ruby by Michael Neuman, Programmieren mit Ruby (http://www.approximit.com/rubybuch2/), ruby-talk:00382 (http://blade.nagaokaut.ac.jp/cgi-bin/ scat.rb/ruby/ruby-talk/00382), ruby-talk:15977 (http://blade.nagaokaut.ac.jp/cgi-bin/ scat.rb/ruby/ruby-talk/15977)*

Odkazy:

- A little bit of Ruby history (http://www.ruby.ch/en/rubyhistory.shtml)

Počátkem 80-tých let byl jeden student v Japonsku nadšen programovacímy jazyky. Snil o tom jednom jediném jazyku. Několik let poté se jeho sen stal skutečností. Vytvořil Ruby, jazyk o němž je tato kniha.

Před nějakým časem se Michael Neuman zeptal autora Ruby, Yukihira Matsumoty (Yukihiro Matsumoto) na historii Ruby a důvody jeho vzniku. Zde je původní odpověď:

* *Ten student se jmenoval Yukihiro Matsumoto, a sám o tom říká:*
* *Originální citovaný text.*

> Well, Ruby was born in Feb. 24 1993. I was talking with my colleague about the possibility of an object-oriented scripting language. I knew Perl (Perl4, not Perl5), but I didn't like it really, because it had smell of toy language (it still has). The object-oriented scripting language seemed very promising.
>
> I knew Python then. But I didn't like it, because I didn't think it was a true object-oriented language. — OO features appeared to be add-on to the language. As a language manic and OO fan for 15 years, I really wanted a genuine object-oriented, easy-to-use scripting language. I looked for, but couldn't find one.
>
> So, I decided to make it. It tooks several months to make the interpreter run. I put it the features I love to have in my language, such as iterators, exception handling, garbage collection.
>
> Then, I reorganized the fatures of Perl into a class library, and implemented them. I posted Ruby 0.95 to the Japanese domestic newsgroups in Dec. 1995.
>
> Since then, highly active mailing lists have been established and web pages formed.

* *FIXME: Následující odstavce opravit podle výše uvedených zdrojů.*

> Ruby 1.0 was released in Dec. 1996, 1.1 in Aug. 1997, 1.2 (stable version) and 1.3 (development version) were released in Dec. 1998.
>
> Next stable version 1.4 will be shipped this months (June 1999), hopefully.

* *Neumělý překlad do češtiny.*

> Dobře, Ruby se zrodil. 24 února 1993. Mluvil jsem s kolegou o možnostech objektově orientovaného skriptovacího jazyka. Znal jsem Perl (Perl4, ne Perl5), ale nelíbil se mi, protože měl nádech jayka na hraní (a pořád má). Objektově orientovaný skriptovací jazyk vypadal velmi slibně (nadějně).
>
> Znal jsem také Python. Ale ten se mi nelíbil, protože se nemyslím že je to byl opravdový objektově orientovaný jazyk — OO vlastnosti se zdají být (appeared) přídavkem k jazyku. Jako maniak (manic) do jazyků a příznivec OO 15 let jsem opravdu potřeboval (a genuine) objektově orientovaný, snadno použitelný skriptovací jazyk. Hledal jsem takový, ale nenalezl.
>
> Rozhodl jesm se tedy si takový udělat. Trvalo to několik měsíců než jsem mohl spustit interpret. Přidal jsem ty vlastnosti, které jsem chtěl mít ve svém jazyku jako iterátory, výjimky, (garbage collection).
>
> Poté jse reorganizoval vlastnosti Perlu do (class library) a implementoval je. I posted Ruby 0.95 to the Japanese Domestic newsgroups in Dec. 1995.
>
> Od té doby jsou ustanoveny (established) poštovní listy (mail list) a zformovány (formed) webovské stránky. Velmi aktivní diskuse byla vedena v mail listech. Nejstarší, ruby-lis má do dneška 14789 zpráv. Ruby 1.0 byl uvolněn/vypuštěn (released) 1996-12, 1.1 v 1997-08, 1.2 stabilní verze a 1.3 vývojová verze byly vypuštěny v 1998-12.

* *Pozor, časové údaje se asi vztahují k 2001-01-04.*

2. Struktura knihy, členění na části

Popíši členění knihy na části a popíši stručně jejich obsah.

* ***To be done.***

3. Zdroje (*Resources*)

Pokud jsou zdroje ze kterých čerpám, snažím se je uvádět na začátku každé jednotlivé kapitoly či sekce v seznamu odkazů. Některé části obsahují vlastně jen seznam odkazů.

4. Konvence použité při psaní tohoto dokumentu

* *Jak se píše kód, jak se v textu zvýrazňují určité druhy slov.*
* ***To be done.***

5. Nazapracované texty a části

Tento dokument započal svůj život jako pracovní sešit, kam jsem si psal poznámky k věcem které se mi špatně hledají, případně je chci mít na očích.

Někdy je těžké rozhodnout kam má daná informace patřit, protože je tento sešit o jazace Ruby a programování v něm, informace které se tohoto přímo netýkají zde proto neuvádím.

* *Poznámky ke struktuře dukumentu: Dokument má tyto části Předmluva, Kap.1 - Úvod, Kap.2 - Instalace (Instalace z binárních balíčků, kompilace ze zdrojů, instalace knihoven z binárních balíčků, kompilace knihoven ze zdrojů. Part I. -*

5.1. Poznámky autora

* *section condition="author"*

Poznámky autora jsou jen v autorské verzi dokumentu. Nejsou určeny k publikaci, ale popisují věci související s vytvářením dokumentu.

ToDo List

- Přidat šablonu kapitoly či kapitol „Calling C from Ruby and Ruby from C“

5.1.1. Struktura knihy (členění)

Navržená struktura knihy. Její členění na části, kapitoly a sekce s případným obsahem sekcí.

I. Předmluva
II. Úvod
III. Instalace a kompilace
IV. Spouštíme Ruby — První kroky
V. Úvod — (Elegance Ruby)
    1. Co je Ruby
    2. Historie
    3. Elegance
    4. Srovnání s ostatními jazyky

VI. Část Jazyk Ruby
    1. Základy jazyka
    2. Literály, konstanty
    3. Proměnné
    4. Operátory
    5. Metody
    6. Bloky — {...}, {|| ... }, do ... end, do | | ... end
    7. Iterátory — each, yield
    8. Větvení — if, unles, then, else, case
    9. Cykly — while, for
    10. Výjimky — begin ... end
    11. Objekty a třídy (Konstrukce objektů a tříd)
    12. Datové typy
    13. Bezpečnost

VII. Část Knihovny
    1. Řetězce (String)
    2. Pole a seznamy (Array)
    3. Slovníky (Hash)

VIII. Nástroje
    1. RDoc
    2. RDTool
    3. RAA *Ruby Application Archive*
    4. GUI
    5. Databáze
    6. Síť (Networking)
    7. XP (Extrémní programování)

IX. Reference: Zabudované proměnné
X. Reference: Globální konstanty

XI. Reference: Zabudované funkce
XII. Reference: Zabudované knihovny
XIII. Návrhové vzory *Design Patterns*
XIV. Příloha: Lidé okolo Ruby
XV. Bibliografie
XVI. Rejstřík
XVII. Index

Zajímavé názvy kapitol či sekcí. Uvážím jejich použití.

- Ruby v akci (*Ruby in Action*)
- Budoucnost ruby

5.1.2. Nový návrh členění knihy

Kniha je na nejvyšší úrovni členěna do částí. Navrhuji tyto části:

Členění knihy na části

- Úvodní část — prvotní kontakt čtenáře s jazkem Ruby, trocha historie, instalace, spuštění. Sestává s kapitol

  Úvod
  Sprovozňujeme ruby

- I – „*Tutoriál*“ v *Ruby* — provádí nás jazykem jako učitel
- Knihovna tříd a modulů — vyčerpávající popis tříd a modulů dodávaných s jazykem Ruby, jedná se vlastně o reference
- Nástroje — popis podpůrných nástrojů jako jsou například programy RDTool, RDoc, ...
- Programy a knihovny — popis některých programů a knihoven jenž jsou pro ruby k dispozici
- Návrhové vzory (*Design Patterns*) — tato část by mohla být i jen kapitolou

5.1.3. Slovníček

Jaké termíny používám.

Ruby

: Ruby s velkým „R“ používám jako název jazyka.

ruby

: ruby s malým „r“ navíc v tagu `application` používám pro označení interpretu jazkya Ruby.

Kapitola 1. Úvod

* *chapter id="chapter.introduction" xreflabel="Úvod"*
* *Popsat historii jazyka, ...*

Lidé kolem Ruby *itemizedlist spacing="compact" security="private"*

- Tomas Borland Valenta , Jan Becvar, Patrik Modesto, Martin Man, Petr Mach, Petr Chromec
- Jim Weirich ([3])
- Bruce Williams (http://codebliss.com) Bruce Williams (http://www.rubygarden.org/ruby?BruceWilliams) (Ruby enthusiast)

Ruby je vyšší (*high level*) programovací jazyk jenž výrůstá z kořenů čistě objektového jazyka Smalltalk a je mimo jine obohacen o „to nejlepší“ z jazyka Perl. Jeho tvůrcem je Matz (Yukihiro Matsumoto). Hal E. Fulton uvádí že ruby je velmi vysoký (*very high level*) programovací jazyk.

1.1. Co je Ruby

* *section security="private"*

Ruby

- je jazyk vyšší úrovně
- je objektově orientovaný
- dynamicky typový

Ruby ...

je jayzyk vyšší úrovně (*high level language*)

FIXME:

beztypový (typeless)

Proměnné v Ruby žádný nemají typ. Typový systém Ruby je dynamický. Typ má konkrétní hodnota. Do proměnné, jenž obsahuje číslo, může být přiřazen řetězec, metoda, objekt, ...

ryze objektově orientovaný

V Ruby „je všechno objekt“. Systém objektů vychází z objektů jazyka Smalltalk. Ruby nepoužívá vícenásobnou dědičnost, ale tu nahrazuje technologie *mix-in*.

interpretovaný

Programy/skripty jsou přímo spustitelné bez kompilace. Existuje interaktivní ruby: `irb`. Nevýhodou může být za určitých okolností pomalejší běh programu než v kompilovaných jazycích. Technologie interpretovaných jazyků ale již vyspěla a rychlost vykonávání programu je srovnatelná. Za určitých okolostí může program v ruby „běžet“ dokonce rychleji. Toto je velmi individuální.

má zabudovaný *garbage collector*

V ruby je zabudovaný *mark-and-sweep grabage collector*. Programátor se nemusí starat o uvolňování přidělené paměti.

portovatelný (*portable*)

Byl portován na Linux, mnoho UNIXů, Macintosh (OS 9, OS X), BeOS, OS/2, DOS, Windows 95/98/NT/2k

* *Podle: Ruby is THE ultimate VHLL-OO-Scripting-Language (http://www.ruby.ch/en/rubywhat.shtml)*

Ruby is THE ultimate VHLL-OO Scripting-Language

Ruby ...

- has a sound syntax
- comes with mark-and-sweep-garbage collection
- is type-less
- is pure object-oriented (i.e. "everything's an object")
- is highly reflective
- implements modules
- implements block closures (a la Smaltalk)
- implements mix-ins
- implements operator overloading
- implements method overloading
- implements a sound exception handling
- comes bundled with a few "go-4" - patterns
- has a powerful regular expression implementation

* *Podle: http://www.s-direktnet.de/homepages/neumann/ruby_en.html*

Ruby je

- interpretovaný jazyk. Výhoda: je přímo spustitelný bez kompilace, Nevýhoda: Pomalejší rychlost vykonávání programu než v kompilovaných jazycích jako je např. Pascal, C++, ...
- objektově orientovaný -- podobně jako ve Smalltalku je všechno objektem. Ruby nepoužívá vícenásobnou dědičnost, ale tu je možno nahradit pomocí mix-in.
- *portabel* ruby je vysoce portabilní. Tak je možné jeden a ten samý program spouštět beze změn na různých platformách UNIX, Windows, DOS, Mac, BeOS a dalších.
- beztypový -- proměnné v ruby nemají žádný typ, podobně jako ve Smalltalku, Pythonu. Ale vlastní data mají svůj typ.

1.2. Srovnání s ostatními jazyky

* *section condition="author"*
* *Podle Comparing Ruby to (http://www.ruby.ch/en/rubycompare.shtml)*

Perl

- - clumsy syntax
- - bad OO implementation/integration, not pure OO
- + good xml support
- + CPAN

Python

- - not pure OO
- -/+ bad identation
- + many bindings/modules
- + good XML support

- \+ CORBA (omniOrb) binding

Smalltalk

- \- inconvenient syntax
- +/- class browswes
- \+ still THE OO language

1.3. Principy

PomLA

principle of matz's least astonishment

PoMN

The matz's first principle of method names. „If you have a "right" name for the method, implement it. If you have any doubt in a name, just wait.“

1.4. Citáty

I was once like you are now, and I know that it's not easy, To be calm when you've found something going on. But take your time, think a lot, Why, think of everything you've got. For you will still be here tomorrow, but your dreams may not.

Son How can I try to explain, when I do he turns away again. It's always been the same, same old story. From the moment I could (ruby-)talk(20270) I was ordered to listen. Now there's a way and I know that I have to go away. I know I have to go.

Kapitola 2. Ruby

Skriptovací, interpretovaný jazyk nove generace

* *Kapitola určená ke zrušení. Její obsah bude rezdělen mezi kapitoly části III – „Knihovny, technologie, postupy“ v Ruby*
* *Kostra kapitoly: - Jazykové konstrukce - Typy (Objekt, číslo, řetězec, Pole, ...)*

Text kapitoly

Odkazy:

- Ruby Home Page (http://www.ruby-lang.org/en/index.html)
- RubyCentral (http://www.rubycentral.com/index.html)
- Ruby Garden (http://www.rubygarden.org/)
- RWiki (http://www.jin.gr.jp/~nahi/RWiki)
- John Johnson Software’s Ruby Stuff (http://www.johnjohnsonsoftware.com/ruby/)
- mod_ruby tutorial (http://sean.chittenden.org/programming/ruby/mod_ruby/apachecon-2002/)
- Pleac (http://pleac.sourceforge.net/)
- IOWA - Interpreted Objects for Web Applications (http://beta4.com/iowa/)
- Ruby Application Archive (http://www.ruby-lang.org/en/raa.html)

2.1. Řízení běhu (toku) programu

2.1.1. until

* *FIXME:*

2.2. Jazykové konstrukce

2.2.1. Metody

Definice metody s parametry

```
def myNewMethod(arg1, arg2, arg3)
    # Zde je kód metody
end
```

Definice metody bez parametrů

```
def myOtherNewMethod
    # Zde je kód metody
end
```

Metodě můžeme zadat implicitní hodnoty parametrů

```
def coolDude(arg1="Miles", arg2="Coltrane", arg3="Roach")
    "#{arg1}, #{arg2}, #{arg3}."
end
```

Metodu s implicitními parametry pak voláme takto

```
coolDude
coolDude("Bart")
collDude("Bart", "Elwood")
coolDude("Bart", "Elwood", "Linus")
```

2.2.1.1. Variable-Length Argument List

```
def varargs(arg1, *rest)
    "Got #{arg1} and #{rest.join(', ')}"
end

varargs("one")
varargs("one", "two")
varargs("one", "two", "three")
```

2.2.1.2. Metody a bloky

```
def takyBlock(p1)
    if block_given?
        yield(p1)
    else
        p1
    end
end
takeBlock("no block")
takeBlock("no block") {|s| s.sub(/no /, '')}

class TaxCalculator
    def initialize(name, &block)
        @name, @block = name, block
    end
    def getTax(amount)
        "#@name on #{amount} = #{ @block.call(amount) }"
    end
end

tc = TaxCalculator.new("Sales tax") {|amt| amt * 0.075 }
tc.getTax(100)
tc.getTax(250)
```

2.3. Datové typy

Základním a vlastně jediným datovým typem je *Objekt*. Toto je dědictvím Smalltalku, jenž plně určuje charakter jazyka. Ovšem pro snazší seznámení uvádím nejdříve konkrétní třídy než popíši samotnou konstrukci objektu v části 2.3.3.

Poznámka: Podobně jako ve Smalltalku je všechno objekt.

2.3.1. Řetězce znaků (string)

* *FIXME: dopsat*

Řetězec je pole (Array) znaků.

Řetězcové konstanty (literály)

Řetězce jenž neexpandují proměnné a bez speciálních znaků

```
'řetězec'
%q/řetězec/, %q[řetězec], %q{řetězec}, %q(řetězec), %q<řetězec>
```

Řetězce s expanzí/substitucí proměnných a se speciálními znaky

```
"řetězec se substitucí proměnné #{var}"
%Q/řetězec/, ...
```

Víceřádkový řetězec

```
a = <<"EOF"
Toto je mnohařádkový dokument
ukončený řetězcem EOF na samostatném řádku
EOF
```

Takovýto víceřádkový řetězec je ukončen stejnou značkou jako je zahájen, tedy ve výše uvedeném případě `EOF`. Tato značka musí být na začátku řádku. V případě že ji tam z estetických či jiných důvodů nechceme mít, použijeme podobnou konstrukci

```
    a = <<-EOF
Toto je mnohařádkový dokument
ukončený řetězcem EOF na samostatném řádku
    EOF
```

2.3.1.1. Operace s řetězci

* *FIXME: dopsat*

2.3.2. Regulární výrazy

```
/regulární výraz/, %r{regulární výraz}, %r|regulární výraz|,
```

2.3.3. Objekt (Object)

Všechno je objekt. A co to tedy je ten objekt. Objekt je konstrukce obsahující data (*atributy*) a kód (*metody*)

2.3.3.1. Metoda `new`

Konstruktor objektu.

2.3.3.2. Metoda `to_s`

Vytvoření textové reprezentace objektu pro tisk.

2.3.3.3. Metoda `initialize`

Inicializace nově vytvořeného objektu.

2.4. Metody objektu

2.4.1. Makra

2.4.1.1. `attr_accessor`

FIXME:

```
class ServiceCall
   attr_accessor :date, :symptom, :solution
end
```

2.5. Spouštíme Ruby

* *Předávání argumentů do programu, proměnné prostředí, knihovny, ... Ruby and Its World (http://www.rubycentral.com/book/rubyworld.html)*

2.5.1. Command-Line Arguments

FIXME:

```
ruby [options] [--] [programfile] [arguments]
```

2.5.2. Command-Line Options

`-0`*`[octal]`*

The number „0“ flag specifies the record separator character

2.5.3. Proměnné prostředí

Proměnné prostředí jenž Ruby používá

RUBYOPT

Additional command-line options to Ruby

RUBYLIB

Additional search path for Ruby programs ($SAFE must be 0).

RUBYPATH

With -S option, search path for Ruby programsdditional command-line options to Ruby

RUBYSHELL

FIXME:

DNL_LIBRARY_PATH

FIXME:

RUBYLIB_PREFIX

FIXME:

2.6. FIXME: vymyslet název

2.6.1. introspection (reflection)

FIXME: doplnit

I. Tutoriál

Tato část knihy je věnována začínajícím uživatelům. Zde vás krok za krokem seznámím s Ruby. Tedy alespoň se o to pokusím.

* *Výukové informace, tutoriály, videa a další materiály výukového charakteru.*

Knihy:

- The-Little-Book-Of-Ruby (http://www.sapphiresteel.com/The-Little-Book-Of-Ruby)
- The Book Of Ruby (http://www.sapphiresteel.com/The-Book-Of-Ruby)

Ruby Programming Tutorial od SapphireSteelDotCom (http://www.youtube.com/user/SapphireSteelDotCom)

1. Getting Started (http://www.youtube.com/watch?v=YQM4kpUxUPk) 3:52 [2009-09-22]
2. Object Orientation (http://www.youtube.com/watch?v=6It5aK9mJi8) 6:37 [2009-09-27]
3. Objects and Inheritance (http://www.youtube.com/watch?v=Pa6-nzeICI8) 4:26 [2009-12-09]

Programming With Ruby od manwithcode (http://www.youtube.com/user/manwithcode)

1. Introduction (ttp://www.youtube.com/watch?v=p3jyESVlA2k) 3:02 [2009-03-19]
2. Getting Started (http://www.youtube.com/watch?v=YLGQyKyWnXM) 4:36 [2009-03-19]
3. Getting Help/Tools (http://www.youtube.com/watch?v=xwzalx7OcA4) 8:58 [2009-03-25]
4. Main Ruby Concepts (http://www.youtube.com/watch?v=8W7MJrAzeWw) 9:55 [2009-04-09]
5. Numbers (http://www.youtube.com/watch?v=qdTM9mp0EsQ) 7:01 [2009-04-23]
6. Strings (http://www.youtube.com/watch?v=1ot2Wlsgsog) 5:59 [2009-06-11]
7. Arrays (http://www.youtube.com/watch?v=_jHM-3h-Bag) 7:42 [2009-06-22]
8. Hashes (http://www.youtube.com/watch?v=LIrTu1UDATk) 4:57 [2009-06-22]
9. Flow Control Part 1 (http://www.youtube.com/watch?v=6uMw60C7tyY) 9:31 [2009-07-06], Flow Control Part 2 (http://www.youtube.com/watch?v=zpqByOutHVU) 4:11 [2009-07-06]
10. Objects and Modules Part 1 (http://www.youtube.com/watch?v=q75BiSgI6QI) 9:51 [2009-07-16], Objects and Modules Part 2 (http://www.youtube.com/watch?v=AmOj09AVI8k)
11. Ruby Projects (http://www.youtube.com/watch?v=_F0UHFpk2R0) 8:35 [2009-07-16]
12. Documentation (http://www.youtube.com/watch?v=BEdmtC03who) 4:15
13. Basic IO (http://www.youtube.com/watch?v=x-ru_Hw7YNI) 9:33 [2009-07-23]
14. YAML (http://www.youtube.com/watch?v=NSifr3DflxQ) 7:37 [2009-07-23]
15. Error Handling (http://www.youtube.com/watch?v=97zxHTwEk6g) 8:59 [2009-07-23]
16. Benchmarking (http://www.youtube.com/watch?v=dsa2RLZQoJY) 6:33 [2009-07-23]
17. Getting Advanced (http://www.youtube.com/watch?v=0dnHp7Yhbuo) 9:45 [2009-07-30]
18. Out Into The World (http://www.youtube.com/watch?v=8jHsE27voRQ) 3:43 [2009-07-30]

Making Games with Ruby od manwithcode (http://www.youtube.com/user/manwithcode)

- Announcing: Making Games with Ruby (http://www.youtube.com/watch?v=ENmkaga2CQ8) [2009-11-27]
- Ep. 1 - Intro (http://www.youtube.com/watch?v=QnXPUEXKrzg) 4:40 [2010-01-31]
- Ep. 2 - Setup on Windows (http://www.youtube.com/watch?v=zJgyefzctRg) 2:28 [2010-01-31]
- Ep. 2 - Setup on Mac (http://www.youtube.com/watch?v=URGqLBfcI5A) 1:12 [2010-01-31]
- Ep. 2 - Setup on Linux (http://www.youtube.com/watch?v=aq0LGlMrQgM) 2:10 [2010-01-31]
- Ep. 3 - Basics (http://www.youtube.com/watch?v=rcsNp8deJVs) 7:37 [2010-02-11]

tknql(tekniqal.com):

- Duck Typing in Ruby (http://www.youtube.com/watch?v=apoy5gJYn7I) 2:01 [2009-03-06]
- Whitespace In Ruby (http://www.youtube.com/watch?v=QMzsPbMeq7Y) 9:19 [2009-03-06]
- Variable Scope in Ruby (http://www.youtube.com/watch?v=jGX3HIhVg0Q) 4:09 [2009-03-06]
- Methods in Ruby (http://www.youtube.com/watch?v=v8dOlaHIiyk) 5:50 [2009-03-09]
- Conditions in Ruby (http://www.youtube.com/watch?v=Wu6jRykvluA) 6:40 [2009-03-11]

- Loops in Ruby (http://www.youtube.com/watch?v=UVLdfHnppTg) 5:41 [2009-03-13]
- String Delimiters in Ruby (http://www.youtube.com/watch?v=qLwslbWuQrM) 6:26 [2009-03-16]
- Symbols in Ruby (http://www.youtube.com/watch?v=TeQIQuAFtpA) 7:39 [2009-03-25]
- Strings and Mutability in Ruby (http://www.youtube.com/watch?v=ZGs9T7qXO50) 4:11 [2009-03-25]
- Identifiers in Ruby (http://www.youtube.com/watch?v=SMcapC3YfXo) 1:06 [2009-03-29]
- Ranges in Ruby (http://www.youtube.com/watch?v=ylxSwdgi56c) 2:10 [2009-04-01]
- Creating Arrays in Ruby (http://www.youtube.com/watch?v=khdJxs7F-zE) 4:42 [2009-04-02]
- Accessing Arrays in Ruby (http://www.youtube.com/watch?v=AX7z21l3O6Y) 4:22 [2009-04-09]
- Manipulating Arrays in Ruby (Part 1 of 2) (http://www.youtube.com/watch?v=dlIxG9wj73U) 4:58 [2009-04-09]
- Manipulating Arrays in Ruby (Part 2 of 2) (http://www.youtube.com/watch?v=B2LLzI7TDmw) 3:58 [2009-04-09]
- Working with Hashes in Ruby (Part 1 of 2) (http://www.youtube.com/watch?v=fIDxM3WXODg) 3:30 [2009-04-10]
- Working with Hashes in Ruby (Part 2 of 2) (http://www.youtube.com/watch?v=SdXxddkTCfg) 3:51 [2009-04-10]

Ruby Tutorial od Lampes tutorials (http://www.youtube.com/user/lampestutchannel):

1. Installation und erstes Projekt (http://www.youtube.com/watch?v=0ZUWY8S9FsE) 6:37 [2010-02-06]
2. Zahlen in ruby (http://www.youtube.com/watch?v=4AeFKZKFPJU) 6:00 [2010-02-06]
3. Strings als Zeichenketten (http://www.youtube.com/watch?v=Lw4zVysh-gQ) 8:19 [2010-02-06]
4. Variablen und Typ umwandlung ! (http://www.youtube.com/watch?v=wpF1Sr9v7Tk) 8:38 [2010-02-06]
5. Von der Tastatur lesen mit gets (http://www.youtube.com/watch?v=4mQaB0dCOg8) 5:21 [2010-02-06]
6. Wahrheitswerte also booleans (http://www.youtube.com/watch?v=Ye-ZSdlfBxw) 4:50 [2010-02-08]
7. die if schleife (http://www.youtube.com/watch?v=eJoHWlbTmE4) 5:38 [2010-02-08]
8. die while schleife (http://www.youtube.com/watch?v=Pik027WxUFg) 6:21 [2010-02-08]
9. upto downto (http://www.youtube.com/watch?v=iVPHKGg2TYM) 5:49 [2010-02-10]
10. case (http://www.youtube.com/watch?v=bRZk9DpO45k) 4:41 [2010-02-21]
11. Array , Arrays (http://www.youtube.com/watch?v=xJ7l190j0dA) 8:00 [2010-02-21]
12. Hashes (http://www.youtube.com/watch?v=x8xpe9ugxgY) 8:17 [2010-02-28]
13. regex Regular Expressions (http://www.youtube.com/watch?v=d-aIa30moLM) 9:51 [2010-02-28]

YouTube cmatthieu (http://www.youtube.com/user/cmatthieu):

1. Rubyology ScreenCast 1 (http://www.youtube.com/watch?v=irkKLFpbG4M) 4:06 [2007-03-31]
2. Rubyology ScreenCast 4 (http://www.youtube.com/watch?v=19ieFcwX5d0) 5:49 [2007-03-31]

GoogleTechTalks:

- Building a More Efficient Ruby Interpreter (http://www.youtube.com/watch?v=ghLCtCwAKqQ) 36:10 [2009-12-14]
- Languages Matter (http://www.youtube.com/watch?v=ix2DeCzuckc) 14:25 [2009-11-20] — A short talk by Yukihiro "Matz" Matsumoto about programming languages.
- Ruby Meet Up 8/13/09: Ruby Files on Google App Engine (http://www.youtube.com/watch?v=pHMpf6hx8Ek) 44:12 [2009-08-13]
- Google I/O 2009 - JRuby & Ioke on Google App Engine for Java (http://www.youtube.com/watch?v=xTC6LVAc6Ps) 1:02:06 [2009-06-01]
- Merb, Rubinius and the Engine Yard Stack (http://www.youtube.com/watch?v=TcMklv40YMY) 47:35 [2008-10-20]
- JRuby: The power of Java and Ruby (http://www.youtube.com/watch?v=PfnP-8XbJao) 1:11:16 [2008-03-01]
- Ruby 1.9 (http://www.youtube.com/watch?v=oEkJvvGEtB4) 49:57 [2008-02-22]

- Code Generation With Ruby (http://www.youtube.com/watch?v=fv7J50IeBLs) 50:37 [2007-10-08]
- Ruby Sig: How To Design A Domain Specific Language (http://www.youtube.com/watch?v=PtVxg4ay63E) 1:02:38 [2007-10-08]
- Code Generation With Ruby (http://www.youtube.com/watch?v=fv7J50IeBLs) 50:38 [2007-10-08]
- Ruby And Google Maps (http://www.youtube.com/watch?v=wB-o6cCgcw0) 1:02:31 [2007-10-08]

Různé:

- Ruby on Rails + Cygwin + Windows Vista (http://www.youtube.com/watch?v=mWHdxN86n0Q) 8:48 [2008-12-02]
- Linux GUI Programming with Ruby (http://www.youtube.com/watch?v=PXpwC1o5AcI) 9:56 [2007-05-21]
- Ruby GUI programming with Shoes (http://www.youtube.com/watch?v=PoZ9bPQ13Dk) 9:59 [2009-06-08]

Kapitola 3. Začínáme

Abychom si mohli vše postupně zkoušet, seznámíme se nejdříve s irb. Irb, celým názvem Interaktivní Ruby je program ve kterém můžeme podobně jako v shellu pracovat v ruby. Můžeme tedy příkazy zadávat z konzoly a dostanem zpět ihned odpověď. Pokud si potřebujeme něco vyzkoušet před tím, než to napíšeme do programu, je to ideální způsob.

```
$ irb
irb(main):001:0>
```

Opisovat všechny příkazy pokaždé do irb není zrovna pohodlné. Proto si ukážeme jak vytvořit rychle a jednoduše ruby script. Použijeme k tomu jakýkoliv textový editor jako je vi či emacs. Script/program musí začínat řádkem podle kterého operační systém pozná že se jedná o program v ruby a bude vědět jak ho spustit. Já používám jeden z univerzálních spouštěcích řádků.

```
#!/usr/bin/env ruby
```

Vice je popsáno v A.8.4 ale toto nám pro začátek bude stačit.

Kapitola 4. Seznámení s jazykem

* *chapter id="seznameni_s_jazykem" condition="author"*
* *Protože kapitola Sprovozňujeme ruby bude přesunuta na konec knihy mezi dodatky, je třeba zde krátce popsat spouštění **irb** abychom si hned mohli všechno odzkoušet.*
* *Zvážit zdali by nebylo vhodné přeměnit jednotlivé sekce nebo skupiny sekcí na samostatné kapitoly v části I – „Tutoriál“ v Ruby.*

4.1. Začínáme
První kontakt

* *section*
* *Protože kapitola Sprovozňujeme ruby bude přesunuta na konec knihy mezi dodatky, je třeba zde krátce popsat spouštění **irb** abychom si hned mohli všechno odzkoušet.*
* *Uvedení do problematiky spouštení ruby*
  Úvod do Ruby začnu jednoduchou aplikací na které si předvedeme jak **ruby** spustit. Napíšeme si ted známou „aplikaci“ `hello.rb`

```
puts "Hello world!"
```

A hned si ji vyzkoušíme

```
$ ruby example/tutorial/hello.rb
Hello world!
```

Zkoušení ruby tímto způsobem, kdy si napíšeme krátký program a ten spouštíme je trochu neohrabané. Obzvláště když máme k dispozici nástroj irb. Irb je interaktivní ruby, a jak již název připomínám, pracujeme s ním interaktivně. Tedy přímo zadáváme příkazy a hned vidíme výsledky.

Protože pro přímé hraní si s jazykem je interpret ruby poněkud neohrabaný, seznámíme se s programem irb. IRB je zkratka z *Interactive RuBy*, tedy interaktivní ruby. Jedná se o skript, program psaný v ruby, který usnadňuje interaktivní práci a hraní si s Ruby. Pro velkou část příkladů a ukázek v této knize byl použit právě irb.

Irb spouštíme

```
$ irb [přepínače] [program] [argumenty_programu]
```

Po spuštění vypíše program výzvu a očekává od nás příkaz

```
$ irb
irb(main):001:0>
```

Po každém příkazu vypíše jeho hodnotu/návratovou hodnotu a opět nás požádá o další příkaz.

```
irb(main):001:0> 23 * 3
=> 69
irb(main):002:0>
```

```
$ irb
irb(main):001:0> 6 * 7
42
irb(main):002:0> quit
$
```

* ***FIXME:****Tuto ukázku odstranit.*

```
# File: session/tutorial-6x7.ses
<prompt>$</prompt> <command>irb</command>
irb(main):001:0> 6 * 7
42
irb(main):002:0> quit
```

Na ukázce je vidět jak spouštíme irb, jak zadáváme příkazy a na konci je vidět příkaz **quit** kterým práci s irb ukončíme.

irb tedy můžeme použít jako kalkulačku.

4.2. Klíčová slova a identifikátory

Nejdříve seznam klíčových slov. To jsou slova, které mají v Ruby nějaký význam sama o sobě jako části jazykových konstrukcí a podobně.

```
__LINE__      __ENCODING__  __FILE__      BEGIN         END           alias
and           begin         break         case          class         def
defined?      do            else          elsif         end           ensure
false         for           if            in            module        next
nil           not           or            redo          rescue        retry
return        self          super         then          true          undef
unless        until         when          while         yield
```

Mimo tato klíčová slova jsou zde ještě 3 slova která rozeznává parser ruby.

```
=begin        =end          __END__
```

Ruby 1.9 přidává klíčová slova:

Klíčová slova nemůžeme použít jako názvy proměnných, tříd, konstatn ani metod. Jsou to vyhrazená slova jenž mají přiřazený význam definicí jazyka Ruby.

Identifikátory jsou názvy různých objektů, proměnných, metod, tříd a podobně. Na identifikátory každé z uvedených kategorii jsou kladeny podobné ale mírně odlišné nároky. Pokud vezmu za základ identifikátor lokální proměnné, mohu popsat ostatní identifikátory pomocí odlišností od identifikátoru lokální proměnné.

Takže nejdřív tedy identifikátor lokální proměnné. Tento sestává z poslopnosti znaků které mohou být číslice (0-9) malá (a-z) a velká (A-Z) písmena a znaku _. Prvním znakem identifikátoru musí být malé písmeno nebo znak _. Regulární výraz popisující identifikátor:

[a-z_][0-9a-zA-Z_]*

Ukázky identifikátorů lokální proměnné:

```
alfa
anObject
posledni_hodnota
_ident
a25
```

Následující nejsou identifikátory lokální proměnné:

```
34a             # nezačíná malým písmenem nebo znakem _
```

```
Beta            # musí začínat malým písmenem
po$ledni        # znak $ nepatří mezi povolené znaky identifikátoru
předek          # znak ř nepatří mezi povolené znaky identifikátoru
```

Nyní, když tedy víme jak vypadá identifikátor (název) lokální proměnné, popíšeme si ve zkratce identifikátory ostatních objektů.

Identifikátor globální proměnné vypadá stejně jako identifikátor lokální proměnné, jen je před něj přidán znak $.

```
$hlavni_hodnota
$rozmer_okna
```

Identifikátor proměnných objektu, tedy proměnných instance třídy jsou opět stejné jako identifikátory lokální proměnné, jen je před ně přidán znak @.

```
@barva_pozadi
@delta_x
```

Identifikátor proměnné třídy je opět stejný jako identifikáto lokální proměnné, jen je před něj přidána dvojce znaků @@.

```
@@pocet_instanci
```

Pro názvy konstant a tříd platí stejná pravidla. Jejich identifikátory musí začínat velikým písmenem.

```
TcpServer
PI
Hradlo
```

Názvy metod jsou opět stejné jako názvy lokálních proměnných. Mám však navíc možnost použít jako poslední znak identifikátoru znak ? nebo !. Použití těchto znaků má zvláštní význam pro programátora, nikoliv pro ruby. Je dobrým zvykem, pojmenovávat metody (funkce) které vrací logickou hodnotu s otazníkem na konci. Vykřičník používáme zase tam, kde metoda provádí změny v objektu. Viz například rozdíl mezi metodami `strip` a `strip!` ve třídě `String`.

```
index
posledni?
zmen!
pridej_novy
```

4.3. Komentáře a vložená dokumentace

* ***Attributy:*** *id="comments"*

Odkazy:

- RDoc

Komentáře se v Ruby zapisují pomocí znaku #. Vše od tohoto znaku až do konce řádku je komentář. Komentáře mohou být tedy jak na samostatných řádcích.

```
# Toto je komentář na samostatném řádku.
# Následovaný dalším řádkem s komentářem.
```

Kometáře mohou být taky na koncích řádků s programem.

```
def fact n
```

```
    case n
        when 0,1: 1            # ošetření speciálních hodnot
        else n * (fact n-1)    # rekurze
    end
end
```

> **Varování**
>
> V ukázce je použit poetický zápis parametrů funkcí a neobvyklé závorkování.

Připomínám, pokud to není zcela zjevné, že pokud je znak # použit v zápisu řetězce tak není otevíracím znakem komentáře, rovněž pokud je použit v zápisu regulárního výrazu.

```
a = "sd#gf"
b = /#neco/
```

Zápis delších komentářů, a také pro vložené dokumentace se provádí pomocí `=begin` a `=end`.

```
=begin
Zde je velmi dlouhý
mnohařádkový komentář.
A nebo také vložená dokumentace.
=end
```

Připomínám že `=begin` a `=end` musí být na začátku řádku.

Za klíčovými slovy `=begin` a `=end` může být libovolný text, musí ovšem být oddělen alespoň jednou mezerou.

4.4. Proměnné

* ***Attributy:*** *id="variables"*

Odkazy:

- Proměnná (http://cs.wikipedia.org/wiki/Proměnná) na Wikipedii
-

Proměnná je úložiště, tedy část paměti, do které se ukládá hodnota. Toto je velmi jednoduchá definice proměnné ale pro začátek nám postačí. Do proměnné, tedy do části paměti, můžeme ukládat různé hodnoty. Typy těchto hodnot, tedy to jestli ukládáme celé číslo, znak, řetezec znaků, nebo jiný druh objektů není proměnnou nijak omezeno. Proměnnou není třeba nijak předem deklarovat, vzniká automaticky, v okamžiku kdy do ní uložíme nějakou hodnotu.

```
a = 14
a = 'Ahoj'
a = [1, 'a', :got, "ola"]
```

Všechna tato přiřazení jsou správná. V každém okamžiku tedy proměnná a obsahuje hodnotu jiného typu.

Máme několik druhů proměnných, lišících se rozsahem platnosti, tedy v kterých částech programu platí a můžeme je používat. Tyto „druhy“ proměnných se liší tím že před samotný název proměnné jsou přidávány speciální znaky, viz. tabulka.

obor platnosti	ukázka názvu
globální	`$varname`

obor platnosti	ukázka názvu
lokální	`varname`
atributy objektu	`@varname`
atributy třídy	`@@varname`

V Ruby má typ hodnota v proměnné, nikoliv proměnná. Do proměnné mohu přiřazovat hodnoty různého typu. Pokud potřebujeme vědět jakého typu je hodnota v proměnné, zeptáme se metodou `class`.

```
$ irb
irb(main):001:0> a = 14
=> 14
irb(main):002:0> a.class
=> Fixnum
irb(main):003:0> a = 'Ahoj'
=> "Ahoj"
irb(main):004:0> a.class
=> String
irb(main):005:0> a = [1, 'a', :got, "ola"]
=> [1, "a", :got, "ola"]
irb(main):006:0> a.class
=> Array
irb(main):007:0>
```

4.5. Konstanty

* ***Attributy:*** *id="Constants*

Odkazy:

- Ruby: Constant values (http://blog.jayfields.com/2006/12/ruby-constant-values.html)
- RUBY CONSTANTS (http://rubylearning.com/satishtalim/ruby_constants.html)
- Ruby Predefined Constants (http://www.tutorialspoint.com/ruby/ruby_predefined_constants.htm)
-

Konstanty v ruby musí začínat velikým písmenem.

`ARGF`

FIXME:

`ARGV`

Pole obsahující parametry příkazového řádku. Na tuto konstantu se dá také odkazovat jménem `$*`.

```
#!/usr/bin/env ruby
puts ARGV.inspect
```

Více v 8.

RUBY_PLATFORM
PLATFORM

RUBY_RELEASE_DATE
RELEASE_DATE

RUBY_VERSION
VERSION

Řetezec obashující číslo verze ruby.

```
$ irb
irb(main):001:0> puts VERSION
1.8.7
=> nil
```

4.6. Metody

* ***FIXME:****Popsat: definici, použití, argumenty*

4.7. Operátory

* ***FIXME:***

4.8. Regulární výrazy

* ***Attributy:*** *id="rexexp"*

FIXME:

```
action if a =~ //

a = Regexp.new('^\s*[a-z]')
b = /^\s*[a-z]/
c = %r{^\s*[a-z]}
```

Modifikátory výrazů

- `i` ignoruje velikost písmen ve výrazu
- `o` vykoná substituci výrazů jen jednou
- `m` mnohořádkový mód (tečka zastupuje i nový řádek)
- `x` rozšířený regex (připouští bílé mezery a komentáře

Symboly používané v regulárních výrazech

- `^` začátek řádku nebo řetězce
- `$` konec řádku nebo řetězce
- . libovolný/jakýkoliv znak z výjimkou nového řádku (mimo POSIX)
- \w znak slova (číslice, písmeno nebo podtržítko „_“)
- \W nikoliv znak slova
- \s bílá mezera (mezara, tabulátor, nový řádek, atd ...)
- \S nikoliv bílá mezera
- \d číslice (stejné jako `[0-9]`)
- \D nikoliv číslice
- \A začátek řetězce
- \Z konec řetězce, nebo před koncem řádku na konci
- \z konec řetězce
- \b hranice slov (jen mimo `[ ]`)
- \B nikoliv hranice slov

- \b BackSpace (jen v `[ ]`)
- [ ] jakákoliv množina znaků
- * žádný nebo více předchozích výrazů
- *?
- +
- +?
- {m,n}
- {m,n}?
- ?
- |
- ( )
- (?#)
- `[[:print:]]`
- `[[:digit:]]` stejné jako `[0-9]`
- `[[:name:]]`
- `[[:alpha:]]`

Proměnné

- `$' $POSTMATCH`
- `$` `$PREMATCH`
- `$1 ... $9`.

Třída regulárních výrazů `Regexp`

Konstanty

`EXTENDED`

FIXME: Ignore spaces and newlines in regexp.

`IGNORECASE`

FIXME: Matches are case insensitive.

`MULTILINE`

FIXME: Newlines treated as any other character.

Tabulka 4-1. Volby u regulárních výrazů (*options*)

číslo	modifikátor	konstanta	poznámka
0			
1	i	IGNORECASE	
2	x	EXTENDED	
4	m	MULTILINE	
16	n		
32	e		
48	s		
64	u		
	o		

Metody

`new`(string [, options [, lang]]]) => regexp
`new`(regexp) => regexp
`compile`(string [, options [, lang]]]) => regexp
`compile`(regexp) => regexp

FIXME:

```
r1 = Regexp.new('^a-z+:\\s+\w+')        >> /^a-z+:\s+\w+/
r2 = Regexp.new(r1, true)               >> /^a-z+:\s+\w+/i
r3 = Regexp.new(r2, Regexp::EXTENDED)   >> /^a-z+:\s+\w+/
```

`escape`
`quote`

FIXME:

`union`

FIXME:

`eql?`
`==`

FIXME:

`===`

FIXME:

`match`(str) => matchdata or nil

FIXME:

`casefold?` => true or false

FIXME:

`hash` => fixnum

FIXME:

`inspect` => string

FIXME:

`kcode` => str

FIXME:

`options` => fixnum

FIXME:

`source` => str

FIXME:

`to_s` => str

FIXME:

~ => integer or nil

FIXME:

Dále již jen příklad, příklady, příklady a zase příklady.

4.9. Cykly

* *Obecné povídání o cyklech*

Jednoduchý příklad cyklu s použitím iterátoru.

```
# $Id: loop-1.ses,v 1.1 2002/12/16 20:34:13 radek Exp $
10.times do |number|
    puts "count = #{number}"
end
count = 0
count = 1
count = 2
count = 3
count = 4
count = 5
count = 6
count = 7
count = 8
count = 9
10
```

Jednou z důležitých konstrukcí jazyka je konstrukce pro opakované vykonání kódu, konstrukce nazývaná cyklus, nebo smyčka. Takovou konstrukce lze v Ruby zapsat několika způsoby. Mám k dispozici cyklus s podmínkou na začátku který můžeme zapsata s pomocí klíčových slov *while*, *until*, *do* a *end*.

```
i=0
while i < 4 do
    print i
    i += 1
end
```

nebo

```
i = 0
until i >= 4 do
    print i
    i += 1
end
```

Máme také konstrukci pro „nekonečný“ cyklus.

```
i = 0
loop do
    print i
    i += 1
end
```

Rovněž cyklus typu for

```
for v in 1..3 do
    print v
end
```

Mimo tyto jazykové konstrukce má řada objektů iterační metody které jsou snadno a intuitivně použitelné.

4.9.1. Cyklus s podmínkou

*

4.9.2. Nekonečný cyklus

*

```
loop do
end
```

4.10. `while` / `until`

* *Obecné povídání o cyklech*

FIXE:

4.11. while / until

```
while condition
    commands
end
```

```
puts ’napis slovo konec pro ukonceni’
while (line=gets.chomp)!=’konec’
    puts "napsal jsi #{line}"
end
```

break

: přerušení cyklu

next

: skok na konec cyklu

redo

: opakování bez znovuvyhodnocení podmínky

retry

: opakování se znovuvyhodnocením podmínky

```
while gets
```

```
    # ...
end

until playList.duration > 60
    playList.add(songList.pop)
end

a *= 2 while a < 100
```

4.12. Cyklus for ... in

* *Obecné povídání o cyklech*

FIXE:

```
for aSong in songList
    aSong.play
end
```

je ekvivalnetní

```
songList.each do |aSong|
    aSong.play
end
```

4.13. Výjimky

* *Povídání o výjimkách*
* *FIXME: Vytvořit samostatnou kapitolu „Chyby“. Část výjimky pak bude její částí „Výjimky a jejich ošetření.“*

```
begin
rescue
[rescue]*
else
ensure
end
```

```
catch (:done) do
    ...
       throw :done
    ...
end
```

4.13.1. Použití výjimek

4.13.1.1. for .. else .. end

```
j = 6
   catch(:out) do
      for i in 1..10
         throw :out if i == j
```

```
        end
        puts "after"
    end
```

4.14. Výjimky

4.14.1. begin .. rescue .. else .. ensure .. end

```
begin # nebo tělo metody
    raise "moje výjimka"
    puts ’tohle se nespustí’
rescue
    puts "nastala výjimka: #{$!}"
ensure
    puts ’vždy spušteno’
end

opFile = File.open(opName, "w")
begin
    # Exceptions raised by this code will be caught
    # by the following rescue clause
    while data = socket.read(512)
        opFile.write(data)
    end

rescue SystemCallError
    $stderr.print "IO failed: " + $!
    opFile.close
    File.delete(opName)
    raise
end

begin
    eval string
rescue SytaxError, NameError => boom
    print "String doesn’t compile: " + boom
rescue StandardError => bang
    print "Error running script: " + bang
end

f = File.open("testfile")
begin
    # .. process
rescue
    # .. handle error
ensure
    f.close unless f.nil?
end

f = File.open("testfile")
begin
    # .. process
```

```
rescue
    # .. handle error
else
    puts "Congratulations -- no errors!"
ensure
    f.close unless f.nil?
end
```

4.14.2. Catch a Throw

* *FIXME:*

4.15. Bloky

* *FIXME:*

Jednotlivé příkazy združujeme do podle potřeby do bloků. Blok pak vystupuje v roli příkazu v jazykových konstrukcích. Blok se ohraničuje buď to složenými závorkami

```
{ ... }
```

nebo klíčovými slovy `do` a `end`

```
do
    club.enrol(person)
    person.socialize
end
```

Oba způsoby zápisu bloku jsou ekvivalentní.

4.15.1. Příklady použití iterátorů

Podle dopisu v ML `ruby-talk(51000)` od Christiana Szegedy

The Ruby way (via yield, simply linked list): (A complete implemenatation for a change):

Příklad 4-1. A

```
class List
  def add(obj)
    @first = [@first,obj]
  end

  def each
    e = @first
    while e
      yield e[1]
      e = e[0]
    end
  end
end

l = List.new
```

```
l.add("hello")
l.add("world")

l.each do |s|
  puts s
end
```

The yield in the "each" function calls the block for each object of the list. The example above prints:

```
hello
world
```

Drawbacks: none.

Příklad 4-2. Example B): Traversing a data structure recursively:

```
class Array
  def traverse_rec
    for e in self do
      if e.kind_of? Array
        e.traverse_rec
      else
        yield e
      end
    end
  end
end

[ [1,2] [1,3,[4,[5] ],2,[3,4] ] ].traverse_rec do |el|
   puts el;
end
```

outputs all elementes of the tree recursively. (The tree is implemented as an array of (possible arrays of (possible ....))...)

Do something with each element of the a data structure (minor variation of example 1):

Příklad 4-3. Example C)

```
class List
  def add(obj)
    @first = [@first,obj]
  end

  def map
    e = @first
    while e
      e[1] = yield e[1]
    end
  end
end

l = List.new

l.add("hello")
l.add("world")
```

```
l.map do |s|
  s.length
end
```

Each element of the list is replaced by its length.

Do something according to a dynamic criterion (Sum up all elements of a list conditionally)

Příklad 4-4. Example D)

```
class List
  def add(*objs)
    objs.each do |obj|
       @first = [@first,obj]
    end
  end

  def sum_if
    e = @first
    sum = 0;
    while e
      sum += e[1] if yield e[1]
    end
  end
end

l = List.new
l.add(1,3,4,5,6,2,1,6);

l.add_if do |i|
  (i%3)==0
end
```

The last function sums all elements of the list which are divisible by 3. I hope it gives you some insides about the power of yield.

Regards, Christian

4.15.2. Použití iterátoru

4.15.2.1. Použití iterátoru

```
From: "Hal E. Fulton" hal9000@hypermetrics.comHere’s a way to look at it.
 ruby-talk(51035)
Do you find "each" useful? And other iterators?

Have you ever wanted to write an iterator for a class
of your own (that didn’t inherit from a class already
having one)?

In a case like that, you would use yield.

Have you ever wanted to write (what I call) a "non-iterating
iterator" that does some housekeeping, calls a block, and
```

```
does more housekeeping? Then you would use yield.

Definitive examples of the non-iterating iterator:
  File.open("file") { block }  # open, do block, close
  Mutex.synchronize { block }  # grab, do block, release
  Dir.chdir("dir")  { block }  # cd, do block, cd back

If you don't understand how this works, try this:

def do_something
  puts "Before"
  yield  # Basically call the block
  puts "After"
end

do_something { puts "I'm doing something" }

Output:
  Before
  I'm doing something
  After

Secondly, note that if you want an "iterating" iterator, you
would do a yield inside a loop. Control is traded back and
forth between the iterator and its block.

Thirdly, note that yield returns a value (which is the last
expression evaluated inside the block). This is useful for
things like Enumerable#select and others.
```

4.16. Iterátory

* *FIXME: dopsat*

```
{ |i| puts i }

3.times do
    print "Ho! "
end

0.upto(9) do |x|
    print x, " "
end

0.step(12,3) {|x| print x, " "}

[1, 1, 2, 3, 5].each {|val| print val, " "}
```

4.16.1. Co je to iterátor?

Iterátor je metoda která jako parametr akceptuje blok nebo objekt třídy `Proc`. Blok kódu se aplikuje na vybrané prvky.

```
kolekce.vyber do |element|
  # použití elementu
end
```

Iterátor se používá k realizaci uživatelsky definovaných řídicích struktur, obzvláště cyklů.

Ukažme si jednoduchý příklad:

```
# File: session/iterator-1.ses
data = [ 'první', 'druhý', 'třetí' ]
["prvn\303\255", "druh\303\275", "t\305\231et\303\255"]
data.each do |slovo|
    puts slovo
end
první
druhý
třetí
["prvn\303\255", "druh\303\275", "t\305\231et\303\255"]
```

Metodě `each` objektu `data` jenž je třídy `Array` je předán blok. Tedy kód uvedený mezi `do ... end`. Metoda tento blok opakovaně vykonává pro každý prvek pole. Tento prvek pak předává do bloku jako parametr *`slovo`*.

> **Varování**
>
> V některých případech mají konstrukce bloku `do ... end` a `{ ... }` odlišný význam.
>
> ```
> foobar a, b do ... end   # foobar je iterátor
> foobar a, b { ... }       # b je iterátor
> ```
>
> Toto je způsobeno odlišnou prioritou `{ ... }`. První případ je ekvivalentní `foobar(a, b) do ... end` zatímco druhý `foobar(a, b { ... })`.

4.16.2. Vytvoření nového iterátoru

Jsou tří způsoby jak blok předaný metodě použít (volat):

1. pomocí klíčového slova `yield`
2. voláním pomocí metody `call`
3. použitím `Proc.new` následovaného `call`

Příkaz `yield` volá blok. Do bloku můžeme předat parametry:

```
def myIterator
    yield 1,2
end
myIterator { |a,b| puts a, b }
```

```
def myIterator(&b)
    b.call(2,3)
end
myIterator { |a,b| puts a, b }
```

```
def myIterator
    Proc.new.call(3,4)
    proc.call(4,5)
    lambda.call(5,6)
end
myIterator { |a,b| puts a, b }
```

Metoda se může pomocí volání `block_given?` dotázat, zdali jí byl předán blok.

4.16.3. Nezapracované příklady

* *condition="author"*

```
class C
    def each(&block)
        @myarray.each(&block)
    end
end
```

4.17. Objekty a třídy

Zjednodušený zápis definice třídy vypadá takto

```
class jméno_třídy
 def název metody
        příkazy  # tělo metody
    end
    ... definice dalších metod
end
```

Jak je i na tomto zjednodušeném příkladu vidět, definujeme jen metody, nikoliv atributy objektu.

K dispozici máme několik konstruktorů přistupových metod pro atributy objektu. Ve zkratce jsou to

- `attr_reader` - vytváří metodu pro čtení atributu
- `attr_writer` - vytváří zápisovou metodu pro atribut
- `attr_accessor` - vytváří jak metodu pro zápis tak pro čtení atributu
- `attr` *`atribut`* [, true/false] - vatváří přístupovou metodu pro čtení a je-li druhý parametr `true`, tak i zápisovou metodu pro atribut. Je ekvivalentní kódu

  ```
  attr_reader attribute; attr_writer attribute if writable
  ```

Zjednodušené zavedení atributů instance a jejich přístupových metod.

```
class Song
    attr_reader :name
    attr_writer :duration
    attr        :volume
    attr_accessor :date, :symptom, :solution
    attr_.....
end
```

Použití konstruktoru `attr_accessor`

```
class Obj
    attr_accessor :foo
end
```

je ekvivalentní definici metod `foo` a `foo=`

```
class Obj
    def foo
        return @foo
    end
    def foo=(newValue)
        @foo = newValue
    end
end
```

4.17.1. Standardní metody objektu

Objekty jsou vytvářeny voláním metody `new` třídy.

Nově vytvořené objekty se inicalizují metodou `initialize` objektu.

```
def initialize (value)
    ... udělej něco s hodnotou value
end
```

4.17.2. Viditelnost metod

Řízení přístupu k metodám objektu *Access Control*

Při návrhu rozhraní třídy můžeme určit jak mnoho, a jakým způsobem má být viditelné pro okolní svět.

K dispozici máme tři úrovně ochrany metod.

*public *wordasword**

veřejné metody, mohou být volány kýmkoliv. Toto je implicitní ochrana všech metod s výjimkou metody `initialize`, která je vždy soukromá (*private*)

protected

chráněná metoda, není pro svět viditelná. Je přístupná jen pro ostatní metody v právě definované třídě a pro metody podtříd. Tedy tříd jenž jsou v dědické linii definované třídy.

private

soukromé metody, nejsou viditelné pro vnější svět. FIXME: doplnit

Poznámka: Ruby se liší od ostatních OO jazyků v jedné důležité věci. Přístupová ochrana je zajišťována dynamicky, za běhu programu, nikoliv staticky.

Při zápisu třídy se používaji pro urční ochrany kličová slova `protected`, `private` a `public`

```
class Aclass
    def method1 ...
  protected
    def protm1 ...
    def protm2 ...
  private
    def privm1 ...
    def privm2 ...
  public
    def pubm1 ...
end
```

Uvedený zápis je ekvivalentní zápisu

```
class Aclass
    def method1 ...
    def protm1 ...
    ...

    public :method1, :pubm1
    protected :protm1, :protm2
    private :privm1, :privm2
end
```

4.17.3. Supertřída `Class`

- Programming Ruby, class Class (http://www.rubycentral.com/book/ref_c_class.html)

Třídy v Ruby jsou objekty první kategorie. Každá je instancí třídy `Class`.

Když vytváříme novou třídu (typicky konstrukcí

```
class Name
    ...
end
```

je vytvořen objekt třídy `Class` a přiřazen do globální konstanty (v tomto případě `Name`).

Příklad 4-5. Předefinování metody `new` třídy `Class`

```
class Class
    alias oldNew new
    def new(*args)
        print "Creating a new ", self.name, "\n"
        oldNew(*args)
    end
end

class Name
end

n = Name.new
```

```
# produces
Creating a new Name
```

Chráněné a veřejné metody

```
class Aclass
    protected
    def faclass1
        puts "faclass1"
    end
    public
    def faclass2
        puts "faclass2"
    end
end
```

Metody třídy

- `inherited`*`aSubClass`*
- `new`(*`aSuperClass`*`=Object`)

Metody instance

- `new`([args]) ⟶ anObject

  Vytváří nový objekt třídy, poté zavolá metodu `initialize` tohoto objektu a předá jí parametry `args`.
- `superclass` ⟶ aSuperClass or `nil`

  Vrací rodičovskou třídu nebo nil.

4.17.3.1. Definice vlastního makra attr_...

* *Title:Definice vlastního „makra“ `attr_...`*

```
#!/usr/bin/env ruby
# $Id: attr_list_accessor.rb,v 1.1 2005/10/04 08:52:07 radek Exp $
# $Source: /home/radek/cvs/ruby-book/example/attr_list_accessor.rb,v $
#- Copyright (C) 2003 Radek Hnilica

class Class
    def attr_list_accessor (*symbols)
 symbols.each do |s|
     class_eval <<-EOS
         def add_#{s}(elem)
      (@#{s} ||= []) << elem
         end
         def each_#{s}(&block)
      (@#{s} ||= []).each(&block)
  end
     EOS
 end
    end
end


class Test
    attr_list_accessor :foo, :bar
end
```

4.17.4. Třída Object

* *Title: Třída `Object`*

Metody instance

- ...
- `==`
- `===`
- `=~`
- `__id__` ⟶ aFixnum

  Synonymum pro `Object#id`.
- `__send__`(aSymbol [, args]+) ⟶ anObject

  Synonymum pro `Object#send`.
- `class`
- `clone`
- `display`
- `dup`
- `eql?`
- `equal?`
- `extend`
- `freeze`
- `frozen?`
- `hash`
- `id`
- `inspect`
- `instance_eval`
- `instance_of?`
- `instance_variables`
- `is_a?`
- `kind_of?`
- `method`
- `method_missing`
- `methods`
- `nil?`
- `private_methods`
- `protected_methods`
- `public_methods`
- `respond_to?`
- `send`
- `singleton_methods`
- `taint`
- `tainted?`
- `to_a`
- `to_s`
- `type`
- `untaint`

4.17.5. Metody třídy

Metody třídy jsou metody samotné třídy nikoliv jejich objektů.

```
# File: session/class-methods.ses
class Test
    def Test.hello
        puts "Hello world!"
    end
end
nil

Test.hello
Hello world!
nil

# File: session/class-methods2.ses
class Test
    def self.hello
        puts "Hello world!"
    end
end
nil

Test.hello
Hello world!
nil
```

4.18. Pokračování (*continuation*)

* *section id="continuation" xreflabel="Pokračování"*

Odkazy a zdroje:

- A page about call/cc (http://www.eleves.ens.fr:8080/home/madore/computers/callcc.html)
- Call With Current Continuation (http://c2.com/cgi/wiki?CallWithCurrentContinuation)
- RubyGarden: Continuation (http://www.rubygarden.org/ruby?Continuations)
- Programming Ruby: class Continuation (http://www.rubycentral.com/book/ref_c_continuation.html)
- Ruby call/cc (http://merd.net/pixel/language-study/various/callcc/ruby.html)
- Ruby Buzz Forum (http://www.artima.com/forums/flat.jsp?forum=123&thread=8359)
- Kata Two Worked -- Call/CC (http://onestepback.org/index.cgi/Tech/Programming/Kata/KataTwoCallCC.rdoc)
- FAQtotum: 11.5 How can I use continuatinos (http://www.rubygarden.org/iowa/faqtotum)
- The Same Fringe Problem (http://onestepback.org/articles/same_fringe/index.html)
- Continuations Made Simple and Illustrated (http://www.ps.uni-sb.de/~duchier/python/continuations.html) by Denys Duchier
- ___ ()
- ___ ()

ToDo

1. První úkol.

Co je to pokračování?

- ... je to zbytek výpočtu
- ... je to výpočet ketrý se stane s výsledkem výrazu

```
    x + y
```

Pokračování x říká
... vezmi jeho hodnotu a přidej k ní y
fn v => v + y

Volání s konkrétním pokračováním je metoda Kernelu (jádro Ruby).

```
callcc {|cont| block } -> anObject
```

Volání vytváří objekt třídy `Continuation` který předá asociovanému bloku (*`block`*). Zavoláme-li v bloku `cont.call` způsobí návrat z bloku a předání řízení za blok. Vrácená hodnota buď'to hodnota bloku, nebo je to hodnota předaná volání `cont.call`.

Continuation objects are generated by Kernel#callcc. They hold a return address and execution context, allowing a nonlocal return to the end of the callcc block from anywhere within a program. Continuations are somewhat analogous to a structured version of C's setjmp/longjmp (although they contain more state, so you might consider them closer to threads).

Ruby's continuations allow you to create object representing a place in a Ruby program, and then return to that place at any time (even if it has apparently gone out of scope). continuations can be used to implement complex control strucures, but are typically more useful as ways of confusing people.

Continuations come out of a style of programming called Continuation Passing Style (CPS) where the continuation to a function is explicitly passed as an argument to the function. A continuation is essentially the code that will be executed when the function returns. By explicitly capturing a continuation and passing it as an argument, a normal recursive function can be turned in to a tail recursive function and there are interesting optimizations that can be done at that point. Dan Sugalski has some writeups in his "What the Heck is ... " series at: http://www.sidhe.org/~dan/blog.

Since a continuation is related to a function invocation, when you ask for a continuation object you need to specify which function invocation the continuation is for. callcc addresses this by invoking the block, and passing the continuation of the block's invocation to the block itself. Since callcc "knows" it needs the continuation before the block is invoked, I suspect that it might be easier for the implementor than if the continuation of just *any* function invocation could be grabbed.

```
# Smple Producer/Consumer
# Usage: count(limit)

def count_task(count, consumer)
    (1..count).each do |i|
        callcc {|cc| consumer.call c, i }
    end
    nil
end

def print_task()
    producer, i = callcc { |cc| return cc }
    print "#{i} "
    callcc { |cc| producer.call }
end

def count(limit)
    count_task(limit, print_task())
    print "\n"
end
```

4.18.1. Ukázka

Ukážeme si volání s aktuálním pokračováním ja příkladu jednoduché rekurzivní funkce, faktoriálu. Faktoriál čísla `n` pro `n > 0` je definován vzorcem `n * f(n-1)`.

```
def fact(n)
    if n == 0
        1
    else
        n * fact(n-1)
    end
end
```

Výpočet můžeme zachytit v proceduře

```
proc {|res| n * res }
```

Jednou zachycený výpočet můžeme předat v kódu dál.

```
fact(n-1, proc {|res| done.call(n * res)})
```

```
def fact_cps(n, done)
    if n == 0
        done.call(1)
    else
        fact_cps(n-1, proc {|res| done.call(res * n)})
    end
end

call_with_current_continuation { |current_continuation|
    puts "This will be printed"
    current_continuation.call
    puts "This will never get printed"
}
```

4.18.1.1. Jednoduchý příklad

```
def level3(cont)
    cont.call("RETURN THIS")
end

def level2(cont)
    level3(cont)
    return "NEVER RETURNED"
end

def top_level_function
    callcc { |cc|
        level2(cc)
    }
end

answer = top_level_function
puts answer
```

```
# $Id: callcc1.ses,v 1.1 2003/11/30 12:32:45 radek Exp $
def level3(cont)
    cont.call("RETURN THIS")
end
nil
def level2(cont)
    level3(cont)
    return "NEVER RETURNED"
end
nil
def top_level_function
    callcc { |cc|
        level2(cc)
    }
end
nil
answer = top_level_function
"RETURN THIS"
puts answer
RETURN THIS
nil



# Setup the call with the top level continuations.  Notice that we
# create two continuations in this function.  The outer-most one
# (+ret+) is the normal return.  The inner continuation (+failed+)
# is designed to indicate failure by returning a -1 from the top
# level function

def chop_with_cc(target, list)
    callcc { |ret|
        callcc { |failed|
        sub_chop_with_cc(target, list, ret, failed)
        }
        -1
    }
end

# Recursive helper function with continuations explicitly passed in.

def sub_chop_with_cc(target, list, found, not_found)
    if list.size <= 1
        (list[0] == target) ? found.call(0) : not_found.call
    else
        mid = list.size/2
        if list[mid] > target
            sub_chop_with_cc(
                target,
                list[0...mid],
                found,
                not_found)
        else
            found.call(
                mid + callcc { |cc|
                    sub_chop_with_cc(
                        target,
```

```
                        list[mid..-1],
                        cc,
                        not_found)
                }
            )
        end
    end
end

class TestChopContinuations < Test::Unit::TestCase
    alias chop chop_with_cc
    include ChopTests
end
```

4.18.2. Příklad Producent/Konzument

```
# -----------------------------------------------
# Simple Producer/Consumer
# -----------------------------------------------
# Connect a simple counting task and a printing task
# together using continuations.
#
# Usage:  count(limit)

def count_task(count, consumer)
    (1..count).each do
        |i|
        callcc {|cc| consumer.call cc, i }
    end
    nil
end

def print_task()
    producer, i = callcc { |cc| return cc }
    print "#{i} "
    callcc { |cc| producer.call }
end

def count(limit)
    count_task(limit, print_task())
    print "\n"
end

# -----------------------------------------------
# Filtering Out Multiples of a Given Number
# -----------------------------------------------
# Create a filter that is both a consumer and producer.
# Insert it between the counting task and the printing task.
#
# Usage:  omit (2, limit)

def filter_task(factor, consumer)
    producer, i = callcc { |cc| return cc }
    if (i%factor) != 0 then
```

```
            callcc { |cc| consumer.call cc, i }
        end
        producer.call
    end

    def omit(factor, limit)
        printer = print_task()
        filter = filter_task(factor, printer)
        count_task(limit, filter)
        print "\n"
    end

    # ----------------------------------------------
    # Prime Number Generator
    # ----------------------------------------------
    # Create a prime number generator. When a new prime
    # number is discovered, dynamically add a new multiple
    # filter to the chain of producers and consumers.
    #
    # Usage:  primes (limit)

    def prime_task(consumer)
        producer, i = callcc { |cc| return cc }
        if i >= 2 then
            callcc { |cc| consumer.call cc, i }
            consumer = filter_task(i, consumer)
        end
        producer.call
    end

    def primes(limit)
        printer = print_task()
        primes = prime_task(printer)
        count_task(limit, primes)
        print "\n"
    end
```

4.18.3. Kooperující procedury

```
    class Coroutine
        def initialize(data = {})
            @data = data
            callcc do |@continue|
                return
            end
            yield self
        end

        attr_reader :stopped

        def run
            callcc do |@stopped|
                continue
```

```
        end
    end

    def stop
        @stopped.call
    end

    def resume(other)
        callcc do |@continue|
            other.continue(self)
        end
    end

    def continue(from = nil)
        @stopped = from.stopped if not @stopped and from
        @continue.call
    end

    def [](name)
        @data[name]
    end

    def []=(name, value)
        @data[name] = value
    end

    protected :stopped, :continue

end

count = (ARGV.shift || 1000).to_i
input = (1..count).map { (rand * 10000).round.to_f / 100}
Producer = Coroutine.new do |me|
    loop do
        1.upto(6) do
            me[:last_input] = input.shift
            me.resume(Printer)
        end
        input.shift # discard every seventh input number
    end
end
Printer = Coroutine.new do |me|
    loop do
        1.upto(8) do
            me.resume(Producer)
            if Producer[:last_input]
                print Producer[:last_input], "\t"
                Producer[:last_input] = nil
            end
            me.resume(Controller)
        end
        puts
    end
end
Controller = Coroutine.new do |me|
    until input.empty? do
        me.resume(Printer)
```

```
    end
end
Controller.run
puts
```

4.18.4. Resumable functions

Callcc můžeme využít k vytváření „obnovitelných/přerušitelných“ funkcí. Následující funkce vrátí hodnotu 1. Když ji obnovíme a vrátí hodnotu 2.

```
def resumable
    callcc do |continuation|
        $resume_point = continuation
        return 1
    end
    return 2
end

x = resumable
puts "Again, X = #{x}"
$resume_point.call if x != 2
```

Při spuštění vytiskne:

```
Again, X = 1
Again, X = 2
```

4.18.5. Ukázky užití `callcc`

```
arr = [ "Freddie", "Herbie", "Ron", "Max", "Ringo" ]
callcc{|$cc|}
puts(message = arr.shift)
$cc.call unless message =~ /Max/

Freddie
Herbie
Ron
Max
```

```
callcc {|cont|
  for i in 0..4
    print "\n#{i}: "
    for j in i*5...(i+1)*5
      cont.call() if j == 17
      printf "%3d", j
    end
  end
}
print "\n"

0:   0  1  2  3  4
```

```
1:   5  6  7  8  9
2:  10 11 12 13 14
3:  15 16
```

4.18.6. Příklady

http://merd.net/pixel/language-study/various/callcc/ruby.listing

```
# dumb examples
p callcc{|k| "foo"}  # => "foo"

p callcc{|k| k.call("foo")} # => "foo"
                                # same as above since
                                # callcc{|k| expr}  <=>  callcc{|k| k.call(expr)}

p callcc{|k|
  k.call("foo")
  raise "ignored"
}    # => "foo"
                                # everything after the call to "k" is ignored

p "foo " + callcc{|k| "bar "} + "boo" # => "foo bar boo"

# imperative constructs

# "return"
def inv(v)
  callcc{|myreturn|
    p "doing things"
    myreturn.call(0) if v == 0 # special case for v = 0
    p "otherwise doing other things"
    1.0 / v
  }
end

# "goto"
p "doing things"
label_here = nil
# creating a label here
callcc{|k| label_here = k}
p "doing other things"
label_here.call
p "this won't be reached"


# "goto" v.2
def mygoto(continuation)
  continuation.call(continuation)
end

p "doing things"
label_here = callcc{|k| k}
p "doing other things"
mygoto(label_here)
```

```
p "this won't be reached"

# returning a special value (dropping the stack of computations)

# return the first i where l[i] == e
def listindex(e, l)
  callcc{|not_found|  # using not_found for getting out of listindex
           # without computing the +1's
    loop = proc{|l|
      if l == [] then not_found.call(nil)
      elsif e == l[0] then 0
      else 1 + loop.call(l[1..-1])
      end
    }
    loop.call(l)
  }
end

def product(l)
  callcc{|mybreak|
    loop = proc{|l|
      if l == [] then 1
      elsif l[0] == 0 then mybreak.call(0) # using "break" as an optimization to drop the comp
      else l[0] * loop.call(l[1..-1])
      end
    }
    loop.call(l)
  }
end

# "delay"ing and coroutines (inspired by http://okmij.org/ftp/Scheme/enumerators-callcc.html)
################################################################################

# first here is an imperative generator (a dumb one)
generate = proc{|f|
  (0 .. 10).each{|i| print "." ; f.call(i) }
}

# we want to use it functionnally
# for this, we generate the list out of the generator
def generator2list(generator)
  l = []
  generator.call(proc{|e| l << e})
  l
end

l = generator2list(generate)
print "l is #{l}\n"

# now, we want to create the list lazily, on demand.
# the generator2list above can't do this

# here is another version of generator2list that uses a coroutine to create the result
def generator2list_(generator)
  callcc{|k_main|
    generator.call proc{|e|
      callcc{|k_reenter|
 k_main.call [e] + callcc{|k_new_main|
```

```
       k_main = k_new_main
       k_reenter.call
     }
      }
    }
    k_main.call []
  }
end   l = generator2list_(generate)
print "l is #{l}\n"

# the advantage of the callcc version above is that it's easy to generate the list lazily
def generator2lazy_list(generator)
  proc{
    callcc{|k_main|
      generator.call proc{|e|
        callcc{|k_reenter|
         k_main.call(e, proc{callcc{|k_new_main|
                 k_main = k_new_main
                 k_reenter.call
    }})
        }
      }
      k_main.call nil
    }
  }
end

def lazy_list2list(lz)
  l = []
  while lz = lz.call
    (e, lz) = lz
    l << e
  end
  l
end

lz = generator2lazy_list(generate)
print "lz is"
print " #{lazy_list2list(lz)}\n"

# weird examples
p callcc{|mygoto|
  mystart, mynext, mylast =
    proc{print "start " ; mygoto.call(mynext)},
    proc{print "next "  ; mygoto.call(mylast)},
    proc{print "last"   ; "done"}
  mystart
}.call          # => returns "done", displays "start next last"
```

4.19. Makra

Čtenář který je obeznámen s jinými programovacími jazyky většinou zná nějaký systém maker či preprocesor. Makra jsou v makroassemblerech, realizuje je **cpp** preprocesor jazyka C, v Lispu či Scheme jsou přímo součástí jazyka. Řada jiných jazyků má svůj systém maker či je používána s externím preprocesorem maker jako je například **m4**. V Ruby žádný takový makrojazyk není, ale stejně jako v Lispu je možné Ruby rozšířit, modifikovat a vytvořit si tak obdobu maker. V povídání o třídách objektů 4.17 jsme již takováto „makra“ použili. Jednalo se o konstruktory přístupových metod k proměnným třídy `attr_accesor`, `attr_reader` a `attr_writer`.

Nyní si ukážeme jak si vytvořit vlastní konstruktory metod podobného druhu.

4.20. Moduly

FIXME:

```
module RDODB
    VERSION = "0.0.0pre1"
end
```

Kapitola 5. Datové typy

* *chapter id="data-types"*

Popis základních datových typů. Jedná se o jednoduché, dále nedělitelné, „atomické“ typy. Jedinou výjimkou jsou řetězce znaků.

5.1. Čísla

* *section id="cisla"*
* *Popis čísel a číselných výrazů. Zápis čísel. Operace s čísly. `ruby` jako kalkulátor*

Prvním objektem se kterým se seznámíme jsou čísla. Ruby podporuje práci s celými čísly i s čísly s plovoucí řádovou čárkou. Celá čísla jsou s neomezenou přesností.

5.1.1. Celá čísla

* ***FIXME:****dopsat*

Celá čísla v Ruby jsou celými čísly tak jak je znáte ze školy a jiných programovacích jazyků. Běžně jsou čísla v rozsahu od -2^{30} do $2^{30}-1$ a nebo od -2^{62} do $2^{62}-1$ (dle implementace) reprezentována v binární formě a jsou objekty třídy `Fixnum`. Celá čísla mimo tento rozsah jsou uložena v objektech třídy `Bignum` a implementována jako řada čísel proměnné délky. Převod čísla třídy `Fixnum` na číslo třídy `Bignum` a zpět se děje zcela transparentně dle potřeb, a programátor se o ně nemusí starat.

Celá čísla zapisujeme jako posloupnost znaků `0-9`, _ a znaménaka `-`. Znak podtžítko _ slouží jen k optickému oddělení skupin čísel libovolné délky a nemá žádný jiný význam. Umožňuje nám místo `6589245` psát mnohem přehledněji `6_589_245`.

Mimo zápis dekadický, který znáte ze školy, můžete použít zápis hexadecimální. Hexadecimální zápis čísla začíná obdobně jako v jazyce C posloupností znaků `0x`, můžeme psát i s velkým `X` `0X`. Rovněž stejně jako v jazyce C je možno psát oktalově, číslo pak začíná nulou `0`. Mimo tyto způsoby zápisu čísla známé z jazyka C existuje ještě možnost zapsat číslo binárně jako posloupnost nul a jedniček. Takováto posloupnost pak začíná `0b` nebo `0B`.

Krátký přehled zápisu čísel

- `456` — celé číslo
- `-367` — záporné celé číslo
- `1_099_511_627_776` — velké celé číslo s podtžítky pro lepší čtení
- `0377` — číslo zadané v osmičkové soustavě, poznáme podle vedoucí nuly.
- `0xFF` — to stejné číslo v šestnáctkové (hexadecimální) soustavě, poznáme podle předpony `0x` nebo `0X`, pro zápis můžeme použít velká i malá písmena
- `0b10101010` — číslo `170`v dvojkové (binární) soustavě, poznáme podle předpony `0b` nebo `0B`

Podtržítko pro lepší čitelnost velikých čísel můžeme použít také v ostantích zápisech, nikoliv jen dekadickém. Například `0xAB20_BFF0_0FFF`, `0b10101011_00100000`.

```
# $Id: priklady-zapisu-celych-cisel.ses,v 1.1 2002/12/16 20:34:13 radek Exp $
irb(main):001:0> 123456
```

```
123456
irb(main):002:0> 123_456
123456
irb(main):003:0> -539
-539
irb(main):004:0> 123_456_789_012_345_678_901
123456789012345678901
irb(main):005:0> 0xFD
253
irb(main):006:0> 0x2569_fd56
627703126
irb(main):007:0> 0377
255
irb(main):008:0> 0b111_010
58
irb(main):009:0>
```

S celými čísly můžeme provádět běžné operace tak jak jsme zvyklí z jiných programovacích jazyků, nebo ze školy. Můžeme je sčítat:

```
# $Id: scitani-celych-cisel.ses,v 1.1 2003/11/19 23:54:35 radek Exp $
456_322 + 32
456354
3478 + 342 + 0xFF
4075
```

odčítat

```
# $Id: odcitani-celych-cisel.ses,v 1.1 2003/11/19 23:54:35 radek Exp $
22 - 32
-10
0xFF - 0b1010
245
```

násobit

```
# $Id: nasobeni-celych-cisel.ses,v 1.1 2003/11/19 23:54:35 radek Exp $
16 * 8
128
0x100 * 10
2560
```

a dělit

```
# $Id: deleni-celych-cisel.ses,v 1.1 2003/11/19 23:54:35 radek Exp $
16 / 4
4
5 / 2
2
7 / 3
2
```

5.1.1.1. Třída `Fixnum`

* *Třídu `Fixnum` přemístnit jinam*

  Zkrácený seznam metod objektů třídy: times, type, upto, downto, step

5.1.1.1.1. Metody instance

<=>

Jméno

<=> — porovnání

Popis

FIXME: doplnit

5.1.1.2. Třída `Bignum`

* ***FIXME:** dopsat*

  Třída celých čísel s velkou přesností.

5.1.2. Čísla s pohyblivou řádovou čárkou

Dalším druhem čísel se kterými můžeme pracovat jsou desetinná čísla s pohyblivou řádovou čárkou, známá z jiných jazyků taky jako `Double`, `Float` nebo `Real`. Konstanty techto čísel zapisujeme:

- `3.1415926` — reálné číslo
- `45.411_564_126` — můžeme použít podtžítka pro zpřehlednění zápisu dlouhých čísel
- `86.7683e16` — reálné číslo v zápisu s exponenetem

* ***FIXME:** dopsat*

5.1.3. Komplexní čísla

* *section condition="author"*
* ***FIXME:** prozkoumat je-li zabudováno*

5.1.4. Operace s čísly

*

5.1.4.1. Dělení

*

Jako operátor dělení používáme znak '/'. Pokud jsou dělenec i dělitel celá čísla, je výsledek celé číslo. Pokud je jeden z obou číslo v plovoucí řádové čárce je výsledek v plovoucí řádové čárce. Pro dělení máme také několik metod.

Number.div(Number)

: Výsledkem je celé číslo.

Number.fdiv(Number)

: Výsledkem je celé číslo v plovoucí řádové čárce.

Number.quo(Number)

: Výsledkem je zlomek (racionální číslo), jestli je to možné.

S dělením souvisí operace výpočtu zbytku po celočíselném dělení, pro kterou se používá znak '%'.

Metoda `divmod` vrací jak výsledek celočíselného dělení, tak i zbytek.

```
podil,zbytek = 10.divmod 3
```

Dělíme-li 0, můžeme obdržet několik odlišných výsledků.

Fixnum / 0 ⟶ způsobí výjimku ZeroDivisionError
Float / 0 ⟶ Infinity
0.0 / 0 ⟶ NaN
0 / 0.0 ⟶ NaN

5.2. Řetězce a znaky

* *Způsoby zápisu řetězců, operace nad řetězci.*

FIXME:

Přehled zápisů znakových konstant

- `?x`
- `?\n`
- `?\\`
- `?\cd`
- `?\C-x`
- `?\M-x`
- `?\M-\C-x`

5.2.1. Řetězcové konstanty (*literály*)

Řetezcové konstnty jsou v zásadě dvojího druhu: expandující (interpolující) a neexpandující (neinterpolující). Uvnitř expandujících dochází k nahrazení posloupnosti `#{varname}` hodnotou proměnné `varname` a rovněž se používá znak '\' pro vkládání řídicích i jiných znaků. Uvnitř neexpandujících se neprovádí interpolace (expanze) řetězců začínajících znakem '#' a '\' je omezeno jen na dvě posloupnosti \\ a \'.

Nejdříve tedy popíši neexpandující řetězce (řetězcové konstanty). Tyto jsou začínají a končí znakem apostrof '. Zápis se může rozprostírat i přes několik řádků. V takovém případě je znak konce řádku součástí řetězce.

```
retezec = 'První řádek.
Druhá řádek.'

irb(main):001:0> retezec = 'První řádek.
irb(main):002:0' Druhý řádek.'
=> "Prvn\303\255 \305\231\303\241dek.\nDruh\303\275 \305\231\303\241dek."
irb(main):003:0> puts retezec
První řádek.
Druhý řádek.
=> nil
```

Pokud potřebujeme v řetězci použít znak ', můžeme jej označit pomocí zpětného lomítka. Větu "I'm ready." zapíšeme

```
irb(main):005:0> veta = 'I\'m ready.'
=> "I'm ready."
irb(main):006:0> puts veta
I'm ready.
=> nil
```

Znak obráceného lomítka má před jiným znakem obráceného lomítka také speciální význam. Je to proto, abychom mohli zapsat \'.

```
irb(main):007:0> s = '\\\"
=> "\\'"
irb(main):008:0> s = '\\'
=> "\\"
```

Expandující (interpolující) řetězce expandují (interpolují) větší sadu znaků. Jednak je to již zmíněné obrácené lomítko pomocí kterého lzě zadávat řídicí posloupnosti jako \n \a \b \f ... a expanze proměnných pomocí #.

Tabulka 5-1. Použití obráceného lomítka pro zápis znaků

posloupnost	význam
`\`*x*	Obrácené lomítko před znakem *x* mění význam je-li to jeden ze znaků: `abcefnrstuvxCM01234567\#"`
`\a`	ASCII kód 7 (BEL). Stejné jako `\C-g` nebo `\007`.
`\b`	ASCII kód 8 (BackSpace). Stejné jako `\C-h` nebo `\010`.
`\e`	ASCII kód 27 (ESC). Stejné jako `\033`.
`\f`	ASCII kód 12 (FF). Stejné jako `\C-l` nebo `\014`.
`\n`	ASCII kód 10 (NewLine). Stejné jako `\C-j` nebo `\012`.
`\r`	ASCII kód 13 (CR). Stejné jako `\C-m` nebo `\015`.
`\s`	ASCII kód 32 (SP). Vkládá mezeru, tedy stejné jako " ".
`\t`	ASCII kód 9 (TAB). Stejné jako `\C-i` nebo `\011`.

posloupnost	význam
\u*nnnn*	Unikódový znak vyjádřený čtyřmi hexadecimálními číslicemi *n*. Podporováno v Ruby 1.9 a vyšším.
\u{*hexa*}	Unikódový znak vyjádřený hexadecimálním číslem *hexa*. Ruby 1.9 a novější.
\v	ASCII kód 11 (VT). Stejné jako \C-k nebo \013.
nnn	ASCII znak jehož oktalový kód je *nnn*.
nn	Stejné jako \0*nn*.
n	Stejné jako \00*n*.
\x*nn*	ASCII znak jehož hexadecimální kód je *nn*.
\x*n*	Stejné jako \x0*n*.
\c*x*	Zkratka pro \C-x
\C-*x*	Znak jenž má v ASCII reprezentaci nejvyšší dva bity v bajtu nastaveny na 0.
\M-*x*	Znak jenž má v ASCII reprezentaci nejvyšší bit v bajtu nastaven na 1. \M může být kombinováno s \C.
\EOL	Lomítko na konci řádku spojuje řádek s následujícím. Tedy posloupnost \ a znak konce řádku je odstraněna.

Expandující jsou například:

"řetězec se substitucí #{var}"
%Q/řetězec.../
%Q:řetězec...:

Neexpandující pak:

'řetězec'
%q[taky řetězec]
%q(samozřejmě řetězec)
%q!už mě to přestává bavit!

Jak jste si na ukázkách mohli všimnout, mimo obvyklé znaky pro uvození řetězců " a ' je možnou použít %q formu. Malé q pro neexpandující a velké Q pro expandující řetězce. V tomto tvaru je řetězec vymezen znakem jenž následuje jako první za písmenem q(Q). Výjimkou jsou znaky (, [a { k nimž je odpovídající koncový znak),], nebo }.

Zvláštní formou jsou pak víceřádkové řetězcové konstanty. Tyto mají tvar

```
var = <<"EOS"
Toto je víceřádkový řetězec
se substitucí. Hodnota Proměnné
var je #{var}
EOS
```

```
irb(main):001:0> pozdrav = <<TADY + <<TAM + "World"
irb(main):002:0" Hello
irb(main):003:0" TADY
irb(main):004:0" There
irb(main):005:0" TAM
=> "Hello\nThere\nWorld"
irb(main):006:0> puts pozdrav
Hello
There
World
```

```
=> nil
```

U víceřádkových řetězcových konstant je prováděna substituce. Pokud chceme substituci potlačit, použijeme

```
# $Id: string-literal-eof2.ses,v 1.1 2003/01/14 13:37:57 radek Exp $
var = <<'EOS'
variable #{a}
EOS
"variable \#{a}\n"
p var
"variable \#{a}\n"
nil
```

Ukončovací značka víceřádkového řetězce musí být na začátku řádku. Protože nám to ne vždy vyhovuje, má ruby ještě variantu u které tato značka může být kdekoliv, ale na samostatném řádku.

```
a = <<-EOF
Toto je víceřádkový řetězec
ukončený značkou EOF na samostatném řádku.
Všiměte si, že značka není na začátku řádku.
    EOF
```

5.3. Datum a čas

Informaci o datumu a čase můžeme ukládat do znakových řetězců, či polí čísel. Ruby má ovšem pro práci s časovými údaji vhodnější nástroje ve formě několika tříd.

- `Date`
- `DateTime`
- `ParseDate`
- `Time`

Odkazy:

- Class Time (http://www.ruby-doc.org/core/classes/Time.html)

Základním objektem, vlastně hlavním, kole kterého se vše točí je třída Time. Objekty této třídy reprezentují časové údaje.

Pro vytváření a plnění objektu třídy `Time` slouží metody `new`, `at`, `utc`, `gm`, `locala` `mktime`.

Pro samotnou operaci analýzy řetezců s časovými údaji použijeme knihovnu `ParseDate`.

```
radek@yoda:~: 1 $ irb
irb(main):001:0> require 'parsedate'
=> true
⋮
irb(main):007:0> t = Time.local(*ParseDate.parsedate("2005-06-27 14:25"))
=> Mon Jun 27 14:25:00 CEST 2005

radek@yoda:~: 0 $ irb
irb(main):001:0> require 'parsedate'
=> true
irb(main):002:0> t1 = Time.local(*ParseDate.parsedate("2005-06-28 15:21:07"))
```

```
=> Tue Jun 28 15:21:07 CEST 2005
irb(main):003:0> t2 = Time.local(*ParseDate.parsedate("2005-06-28 15:28:16"))
=> Tue Jun 28 15:28:16 CEST 2005
irb(main):004:0> t2-t1
=> 429.0
irb(main):005:0>
```

Prezentaci času nám usnaduje řada konverzních metod. Jedná se o metody převádějící čas na jiné reprezentace/hodnoty to_a, to_f, to_i a to_s. Dále pak metoda `strftime` jenž převádí čas na řetězec podle zadaného formátování.

5.3.1. Třída Time

new

Jméno

`new` — Vytvoření nového objektu, objekt je inicializován aktuálním časem.

Popis

FIXME:

Time.new

to_a, to_f, to_i, to_s

Jméno

`to_a, to_f, to_i, to_s` — Vrátí reprezentaci časové informace jako pole (to_a), reálné číslo (to_f), celé číslo (to_i) nebo řetezec (to_s)

Time

Popis

FIXME:

Ukázka

```
radek@yoda:~: 0 $ irb
irb(main):001:0> t = Time.now
=> Mon Jun 27 21:29:35 CEST 2005
irb(main):002:0> t.to_s
=> "Mon Jun 27 21:29:35 CEST 2005"
irb(main):003:0> t.to_f
=> 1119900575.79187
irb(main):004:0> t.to_a
=> [35, 29, 21, 27, 6, 2005, 1, 178, true, "CEST"]
irb(main):005:0> t.to_i
=> 1119900575
```

parse

Jméno

`parse` — Vytvoření nového objektu a jeho inicializace podle obecného řetězce popisujícího datum a čas.

Popis

FIXME:

Poznámka: Podle dokumentace má třída `Time` tuto metodu znát, ale v mém případě tomu tak není. Ruby mám nainstalováno z Debian Sarge.

```
yoda:~# ruby --version
ruby 1.8.2 (2005-04-11) [i386-linux]
yoda:~# dpkg -l ruby
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Installed/Config-files/Unpacked/Failed-config/Half-installed
|/ Err?=(none)/Hold/Reinst-required/X=both-problems (Status,Err: uppercase=bad)
||/ Name           Version        Description
+++-==============-==============-============================================
ii  ruby           1.8.2-1        An interpreter of object-oriented scripting
```

Ukázka

ukázka převzatá z dokumentace.

```
# Suppose it is "Thu Nov 29 14:33:20 GMT 2001" now and
# your timezone is GMT:
Time.parse("16:30")     #=> Thu Nov 29 16:30:00 GMT 2001
Time.parse("7/23")      #=> Mon Jul 23 00:00:00 GMT 2001
Time.parse("Aug 31")    #=> Fri Aug 31 00:00:00 GMT 2001
```

Time.at

Jméno

`Time.at` — Vytvoří nový objekt inicializovaný číselnou hodnotou

Popis

Vytváří nový objekt reprezentující časový údaj. Objekt je inicializován podle zadaných číselných parametrů. První parametr udává počet sekund od záčátku epochy (nové éry). Druhý, nepovinný parametr, pak počet mikrosekund od začátku sekundy.

```
radek@yoda:~: 0 $ irb
irb(main):001:0> Time.at(123954)
=> Fri Jan 02 11:25:54 CET 1970
```

Kapitola 6. Řízení běhu programu

* *chapter id="control-structures" xreflabel="Řízení běhu programu"*

Popis konstrukcí řídících běh programu, tedy větvení a cykly.

6.1. Jednoduché větvení if

* *section id="if" xreflabel="if"*

Základním způsobem větvení programu je konstrukce **if**. V nejjednodušším tvaru vypadá například takto:

```
# $Id: tut-if.ses,v 1.1 2004/02/02 21:14:46 radek Exp $
vaha = 81
81
if vaha > 80 then
    puts "je to moc t<65533>k<65533>"
end
je to moc t<65533>k<65533>
nil
```

Při zápisu na jeden řádek můžeme použít modifikátor příkazu. Podmínku napíšeme za příkaz. Tuto vlastnost zdědil Ruby po jazyce Perl

```
puts "je to moc těžké" if vaha > 80
```

Všechny možnosti konstrukce **if** lze načrtnout takto

```
if podmínka then
    příkaz nebo příkazy
elsif další podmínak then
    příkaz nebo příkazy
else
    příkaz nebo příkazy
end
```

Přičemž část **else** může být vypuštěna. Část **elsif** může být taktéž vypuštěna, a nebo můžeme uvést více částí **elsif**.

FIXME:

```
if condition
    commands
elsif condition
    commands
else
    commands
end
```

```
if count > 10
    puts "Try again"
elsif tries == 3
    puts "You lose"
else
```

```
    puts "Enter a number"
end

unless aSong.duration > 180 then
    cost = .25
else
    cost = .35
end

if artist == "John Coltrane"
    artist = "'Trane"
end unless nicknames == "no"
```

This path leads to the gates of madness.

6.2. unless

* *section id="unless" xreflabel="unless"*

Konstrukce **unless** je identická s konstrukcí **if**, jen podmínka má jiný (opačný) význam. Příkaz/příkazy za **unless** se vykonají v případě že podmínka není splněna. Je tomu tedy přesně naopak než u **if**. Formální tvar příkazu **unless** vypadá takto:

```
unless podmínka then
    příkaz nebo příkazy
elsif další podmínak then
    příkaz nebo příkazy
else
    příkaz nebo příkazy
end
```

Stejně jako u if může být část **else** vypuštěna a část **elsif** taktéž vypuštěna nebo jich může býti více.

6.3. if a unless jako modifikátory

Příkazy if a **unless** můžeme použít jako modifikátory příkazu. V tomto případě se if/`unless` píše až za příkaz, kterého se týká, a to na stejný řádek jako tento. Tuto vlastnost má Ruby po Perlu.

Ruby má po Perlu jednu vlastnost, a tom jsou modifikátory příkazu. Jsou dva, **if** a **unless**.

```
příkaz if podmínka
příkaz unless podmínka
```

* *FIXME: dopsat, zmínit možnost `if/unles podmínka příkaz end`*

6.4. Vícenásobné větvení case

Vícenásobné větvení **case** je jistým zjednodušením příkazu **if** s větším počtem částí **elsif**. Je přehlednější při zápisu a i lépe čitelný.

Formálně vypadá zápis příkazu takto:

```
case výraz
```

```
    when hodnota
        příkaz nebo příkazy
    when jiná hodnota
        příkaz nebo příkazy
    else
        příkaz nebo příkazy
end
```

Část **else** je nepovinná a může být vypuštěna, a částí **when** může být libovolný počet.

* *Dopsat, zmínit se, že příkaz case stejně jako if funguje taky jako výraz/funkce.*

```
kind = case year
    when 1850..1889 then "Blues"
    when 1890..1902 then "Ragtime"
    else "JazzL
end

case expression
when /regularní_výraz/
    commands
when /regularní_výraz/
    commands
else
    commands
end

s = gets.chomp

case s
when /ruby/
    puts ':-)'
when /p(erl|ython)/
    puts ':-('
else
    puts 'nevim'
end

kind = case year
    when 1850..1889 then "Blues"
    when 1890..1909 then "Ragtime"
    when 1910..1929 then "New Orleans Jazz"
    when 1930..1939 then "Swing"
    when 1940..1950 then "Bebop"
    else                 "Jazz"
end
```

6.5. Vrácené hodnoty

V ruby má každý výraz vlastní hodnotu. I Konstrukce if ... then ... else ... end má vlastní hodnotu. Tohoto je možno využít v situacích jako je tato

```
v = if a > b then
    a
else
    b
```

```
    end
```

Kapitola 7. Datové struktury

* *chapter id="data-structures" xreflabel="Typy a datové struktury"*

V této kapitole podrobně popíši vše kolem datových typů a struktur.

V Ruby je vše objekt, i datové struktury jsou objekty různých tříd.

V ruby platí věta „Všechno je objekt“. Některé třídy objektů jasou ale natolik významné četností svého použití, že mají často specifický zápis použití.

7.1. Pole

- ___ (http:___)

* *Vysvětlit/zmínit se co je pole*

Pole (`Array`) je setříděná množina prvků indexovaná celými čísly. Index začínají od `0` a postupují k vyšším číslům. Záporný index je interpretován jako relativní od konce množiny. T.j. index `-1` označuje poslední prvek, index `-2` pak prvek předposlední, atd.

Než si začneme vykládat o polích, tak si dáme malou ukázku.

```
# $Id: array-intro.ses,v 1.1 2003/11/19 23:54:35 radek Exp $
[[1,2,3],[4,5,6],[7,8,9]].each {|arr| puts arr.join(',')}
1,2,3
4,5,6
7,8,9
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
$a = [[1,2,3],[4,5,6],[7,8,9]]
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
(0..2).each do |i|
  (0..2).each do |j|
    print $a[i][j]
  end
  print "\n"
end
123
456
789
0..2
```

Pole v Ruby ma širší použití než jaké známe z jazyků jako je C, Pascal. V Ruby používáme pole mimo klasického způsobu také misto seznamů (*list*) a n-tic (*tuple*). Vzhledem k častosti svého použití má vlastní syntaxi pro vytváření nových polí. Místo obvyklého

```
aArray = Array.new
```

můžeme psát

```
aArray = []
```

Prvky pole jsou číslovány celými čísly počínaje nulou. Na první prvek se tedy odkazujeme `aArray[0]`, na druhý `aArray[1]`, a tak dále. Mimo kladné celé čísla a nuly můžeme použít i záporné indexy. Tyto záporné

indexy počítají prvky pole od konce s tím rozdílem, že pro poslední prvek je použit index `-1`. Tedy `aArray[-1]` je poslední prvek v poli, `aArray[-2]` je předposlední prvek a tak dále.

Při generování tabulek pro program v C jsem potřeboval tyto pěkně formátovat. Není třeba psát algoritmus na mnoho řádků. S výhodou lze použít metody `each_slice`, která seskupí prvky pole po skupinách. Poté správnou aplikací `join` dosáhneme požadovaného efektu.

```
          puts tabulka.collect{|n| "%3d" % n.to_i} \
          .each_slice(8).collect{ |row| "\t" + row.join(', ') }.join ",\n"
```

```
irb(main):007:0> puts (1..19).each_slice(5).collect{|row| "\t" + row.join(', ')}.join ",\n"
 1, 2, 3, 4, 5,
 6, 7, 8, 9, 10,
 11, 12, 13, 14, 15,
 16, 17, 18, 19
=> nil
```

7.1.1. Literály, konstanty typu `Array`

Pro časté použití pole byly vytvořena syntaxe pro přímý zápis pole do programu. Tento má tvar:

```
aArray = [1,2,3,4]
```

Protože často potřebujeme zapisovat pole textových řetězců jejichž zápis je velmi zdlouhavý, například:

```
aArray = ["první", "druhý", "třetí"]
```

byla zavedena konstrukce `%w`. Výše uvedené pole pak zapíšeme jako

```
aArray = %w( první druhý třetí )
```

kterýžto zápis je kratší a přehlednější.

7.1.2. Přidání prvku do pole <<

```
songs << prvek
```

7.1.3. Metoda `each`

Iterace přes všechny prvky se provádí metodou `each`

```
songs.each do |song|
    # kód pracující s song
end
```

7.1.4. Nezatříděné poznámky

Při práci s poli se používají některé „zvláštní“ konstrukce. Například když potřebujeme vytvořit pole s daným obsahem jen pokud již neexistuje, použijeme operátor `||=`.

```
# $Id: array-or-create.ses,v 1.1 2005/12/05 10:19:35 radek Exp $
a ||= [0,1,2]
[0, 1, 2]
p a
[0, 1, 2]
nil
```

FIXME:

```
# $Id: array-or-create-append.ses,v 1.1 2005/12/05 10:19:35 radek Exp $
(b ||= []) << "dalsi"
["dalsi"]
(b ||= []) << "konec"
["dalsi", "konec"]
p b
["dalsi", "konec"]
nil
```

7.1.5. Třída Array

Pole (*array*) je setříděná, očíslovaná (celými čísly) množina objektů.

7.1.5.1. metody třídy

Zde uvádím všechny metody třídy `Array`.

- `[]` — vytvoření pole zaplněného objekty
- `new` — vytvoření nového pole

[]

Jméno

`[]` — Vytvoření nového pole s danými objekty.

Array

Přehled

```
[]([anObject]*);
```

Popis

Vytvoří nové pole. Toto pole může být rovnou inicializováno danými objekty.

```
Array.[](1, 'a', /^A/) ⟶ [1, "a", /^A/]
Array[1, 'a', /^A/] ⟶ 1, 'a', /^A/
[1, 'a', /^A/] ⟶ [1, 'a', /^A/]
```

new

Jméno

`new` — Vytvoření nového pole daných rozměrů. Může být inicializováno.

Array

Přehled

```
new(anInteger = 0, anObject = nil);
```

Popis

Vytvoří nové pole. Toto pole může mít nastavenu velikost a může být inicializováno jedním objektem.

```
Array.new ⟶ []
Array.new(2) ⟶ [nil, nil]
Array.new(4, "N") ⟶ ["N", "N", "N", "N"]
Array.new(3, Hash.new) ⟶ [{}, {}, {}]
```

insert

Jméno

`insert` — vsunutí prvku do pole, rozhrne stávající prvky

Popis

Vsunutí prvku do pole. Rozhrnutí.

```
# $Id: array-insert-1.7.ses,v 1.1 2002/12/16 20:34:12 radek Exp $
# Needs Ruby 1.7.x
ary = [0,1,2]
[0, 1, 2]
ary.insert(1,"a")
[0, "a", 1, 2]
p ary
[0, "a", 1, 2]
nil
```

7.1.5.2. metody instancí

- `&` — množinový průnik
- `*` — opakování
- `+` — spojování polí
- `-` — množinový rozdíl
- `<<` — připojení prvku na konec pole
- `<=>` — porovnávání polí
- `==` — porovnání dvou polí na shodu
- `===` — porovnání dvou polí
- `[]` — přístup k prvku pole (získání obsahu)
- `[]=` — přístup k prvku pole (přiřazení hodnoty)
- `|` — množinové sjednocení polí
- `assoc` — FIXME:
- `at` — vrátí prvek pole na pozici určené indexem
- `clear` — odstranění všech prvků z pole
- `collect!` — vrací pole prvků vytvořených vyvoláním bloku kód na každy prvek pole
- `compact` — vrátí pole bez prvků `nil`
- `compact!` — odstraní z pole všechny prvky `nil`
- `concat` — připojení prvků jiného pole na konec pole
- `delete` — FIXME:
- `delete_at` — FIXME:
- `delete_if` — FIXME:
- `each` — FIXME:
- `each_index` — FIXME:
- `empty?` — FIXME:
- `eql?` — FIXME:
- `fill` — FIXME:
- `first` — FIXME:
- `flatten` — FIXME:
- `flatten!` — FIXME:
- `include?` — FIXME:
- `index` — FIXME:
- `indexes` — FIXME:
- `indices` — FIXME:
- `join` — FIXME:
- `last` — FIXME:
- `length` — FIXME:
- `map!` — FIXME:
- `nitems` — FIXME:
- `pack` — FIXME:
- `pop` — FIXME:
- `push` — FIXME:
- `rassoc` — FIXME:
- `reject!` — FIXME:
- `replace` — FIXME:
- `reverse` — FIXME:
- `reverse!` — FIXME:

- `reverse_each` — FIXME:
- `rindex` — FIXME:
- `shift` — FIXME:
- `size` — FIXME:
- `slice` — FIXME:
- `slice!` — FIXME:
- `sort` — FIXME:
- `sort!` — FIXME:
- `to_a` — FIXME:
- `to_ary` — FIXME:
- `to_s` — FIXME:
- `uniq` — FIXME:
- `uniq!` — FIXME:
- `unshift` — FIXME:

amp;

Jméno

`amp;` — Průnik množin.

Array

Přehled

```
&(anOtherAray);
```

Popis

Vytvoří průnik dvou množin reprezentovaných poli.

```
[1, 1, 3, 5] & [1, 2, 3]  ⟶ [1, 3]
[1, 1, 1, 3, 2, 2] & [1, 1, 2, 2] ⟶ [1, 2]
```

*

Jméno

`*` — **FIXME:**

Array

Popis

Opakování

7.2. Enumerable

`Enumerable` není datový typ ale množina operací (mixin) nad třídami, které implementují metodu `each`. Pokud mají být použity metody `max`, `min` a `sort`, musí třída definovat smysluplně operátor `<=>`, který vlastně nad třídou definuje pořadí/řazení prvků.

Nejprve se podíváme na metody které transformují seznam na jiný seznam. Takové jsou zejména ale nejen: `collect`, `map`, `sort`

method(Array) ⟶ Array

Jedná se o operace/metody: inject, reduce, zip

Array ⟶ Value

Array,Array ⟶ Array

Array ⟶ Array,Array

```
irb(main):....> a=[1,2,3]
irb(main):....> b=%w(a b c)
irb(main):....> a.zip(b)
=> [[1, "a"], [2, "b"], [3, "c"]]
```

7.2.1. Reduce / Inject

Metoda která redukuje celou množinu objektů na jednu hodnotu. Princip redukce je algoritmus:

```
reduce (objects) :: Enumerable ⟶ Value
    accumulator = 0
    objects.each do |accumulator, object|
        accumulator += object
    end
    return accumulator
end
```

Reduce nahrazuje celý takový programový zápis. Protože v některých jazycích je známa pod názvem Inject, má ruby dva názvy reduce a inject které jsou synonyma. Z různých důvodů upřednostňuji název inject, zadáváme-li počáteční hodnotu a název reduce když tuto hodnotu nezadáváme.

```
irb(main):001:0> (1..5).reduce(:+)
=> 15
irb(main):002:0> (1..5).inject(1, :*)
=> 120
```

7.3. Asociativní pole (hash)

* ***FIXME:***

Asociativní pole, v některých jazycích nazývané *hash* je v podstatě pole indexované jakýmikoliv, tedy i nečíselnými, indexy.

```
barvy = {
    'cervena'   =>  'FF0000',
    'zelena'    =>  '00FF00',
    'modra'     =>  '0000FF',
    'bila'      =>  'FFFFFF'
}

hist = Hash.new
```

7.4. Kontrola a zjišťování typů

Potřebuji-li se zeptat zda je objekt instance konkrétního typu použiji metodu `Object::instance_of?`.

```
# $Id: typecheck.ses,v 1.1 2003/11/19 23:54:35 radek Exp $
s = "ahoj"
"ahoj"
s.instance_of? String
true
s.instance_of? Object
false
s.instance_of? Integer
false
```

Není-li můj dotaz takto konkrétní, ale potřebuji jen prostě znát typ objektu zeptám se přímo metodou `Object::type` jenž mi vrátí typ objktu jako

```
# $Id: type.ses,v 1.1 2003/11/19 23:54:35 radek Exp $
s = "ahoj"
"ahoj"
s.type
(irb):2: warning: Object#type is deprecated; use Object#class
String
1.type
(irb):3: warning: Object#type is deprecated; use Object#class
Fixnum
1.1.type
(irb):4: warning: Object#type is deprecated; use Object#class
Float
```

Jak je na ukázce vidět, je v novějších verzích ruby užití metody `Object::type` zavrženo a doporučuje se nahradit voláním metodoy `Object::class`

```
# $Id: object-class.ses,v 1.1 2003/11/30 12:32:45 radek Exp $
s = "ahoj"
"ahoj"
s.class
String
1.class
```

```
Fixnum
1.1.class
Float
```

7.4.1. Duck typing

* ***Attributy:*** *section id="duck-typing" xreflabel="Duck typing"*

What Duck typing is based mostly on realising what sort of operations you want to do with the object and testing for those operations, rather than testing for the class. As Dave is fond of saying: „type and class aren't the same".

Duck typing is based mostly on realising what sort of operations you want to do with the object and just doing them, rather than worrying if the object inherits from the proper base class or interface.

I've heard others also explain duck typing in terms of explicitly testing for particular methods and I feel that leaves the wrong impression. If we say Ruby supports duck typing, then newcomers are left with the impression that you need to do a lot of testing for particular methods (which you don't).

Ukázka Duck Typing

```
class Dog
    def talk
        puts "Woof"
    end
end
class Duck
    def talk
        puts "Quack"
    end
end
[Dog.new, Duck.new].each { |a| a.talk }
```

7.5. StrongTyping

* *section id="strongtyping" xreflabel="StrongTyping" condition="author"*

Odkazy a zdroje:

- StrongTyping 2.0 (http://mephle.org/StrongTyping)

StrongTyping is a little ruby module that provides a convenient way for ruby methods to check parameter types, and also dynamically query them. In addition to merely checking a single set of types, it allows easy overloading based on a number of different templates.

Mějme funkci

```
def foo(a, b)
    ...
 end
```

Now let's say this function wants 'a' to always be a String, and 'b' should be Numeric:

```
require 'strongtyping'
include StrongTyping

def foo(a, b)
```

```
      expect(a, String, b, Numeric)
      ...
   end
```

Overloading is just as easy:

```
   require 'strongtyping'
   include StrongTyping

   def bar(*args)
      overload(args, String, String) {
         | s1, s2 |
         ...
         return
      }

      overload(args, String, Integer) {
         | s, i |
         ...
         return
      }

      overload_default args
   end
```

* *Uvádím popis modulu `strongtyping` v kapitole Extrémní programování protože se domnívám že má k této tématice nejblíž.*

Ruby je jazyk s dynamickými typy. To znamená že typ není svázán s proměnnou/parametrem. U předávaných parametrů tedy není typ znám a když předáme metodě takový typ který není logikou zpracování očekáván nastane výjimka způsobená chybou při práci s hodnotou. Modul `StrongTyping` nám dává nástroj pro snadnou kontrolu typu předávané hodnoty.

7.5.1. Překlad a instalace

K instalaci potřebujeme **ruby** verzi 1.6 nebo 1.8, překladač jazyka C a zdrojové kódy modulu StrongTyping. Ty získáme na http://mephle.org/StrongTyping. Jak přeložit je popsáno v souboru `README.en` jenž se nachází mezi zdrojovými kódy. Ve zkratce překlad provedeme příkazy:

```
$ ruby extconf.rb
$ make
```

U instalace záleží jestli je **ruby** nainstalován uživatelsky, t.j. přeložili jsme si jej sami a nainstalovali v našem `home` adresáři, nebo jestli je nainstalován v systémových adresářích. V případě systémových adresářů musíme instalovat jako **root**.

```
$ su
# make install
```

Máme-li **ruby** instalované uživatelsky, stačí rovnou nainstalovat.

```
$ make install
```

7.5.2. Použití

Tak modul již mám připraven k použití a tak se podívám co s ním. Nejdříve jej musím importovat

```
require 'strongtyping'
include StrongTyping
```

Tím jsm si zpřístupnil metody tohoto modulu. Ve svém programu mám metodu `initialize` třídy `Cons`

```
def initialize(car, cdr)
    @car = car; @cdr = cdr
end
```

Nyní využiji z připraceného modulu metodu `expect` a zkontroluji typy předávaných parametrů *`car`* a *`cdr`*. Od těchto očekávám že jsou to objekty odvozené od třídy `SExp`

```
def initialize(car, cdr)
    expect car, SExp, cdr, SExp
    @car = car; @cdr = cdr
end
```

Pokud budou mít oba předávané parametry správny typ, nic se nestane a výpočet bude dále pokračovat. Ale pokud aspoň jeden parametr nebude mít správny typ, bude vyvolána výjimka `StrongTyping::ArgumentTypeError`.

7.5.3. Metody modulu `StrongTyping`

`StrongTyping::expect(`*`par0`*`,` *`Type0`*`[,` *`par1`*`,` *`Type1`*`[,...`*`parN`*`,` *`TypeN`*`]])`

Kontrola typu parametrů. Jako typ je možno použít název třídy nebo název modulu. Pokud bude alespoň jeden parametr nesprávného typu, metoda vyvolá výjimku typu StrongTyping::ArgumentTypeError

`overload(args, [Module0[, Module1[,...ModuleN]]]) { | o0, o1,..oN | }`

FIXME: Call the block with 'args' if they match the pattern Module0..ModuleN. The block should _always_ call return at the end.

`overload_exception(args, [Module0[,...ModuleN]]]) { | o0, o1,..oN | }`

FIXME: This acts identically to overload(), except the case specified is considered invalid, and thus not returned by get_arg_types(). It is expected that the specified block will throw an exception.

`overload_default(args)`
`overload_error(args)`

FIXME: Raise OverloadError. This should _always_ be called after the last overload() block. In addition to raising the exception, it aids in checking parameters. As of 2.0, the overload_error name is deprecated; use overload_default.

`get_arg_types(Method)`

FIXME: Return an array of parameter templates. This is an array of arrays, and will have multiple indices for functions using multiple overload() blocks.

`verify_args_for(method, args)`

FIXME: Verify the method 'method' will accept the arguments in array 'args', returning a boolean result.

Třídy reprezentující výjimky

`StrongTyping::ArgumentTypeError` < `ArgumentError`

FIXME: This exception is raised by expect() if the arguments do not match the expected types.

`StrongTyping::OverloadError` < `ArgumentTypeError`

FIXME: This exception is raised by overload_default() if no overload() template matches the given arguments.

Kapitola 8. Parametry příkazové řádky a jejich analýza

* ***Attributy:*** *id="cli-arguments"*

  Odkazy:

  -
  -
  -
  - Getopts — je součástí ruby
  - GetoptLong — je součástí ruby
  - OptionsParser
  - CommandLine::OptionParser

  Jak bylo již zmíněno v 4.5, parametry příkazové řádky jsou programu předány jako pole `ARGV`.

8.1. OptionParser

*

  Odkazy:

  - OptionParser (http://ruby-doc.org/stdlib/libdoc/optparse/rdoc/classes/OptionParser.html)
  - OptionParser- -Parsing Command-line Options the Ruby Way (http://ruby.about.com/od/advancedruby/a/optionparser.htm)
  - Using OptionParser to Parse Commands in Ruby (http://ruby.about.com/od/advancedruby/a/optionparser2.htm)
  - Ruby ARGV, options and OptionParser (http://soniahamilton.wordpress.com/2009/10/02/ruby-argv-options-and-optionparser/)
  - A script to make adding new GIT repositories easier (http://parkersmithsoftware.com/blog/post/19-a-script-to-make-adding-new-git-repositories-easier)
  -
  -

```
require 'optparse'

options = {}
OptionsParse.new do |opts|
    opts.banner = "Usage: example.rb [options]"

    opts.on("-v", "--[no-]verbose", "Run verbosely") do |v|
        options[:verbose] = v
    end
end.parse!
p options
p ARGV
```

8.2. CommandLine::OptionParser

* ***Attributy:*** *id="CommandLine.OptionParser"*

  Odkazy:

  - CommandLine::OptionParser (http://rubyforge.org/docman/view.php/632/170/index.html) by Jim Freeze [2005]

- CommandLine (http://rubyforge.org/docman/view.php/632/233/posted-docs.index.html) by Jim Freeze [2005]
-
-

8.3. Nezpracované poznámky

Odkazy:

- Ruby > Command-line option parsing (http://www.ruby-forum.com/topic/49989) — z mailové konference [2005-12-28]
-

```
# Mandatory argument.
opts.on("-r", "--require LIBRARY", "Require the LIBRARY before executing your script") do |lib
        options.library << lib
end

Optional argument; multi-line description.
opts.on("-i", "--inplace [EXTENSION]", "Edit ARGV files in place",
                                       "  (make backup if EXTENSION supplied)") do |ext|
        ...
end

# Another typical switch to print the version.
opts.on_tail("--version", "Show version") do
        puts OptionParser::Version.join('.')
        exit
end
```

V dalším emailu je následující ukázka:

```
require 'rubygems'
require 'commandline'

class MyApp < CommandLine::Application
  def initialize
    # Mandatory argument
    option :names => %w(--require -r),
               :opt_description => "Require the LIBRARY "+
                                           "before executing your
script",
               :arg_description => "LIBRARY",
               :opt_found => get_arg,
               :opt_not_found => required

    option :names => %w(--inplace -i),
               :arity => [0,1],
               :opt_description => "Edit ARGV files in place",
               :arg_description => "[EXTENSION]",
               :opt_found => get_arg,
  end

  def main
    #put your code here
```

```
    p opts
  end
end#class MyApp

opts.parse!(args)
```

Kapitola 9. Ruby a čeština

* ***FIXME:****Najit vhodnější místo pro tento text.*
* *Tento text je velmi starý. V současné době je v Rubi již podpora UNICODE.*

Měl bych zmínit vztah Ruby k češtině. Češtinou mám na mysli dvě různé věci. Ta první je použití češtiny v hodnotách, tedy v řetězcích. Tedy schopnost Ruby zpracovávat český text. Zcela určitě umí poslední verze stabilní řady, tedy v tuto chvíli verze 1.8.5, a předpokládám že i některé předchzí, pracovat s češtinou v kódování utf-8. Starší verze se zcela jiste srovnají s jakýmkoliv 8-mi bitovým kódováním kódováním češtiny jako je například ISO-8859-2 známé taky jako Latin2.

Chceme-li tedy pracovat s češtinou v UTF-8 musíme definovat odování a nahrát moduly jenž umožňují některé textové operace v utf-8.

```
$KCODE='u'
require 'jcode'
```

* ***To be done.***

Druhou věcí jenž rozumím pod pojmem schopnost pracovat s češtinou je možnost použít české a nejen české znaky mimo hodnoty. Tedy v názevech symbolů, proměnných, metod, tříd a modulů. Tento způsob umožňuje zápis jako:

```
poloměr = 23
průměr = poloměr * 2
puts "Výsledek je #{průměr}"
```

Ve vývojové řadě verzí 1.9 již byla tato možnost implementována, takže následující kód vám bude fungovat:

```
#!/usr/bin/env ruby1.9
# -*- mode:ruby; coding:utf-8; -*-
$KCODE='utf8'
require 'jcode'

symbol = :český
p symbol

proměnná = :ěščřžýáié
p proměnná
```

Použití znaků mimo standardní sadu ASCII nám otevírá nové a zcela netušené možnosti. Nicméně bychom měli přesně vědět z jakých důvodů se do těchto velmi zrádných vod pouštíme. Pokud chceme učit programování děti, můžeme tuto možnost využít. Nemusí se zatěžovat problémy s angličtinou či *cestinou*. Myslím že pro tento účel je použití národních znaků akceptovatelné a ospravedlnitelné.

Proč píši ospravedlnitelné. Inu proto že hodláme-li psát netriviální program, který dostanou k dispozici lidé na celem světě, je absolutním zvěrstvem použít národní znaky v kódu programu jako názvy proměnných a symbolů. Když pominu, že češtiny neznalý programátor takový text vnímá jako obrázkové písmo a má problém se v něm zorientovat. Tak vyvstane téměř neřešitelný problém s jeho možností takový text editovat. On zcela jistě na své národní klávesnici (například španělské) nemá žádnou možnost jak napsat písmeno "ř". Takže nemá žádnou možnost jak takový program upravovat. Když už tedy nemůžeme, například z důvodu neznalosti angličtiny používat anglické názvy, buď me k programátorům, například z brazílie, alespoň na tolik shovívaví, že použijeme místo názvů českých názvy *ceske*. Tedy bez diaktitiky.

Pokud jste pořád ještě nepochopili o čem zde píši, představte si že vám na stole (v počítači) skončí program, který by jste rádi upravili pro vlastní potřebu, a on jako naschvál je psaný v čínštině. Tedy v čínštině jsou nejen

komentáře, ale také názvy proměnných a symboly. Mějte tohle vždy na paměti, pokud vás popadne touha psát program česky.

Snad bych dodal ještě jednu věc. Pokud se chcete věnovat programování na profesionální úrovni, je znalost angličtiny stejně nezbytná, jako byla nezbytná znalost latiny ve středověku pro jakéhokoliv učence.

Pro toho kdo nepochopil malá ukázka:

* ***FIXME:****připravit ukázku ve formě obrázku. Jako text s tím mám problémy.*

```
<33394><12399> = 45 + <21250><12408>
<12393><25955><12426> = <12396><12427><12434>(<33394><12399>)
puts "#{<12393><25955><12426>}"
```

Kapitola 10. Konfigurace programu

*

Odkazy:

- Configatron: Simple, Persistent Configs for your Ruby App(s) (http://www.rubyinside.com/configatron-ruby-app-configuration-library-1130.html)
- Flexible Ruby Config Objects (http://mjijackson.com/2010/02/flexible-ruby-config-objects) [2010-02-09]
-
-

```
class Config
    def initialize(data={})
        @data = {}
        update!(data)
    end

    def update!(data)
        data.each do |key, value|
            self[key] = value
        end
    end

    def [](key)
        @data[key.to_sym]
    end

    def []=(key,value)
        if value.class == Hash
            @data[key.to_sym] = Config.new(value)
        else
            @data[key.to_sym] = value
        end
    end

    def method_missing(sym, *args)
        if sym.to_s =~ /(.+)=$/
            self[$1] = args.first
        else
            self[sym]
        end
    end
end
```

Použití:

```
config = Config.new
config.database = 'database_name'
config.username = 'user'
config.db_hosts = {
    'sj' => 'sj.example.com',
    'ny' => 'ny.example.com'
}

config.username      # => 'user'
config.db_hosts.ny   # => 'ny.example.com'
```

Použití třídy Hash místo třídy OpenStruct má své výhody hierarchických konfigurací.

```
yaml_data = "
---
database: mydb
auth:
    user: myuser
    pass: mypass
"

require 'yaml'
config = Config.new(YAML.load(yaml_data))
config.auth.user   # => "myuser"
```

Použití Configatron

```
def configatron
    @configatron ||= Configatron.new
end

configatron do |config|
    config.app_name = "My Awesome App"
    config.database_url = "postgres://localhost/somedb"
    # etc ...
end
```

Kapitola 11. Kostra aplikace

Odkazy:

- rubigen (http://rubigen.rubyforge.org/) — Ruby Generator Framework [ver=1.5.5]
-
-
-

Tato kapitola se zabývá "nezáživnými" věcmi okolo kostry aplikace.

Adresářová struktura:

```
Rakefile        --
README          -- popis aplikace
lib             -- adresář s knihovnami
test            -- adresář s testy
```

11.1. Hoe

Odkazy:

- Tutorial: Publishing RubyGems with Hoe (http://nubyonrails.com/articles/tutorial-publishing-rubygems-with-hoe)
-
-

Instalace:

```
$ gem install hoe
Successfully installed hoe-2.6.2
1 gem installed
Installing ri documentation for hoe-2.6.2...
Installing RDoc documentation for hoe-2.6.2...
```

Gem Hoe obsahuje jeden jediný spustitelný program a to program **sow**. Tímto programem se vytváří nové projekty.

11.2. Rubigen

*

Instalace:

```
$ gem install rubygen
$ cd my-app
$ install_rubigen_scripts
$ ruby script/generate
```

Kapitola 12. Práce se soubory

*

Na existenci souboru nebo adresáře se zeptáme

```
if File.exist? "cesta/k/souboru"
    # soubor existuje
end
```

Kapitola 13. Úprava a formátování kódu

*

Odkazy:

- Ruby indentation for access modifiers and their sections (http://fabiokung.com/2010/04/05/ruby-indentation-for-access-modifiers-and-their-sections/) Fabio Kung [2010-04-05]
- RUBY-STYLE (http://github.com/chneukirchen/styleguide/blob/master/RUBY-STYLE) na github
- Ruby Style Guides and Tools: How to Write Good Looking Ruby (http://www.rubyinside.com/ruby-style-guides-and-tools-how-to-write-good-looking-ruby-1272.html) [2008-10-27]
-
-
-

II. Nástroje

Seznámení s některými nástroji.

Kapitola 14. Komentování a dokumentace kódu

* *chapter id="code-documenting" xreflabel="Komentování a dokumentace kódu"*

Jednou z důležitých činností práce programátora je dokumentace a komentování kódu. Každý kdo se musel orientovat v kódu jiného, či ve svém starém kódu mi určitě dá za pravdu.

14.1. RD

* *section id="rd" xreflabel="RD"*

Odkazy a zdroje:

- RD working draft (http://www.rubyist.net/~rubikitch/RDP-en.cgi?cmd=view;name=RD)

ToDo

1. První úkol.

RD je dokumentační formát pro psaní „vnořené dokumentace“, tedy dokumentace která je součástí zdrojového kódu. V RD píšeme čistý text obohacený o několik druhů značek. Jedná se vlastně o velmi jednoduchý značkovací jazyk.

Protože píšeme dokumentaci i program v jednom, musí nějak **ruby** i další programy rozlišit text od kódu. K tomuto jsou určeny značky

```
=begin
=end
```

Vše mezi nimi je považováno za dokumentaci.

Řádky začínající ’`---`’ jsou speciálním případem labeled list, kde label je název metody a signatura.

14.1.1. Nadpisy/Titulky

První značkou kterou si popíšeme je značení nadpisů. Značka začíná na začátku řádku a skládá se z jednoho či několika znaků „=“ nebo „+“ následovaných textem. Použitý znak a počet určuje úrověň nadpisu a text je vlastním nadpisem.

1. `= Nadpis první (nejvyšší) úrovně`
2. `== Nadpis druhé úrovně`
3. `=== Nadpis třetí úrovně`
4. `==== Nadpis čtvrté úrovně`
5. `+ Nadpis páté úrovně`
6. `++ Nadpis šesté úrovně`
7. `+++ Nadpis sedmé úrovně`
8. `++++ Nadpis osmé úrovně`

14.1.2. Průběžné značení textu, zvýrazňování textu

Tento druh značení se používá upreostřed řádku v případě kdy potřebujeme zvýraznit text, vložit odkaz, ukázku kódu, název proměnné.

```
((*emphasis*))
(({ code
stuff }))
((|variable|))
((%type me%))
((:index term:))
((<reference>))
((-footnote-))
(('verbatim'))
```

14.1.3. Seznamy

```
This is normal text
* start of a
  multiline bullet item
* and another
  * nested item
  * second nested
* third item at a top level

(1) A numbered item
    * subitem in a bulleted list
    * subitem
(2) Second numbered item
(9) This will actually be labaled '3'

: red
  when the light is red, you
  must stop
: amber
  the amber light means that things are about to change. Either
  * step on the gas, or
  * slam on the brakes
: green
  green means GO
```

14.2. rdtool

* *section id="rdtool" xreflabel="RDTool"*

Odkazy a zdroje:

- RDtool v RAA (http://raa.ruby-lang.org/list.rhtml?name=rdtool)
- RDtool (http://www2.pos.to/~tosh/ruby/rdtool/)
- RD (http://www.rubyist.net/~rubikitch/RDP-en.cgi?cmd=view;name=RD)
-

ToDo

1. První úkol.

Nástroj RDTool slouží k zpracování dokumentace vložené do zdrojových souborů s ruby programem.

What is RD? RD is multipurpose documentation format created for documentating Ruby and output of Ruby world. You can embed RD into Ruby script. And RD have neat syntax which help you to read document in Ruby script. On the

other hand, RD have a feature for class reference. But RD's neat, natural, easy writing syntax appeals to some Rubyist. So, they use RD for other category of document than its original usage. Some write Web pages in RD, and translate it into HTML with formatter. If you want to know more about RD, please read RD on RDP What is RDtool? RDtool is formatter which can translate RD into HTML, man or other type of format. Although RD is neat enough for you to read, translator into HTML is sometimes useful. RDtool's develop is still continued now, while we discuss about RD still now. Tosh is its maintainer. You can download RDtool from here. And it is registed to RAA.

What is RD? RD is Ruby's POD, embeddable documentation format in script file. RD is influenced mainly from plain2, a program to translate from plain text to some mark-up language. So, RD looks like plain text, and its simpleness and neatness make it easy to read and write. How does the interpreter work for RD? Ruby's interpreter, ruby, simply ignores text between a line beginning with "=begin" and one beginning with "=end". So, RD is not only embeddable. You can write anything between =begin and =end. RD is one of them, but RD will be a standard one.*1

```
=begin
=end
```

14.3. RDoc

* ***Attributy:*** *id="rdoc" xreflabel="RDoc"*

Odkazy a zdroje:

- 4.3
- RDoc - Ruby Documentation System (http://rdoc.sourceforge.net/doc/index.html)
- RDoc na SourceForge (http://rdoc.sourceforge.net/)

Rdoc je jednoduchá aplikace jenž extrahuje ze zdrojových kódu dokumentaci. Využívá přitom značkovacího jazyka podobného RD ale jednoduššího a znalosti syntaxe jazyka Ruby.

Program **rdoc** který je součástí distribuce ruby slouží k vytváření standardní dokumentace. Dokumentace se vytváří z odpovídajícím způsobem formátovaných komentářů a samotného zdrojového kódu.

Vytváří strukturovanou HTML a XML dokumentaci ze zdrojů psaných v Ruby a z rozšíření psaných v C.

Pro zápis textu, který má být v dokumentaci platí určitá pravidla. V dokumentaci jsou texty získané z komentářů které se nachází před definicí takových částí programu jako jsou třídy a metody. V komentářích je možno použít značkovacího jazyka který v ukázce uvedu.

```
#
# Toto je první odstavec.
#
# Odstavce jsou od seb odděleny prázdnými řádky.
#
# = Nadpis
# == Podnadpis
# === Podnadpis více zanořený
# Zanoření podnadpisů může být ještě větší
#
# = Příklad
#
#  Tento text je zobrazen fontem pevné šířky a je zachováno řádkování.
#   rdoc jej pozná tak že je odsaten od začátku řádku.
#
# = Seznamy
# * Volba obyčejného seznamu je uvozena znakem '*' nebo '-'
# * Další volba
#
# 1. Číslované seznamy začínají číslem.
```

```
# 9. Na velikosti čísla ani na pořadí nezáleží.
# 1. Je možné používat pořád to samé číslo.
#
# Pro zvýraznění textu je možno použít _kurzívu_ nebo <i>je-li slov více</i>.
# Dále _tučné_ písmo nebo <b>je-li slov více</b>.
# Také +code+ když potřebujeme font s pevnou šířkou nebo <tt>je-li slov více</tt>.
#
# Slovníkový seznam se zapisuje pomocí hranatých závorek:
# [první slova] Význam termínu.
# [další] Význam termínu 'další'.
#
```

14.3.1. Ukázky užití

Vytvoření dokumentace ke knihovně ruby-lisp-0.1

```
$ rdoc -T kilmer format.rb
```

Parametr `-T klimer` určuje téma.

14.4. YARD

* ***Attributy:*** *id="YARD"*

Odkazy:

- yard (http://yardoc.org/) — yay, a documentation tool
-
-

Kapitola 15. Interaktivní dokumentace

Zdroje a odkazy:

- http://www.pragmaticprogrammer.com/ruby/downloads/ri.html
- http://bocks.psych.purdue.edu/~pizman/myri/

15.1. ri

* ***Attributy:*** *section id="ri" xreflabel="ri"*

ri is a command line tool that displays descriptions of built-in Ruby methods, classes, and modules. For methods, it shows you the calling sequence and a description. For classes and modules, it shows a synopsis along with a list of the methods the class or module implements. All information is taken from the reference section of the book Programming Ruby.

Kapitola 16. RubyGems

* *chapter id="rubygems" xreflabel="RubyGems"*

Odkazy:

- Welcome to the RubyGems Project Wiki (http://rubygems.rubyforge.org/wiki/wiki.pl?action=browse&id=RubyGems&oldid
- RubyGems na RubyForge (http://rubyforge.org/projects/rubygems/)
- Installing ruby gems in Debian with stow (http://unpluggable.com/?p=116)
- Position on RubyGems (http://pkg-ruby-extras.alioth.debian.org/rubygems.html)

Program/balíček RubyGems je pokusem jak zjednodušit instalaci doplňků a knihoven do ruby. Tedy instalace na vyšší úrovni. Taková magie kdy zadáme

```
$ gem install --remote progressbar
```

A knihovna/balíček `prograssbar` nahraje z internetu a korektně nainstaluje. Nemusíme tedy ručně provádět postup instalace, sestávající se stažení balíčku s programem/knihovnou, rozbalení do adresáře, přečtení `README` a/nebo `INSTALL` a ručního spuštění několika programů končící nainstalováním balíčku do systému.

Ještě bych zmínil kontroverzi okolo „balíčkovacího“ systému RubyGems a jeho konflikty s použitím v reálném světě. O preblémech je lépe vědět předem abychom se na ně mohli připravit.

Informace o problémech s distribucí software:

- Ruby has a distribution problem (http://www.madstop.com/ruby/ruby_has_a_distribution_problem.html)
- Ruby has a distribution problem (http://gwolf.org/node/1740)
- RubyGem is from Mars, AptGet is from Venus (http://stakeventures.com/articles/2008/12/04/rubygem-is-from-mars-aptget-is-from-venus)
- It’s just a different mindset. Not necessarily a _sane_ one, though... (http://gwolf.org/node/1869)
- Rails? Stay the f* away from my system. (http://www.grep.be/blog/en/computer/cluebat/rails_stay_away)
-

Příklad použití gemu se specifikací verze(í)

```
require ’rubygems’
gem ’RedCloth’, ’> 2.0’, ’< 4.0’
require ’RedCloth’
```

16.1. Instalace

* *section id="rubygems.install" xreflabel="Instalace"*

Pokud v systému/Ruby nemáme nainstalován samotný rubygems, nemůžeme jej při instalaci použít a musíme tedy instalovat po staru. Stáhneme tedy zdroje, napříkd z uvedené stránky na RubyForge (http://rubyforge.org/projects/rubygems/). Stažený balíček rozbalíme do pracovního adresáře, přečteme si `README` a jestli se od dob verze 0.8.6 která byla v době psaní tohoto textu (2005-03-11) aktuální nic nezměnilo, spustíme `setup.rb`.

```
$ cd ~/tmpsrc
$ tar xzvf /cesta/k/rubygems-0.8.6.tgz
$ cd rubygems-0.8.6/
$ ruby setup.rb
```

* ***To be done.***

```
$ cd /tmp
$ tar xzf cesta/k/rubygems-0.8.10.tgz
```

```
$ su root -c ruby setup.rb
Password:
...
  Successfully built RubyGem
  Name: sources
  Version: 0.0.1
  File: sources-0.0.1.gem
```

Tím jsme rubygems nainstalovali a můžeme je začít používat.

Na počítačích kde je nainstalován `rootstuff` použijeme postup

```
# cd /tmp
# tar xzf /var/lib/rootstuff/pkg/src/rubygems-0.8.11.tgz
# cd rubygems-0.8.11
# ruby setup.rb
```

Tím máme `rubygems` nainstalovány. Můžeme se o tom přesvedčit.

```
# gem --version
0.8.11
```

16.1.1. Debian 5.0 Lenny

Pokud nainstalujeme rubygems

-
-

16.2. Instalace v uživatelském prostoru

* ***Attributy:*** *section id="rubygems.install.userspace" noent*

  Odkazy:

  - Installing RubyGems (http://docs.rubygems.org/read/chapter/3)

* ***FIXME:*** *Bylo by vhodné naplánovat jinak adresáře a upravit postup na tyto nové adresáře.*

  -
  -
  -

Ne vždy máme možnost instalovat RubyGems jako správcové serveru přím do systémové části. Nebo naopak nemáme zájem modifikovat systém a chceme mít vše odděleno. Další motivací je přístup distriuce Debian, která potřebuje odlišné zacházení. Ve všech těchto případech s výhodou využijeme možnost nainstalovat si RubyGems jako obyčejný oživatel.

* *Následující postup byl proveden na Debian Lenny.*

Nejdříve nastavíme proměnné. Protože tak nechci činit pokaždé, ani to řešit přes zvláštní skripty, rozhodl jsem se nastavit si je uživatelsky v konfiguraci Bashe (../unix/bash.html). Umístnil jsem následující řádky na začátek souboru `~/.bashrc`.

```
# Local, userspace RubyGems in ~/lib/gems
export GEM_HOME=$HOME/lib/gems
export RUBYLIB=$HOME/lib/gems/lib
```

```
export PATH=$HOME/lib/gems/bin:$PATH
```

Důležité je na začátek souboru, před kód který kontroluje beží li bash v interaktivním režimu.

```
# If not running interactively, don't do anything
[ -z "$PS1" ] && return
```

Máme-li proměnné nastaveny, otevřeme si nové terminálové okno, aby se do nové instance bashe načetly naše proměnné. V této chvíli můžeme instalovat.

Stáhneme si poslední verzi RubyGems například z RubyForge (http://rubyforge.org/frs/?group_id=126), a rozbalíme do pracovního adresáře. V tomto adresáři spustíme instalaci.

* *V době psaní této poznámky to byla verze 1.3.6 ze dne 2010-02-20. Ale předtím jsem tento postup použil na verzi 1.3.5. Vše pod aktualizovaným Debian Lenny.*

```
$ mkdir src
$ cd src
$ wget http://rubyforge.org/frs/download.php/69365/rubygems-1.3.6.tgz
$ tar xzf rubygems-1.3.6.tgz
$ cd rubygems-1.3.6
$ ruby setup.rb --prefix=~/lib/gems
```

Spustitelný program se nainstaluje do `~/lib/gems/bin` pod jménem `gem1.8`. Pokud jej chceme spouštět příkazem **gem**, vytvoříme si na něj symbolický odkaz.

```
$ cd ~/lib/gems/bin
$ ln -s gem1.8 gem
```

Protože jsme si správně nastavili proměnné prostředí, je právě nainstalovaný program přímo použitelný. O tom že vše funguje se přesvědčíme tím že se jej zkusíme zeptat na číslo verze. Číslo verze které nám řekne **gem** musí být stejné jako číslo verze kterou jsme instalovali.

```
$ gem --version
1.3.6
```

16.3. Instalace v lokálním prostoru

Vytvoříme si adresář ve kterém budeme mít gemy.

```
# mkdir /usr/local/gems
```

V tomto adresáři vytvoříme skript `setvars` s následujícím obsahem:

```
#!/bin/bash
export GEM_HOME=/usr/local/gems
export RUBYLIB=$GEM_HOME/lib
export PATH=$GEM_HOME/bin:$PATH
```

Skript načteme a nastavíme si proměnné

```
# source setvars
```

Stáhneme a nainstalujeme rubygems.

```
# mkdir /usr/local/download
```

```
# cd /usr/local/download
# wget http://rubyforge.org/frs/download.php/69365/rubygems-1.3.6.tgz
# cd /usr/local/src
# tar xzvf /usr/local/download/rubygems-1.3.6.tgz
# cd rubygems-1.3.6
# ruby setup.rb --prefix=/usr/local/gems
```

Při instalaci se v adresáři `/usr/local/gems/bin` objeví program **gem1.8**. Pokud chceme, můžeme si jej „přejmenovat“ na **gem**.

```
# cd /usr/local/gems/bin
# ln -s gem1.8 gem
```

Nyní je ještě třeba každému uživateli modifikovat jeho `~/.bashrc` soubor aby ruby vědělo kde má rubygems. To udělám tak že na začátek `~/.bashrc` přesně uvedeme:

```
# Modifikace kvůli lokálním rubygems
source /usr/local/gems/setvars
```

16.4. Příkazy

Krátký přehled příkazů

* *section id="rubygems.commands" xreflabel="Příkazy RubyGEMs"*

Po nainstalování máme k dispozici dva programy (scripty). A to **gem** a **gem_server**. **gem** je skript který realizuje všechny operace s gemy. Jednotlivé operace zadáváme jako příkazy programu **gem**. Instalace uvedená na začátku kapitoly RubyGems je takovým jednoduchým příkladem užití programu **gem**. Nyní si povíme něco o příkazech jenž nám **gem** nabízí. Ale před tím, než si některé popíšeme více uvedu jejich úplný seznam s krátkými komentáři. Tento seznam je součástí programu **gem** a můžeme si jej vypsat příkazem **help commands**

```
$ gem help commands

GEM commands are:

    build             Build a gem from a gemspec
    check             Check installed gems
    environment       Display RubyGems environmental information
    help              Provide help on the 'gem' command
    install           Install a gem into the local repository
    list              Display all gems whose name starts with STRING
    query             Query gem information in local or remote repositories
    rdoc              Generates RDoc for pre-installed gems
    search            Display all gems whose name contains STRING
    specification     Display gem specification (in yaml)
    uninstall         Uninstall a gem from the local repository
    unpack            Unpack an installed gem to the current directory
    update            Upgrade currently installed gems in the local repository

For help on a particular command, use 'gem help COMMAND'.

Commands may be abbreviated, so long as they are unambiguous.
e.g. 'gem i rake' is short for 'gem install rake'.
```

Nápovědu k jednotlivým příkazům pak zobrazíme pomocí

gem help *příkaz*

Například nápovědu k update zobrazíme příkazem

```
$ gem help update
```

Tabulka 16-1. gem příkazy

build	Build a gem file from a specification
check	Check installed gems
cleanup	Cleanup old versions of installed gems in the local repository
dependency	Show the dependencies of installed gems in the local repository
environment	Display RubyGems environmental information
help	Provide help on the 'gem' command
install	Install a gem into the local repository
list	Display all gems whose name starts with STRING
query	Query gem information in local or remote repositories
rdoc	Generates RDoc for pre-installed gems
search	Display all gems whose name contains STRING
specification	Display gem specification (in yaml)
uninstall	Uninstall a gem from the local repository
unpack	Unpack an installed gem to the current directory
update	Upgrade currently installed gems in the local repository

16.4.1. update

FIXME:

16.5. Postupy

Jak řešit vybrané situace

FIXME:

Pokud potřebujeme/chceme aktualizovat gem balíčky v našem systému, použijeme příkaz update.

```
# gem update
```

```
yoda:~# gem --version
0.8.11
yoda~# gem update --system
```

FIXME:doplnit až bude vyšší verze rubygem.

16.5.1. Smazání cache

Odkazy:

- http://onestepback.org/index.cgi/Tech/Ruby/DeleteYourCache.red
- http://www.jonbaer.com/articles/2006/07/03/delete-your-rubygems-cache

Někdy se nám může stát, a mě se i stalo, že RubyGems nemůže najít balíček, nebo vrací podivné chyby. Příčinou může být poškození cache a indexu. Z tohoto stavu se zotavíme tak, že cache odstraníme. Použijeme k tomu následující postup.

```
# gem env gemdir
/usr/lib/ruby/gems/1.8
# rm /usr/lib/ruby/gems/1.8/source_cache
```

Prvním příkazem zjišťujeme kde se cache nachází a pak ji smažeme. Stejného výsledku bychom dosáhli jedním příkazem:

```
# rm $(gem env gemdir)/source_cache
```

16.5.2. Problémy po upgrade na verzi 1.0.1

Po upgrade na verzi 1.0.1 přestal fungovat program **gem**. Problém je v tom, že v něm chybí příkaz

require ’rubygems/gem_runner’

Od novější verze rubygems se tyto spouštějí už nikoliv přes příkaz **gem** ale přes příkazy **gem*verze***, kde verze je číslo verze ruby. Takže správně by v mém případě bylo používat **gem1.8**.

Pokud chci i nadále používat příkaz **gem** bude nejvhodnější odstranit jej a vytvořit jako symbolický odkaz na správnou verzi.

```
# cd /usr/bin
# mv gem gem.orig
# ln -s gem1.8 gem
```

16.6. Bundler

*

Odkazy:

- Bundler (http://gembundler.com/)
-
-

Použití ve zkratce opsané z webu Bundler (http://gembundler.com/).

```
$ gem install bundler
```

```
$ gem update --system
```

Příklad 16-1. Ukázka `Gemfile` souboru

```
source 'http://rubygems.org'
gem 'nokogiri'
gem 'rakck', '~>1.1'
gem 'rspec', :require => 'spec'
```

```
$ bundle install
$ git add Gemfile
```

Kapitola 17. Ruby Version Manager

Odkazy:

- Ruby Version Manager (http://rvm.beginrescueend.com/rvm/)
- Instalace RVM na Debian/Ubuntu (http://vaclavovic.blog.root.cz/2010/10/23/instalace-rvm-na-debian-ubuntu/)
-

Kapitola 18. Rake

Make podle Ruby

* *chapter id="rake" xreflabel="Rake" status="draft"*

Odkazy:

- RAKE — Ruby Make (http://rake.rubyforge.org/) (API Documents), aktuální verze
- Rake Documentation Home (http://docs.rubyrake.org)
- Project Page (http://rubyforge.org/projects/rake)
- SAKE BOMB (http://errtheblog.com/posts/60-sake-bomb)
- Rake (http://en.wikipedia.org/wiki/Rake_(software)) na Wikipedii
- Using the Rake Build Language (http://martinfowler.com/articles/rake.html) by Martin Flower [2005-08-10]
-
-

Rake je Make (http://www.gnu.org/software/make/) napsané v Ruby a orientované na Ruby. Při definování úloh v něm máme k dispozici celou sílu jazyka.

Rake je využito například v: Rake a Rails.

18.1. Instalace, konfigurace

FIXME:

Rake můžeme instalovat přímo ze zdrojů. Tyto získáme na RubyForge (http://rubyforge.org/projects/rake/). Zdroje stáhneme, rozbalíme a nainstalujeme.

```
$ wget ...
...
$ ruby install.rb
```

Instalace pomocí gem je stejně jednoduchá.

```
$ gem install --remote rake
```

18.2. Poznámky

Uvnitř úlohy (task) můžeme přímo volat jinou úlohu příkazem podle vzoru:

```
Rake::Task["db:migrate"].invoke
```

Úloha může bát závislá na jiných úlohách, jenž se musí vykonat před ní.

```
desc "Depends on first and second"
task :all => [:first, :second]    # V případě jedné úlohy jen => :first
    ...
end
```

Pokud potřebujeme jen vyjádřit závislost, a v úloze již neprovádíme žádné akce, můžeme vypustit `do end` blok:

```
task :all => [:first, :second]
```

Úlohy s akcemi

```
task :name [:prereq1, :prereq2] do |t|
end
```

18.3. Ukázky konfigurací

*

18.3.1. Úklid

*

Pro spouštění úklidu slouží dva cíle, *clear* a *clobber*. Do svého `Rakefile` je zavedeme příkazem:

```
require 'rake/clean'
```

Kapitola 19. Distribuce aplikací

Pokud programujeme pro někoho jiného, nebo potřebujeme přenést naše programy na jiný počítač, ocitneme se před otázkou jak přenášet / distribuovat naši aplikaci. Protože je Ruby skriptovací jazyk, nemám v něm přímo možnost vytvořit spustitelný soubor s celou aplikací jako je tomu například při vytváření aplikací v jazyce C nebo Java.

* *Ujasnit si pojmy distribution/deployment a jejich překlady do češtiny. Sjednotit s kapitolou o Capistrano.*

Odkazy k prozkoumání, software možná použitelný pro dostribuci.

- RubyScript2Exe — A Ruby Compiler (http://www.erikveen.dds.nl/rubyscript2exe/index.html) by Erik Veenstra [2007-05-29]
- Tar2RubyScript — A Tool for Distributing Ruby Applications (http://www.erikveen.dds.nl/tar2rubyscript/index.html) by Erik Veenstra [2007-05-25]
- github ryanbooker / tar2rubyscript (http://github.com/ryanbooker/tar2rubyscript) [2009-06-15]
-
-
-

19.1. RubyScript2Exe

*

Funguje tak, že v jednom jediném 'exe' souboru je zabaleno vše, konkrétní implementace ruby v četně dalších rozšíření a knihoven a soubory aplikace, datové sobory aplikace, prostě vše co je třeba k běhu aplikace. Toto je při spuštění programu rozbaleno do TEMP adresáře a spuštěno.

19.2. Crate

* ***Attributy:*** *id="Crate"*
* *Software by Jeremy Hinegardner*
* *Tento balíček/program vypadá valmi hezky, ale mám problémy s jeho sprovozněním. Navíc to vypadá že na něj už přes rok nikdo nesáhl. Použité verze ruby, knihoven a gemů jsou staré a není mi jasné jak je vyměnit.*

Odkazy:

- Packaging an Application With Crate (http://copiousfreetime.org/articles/2008/11/30/package-an-application-with-crate.html) na Copiuous Free Time [2008-11-30]
- Crate (http://copiousfreetime.rubyforge.org/crate/) — dokumentace
- github copiousfreetime / crate (http://github.com/copiousfreetime/crate) [2009-04-16]
- github halorgium / crate (http://github.com/halorgium/crate) [2009-04-16]
-
-

Bylo nutno doinstalovat balíčky

```
# aptitude install gperf
```

Data aplikace jsou uchovávána v SQLite databázích.

```
CREATE TABLE rubylibs (
```

```
        id              INTEGER PRIMARY KEY AUTOINCREMENT,
        filename        TEXT UNIQUE,
        compressed      BOOLEAN,
        contents        BLOB
);

CREATE TABLE bootstrap (
        id              INTEGER PRIMARY KEY AUTOINCREMENT,
        filename        TEXT UNIQUE,
        compressed      BOOLEAN,
        contents        BLOB
);
```

```
$ gem install crate
```

Zlib verze 1.2.3 již není na původním místě dostupný. Proto jsem jej zkusil zaměnit novější verzí 1.2.4, s tou jsem ovšem neuspěl. Nakonec se mi podařilo najít na netu tu správnou kopii původní verze 1.2.3.

```
$ $ diff -U1 recipes/zlib/zlib.rake.orig recipes/zlib/zlib.rake
--- recipes/zlib/zlib.rake.orig 2010-04-20 01:14:40.020517729 +0200
+++ recipes/zlib/zlib.rake 2010-04-20 01:42:00.079549777 +0200
@@ -4,3 +4,3 @@
 Crate::Dependency.new( "zlib", "1.2.3") do |t|
-  t.upstream_source = "http://www.zlib.net/zlib-1.2.3.tar.gz"
+  t.upstream_source = "http://downloads.sourceforge.net/project/libpng/zlib/1.2.3/zlib-1.2.3.tar.gz?use
   t.upstream_md5    = "debc62758716a169df9f62e6ab2bc634"

$ crarte -v
Crate 0.2.1
```

Na github (http://github.com)u je původní verze od autora copiousfreetime (http://github.com/copiousfreetime)(Jeremy Hinegardner) naposledy modifikovaná 2009-04-17. K této původní verzi jsem našel 3 forky od dunedain289 (), hlorgium () a dakrone ().

19.2.1. Ruční kompilace

*

Ruční vytvoření upravené verze ruby. Zkouším vše provést sám ručně, abych měl představu co se děje a byl schopen vytvořit upravené ruby i na jiné platformě či za změněných podmínek.

19.2.1.1. zlib

V původním crate se stahovala a kompilovala verze 1.2.3. Ta již na původním webu není. Není tam dokonce ani verze 1.2.4 se kterou jsem před pár týdny experimentoval. Jediná dostupná je v tuto chvílí (2010-05-10) verze 1.2.5.

```
$ mkdir download
$ cd download
$ wget http://zlib.net/zlib-1.2.5.tar.bz2
$ cd ..
$ mkdir src
$ cd src
$ tar xjvf ../download/zlib-1.2.5.tar.bz2
$ cd zlib-1.2.5
$ ./configure --64 --prefix=/usr
```

```
$ make install prefix=../../root/usr
$ make distclean
$ cd ../..
```

19.2.1.2. openssl

U OpenSSL jsem rovněž stáhl nejnovější verzi.

```
$ cd download
$ wget http://openssl.org/source/openssl-1.0.0.tar.gz
$ cd ../src
$ tar xzvf ../download/openssl-1.0.0.tar.gz
$ cd openssl-1.0.0
$ ./config --prefix=/usr zlib no-threads no-shared -fPIC
$ make depend
$ make
$ make install_sw INSTALL_PREFIX=$(pwd)/../../root
$ make clean
$ cd ../..
```

19.2.1.3. ruby 1.9

Když už jsem tak v těch nejnovějších verzích, ruby taky zkouším tu nejnovější z řady 1.9.

```
$ cd download
$ wget ftp://ftp.ruby-lang.org/pub/ruby/1.9/ruby-1.9.1-p378.tar.bz2
$ cd ../src
$ tar xjvf ../download/ruby-1.9.1-p378.tar.bz2
$ cd ruby-1.9.1-p378
$ cp ../../root/usr/lib*/{libz,libcrypto,libssl}.a .
$ export CPPFLAGS="-I../../usr/include"
$ export LDFLAGS="-L../../usr/lib -L../../usr/lib64"
$ ./configure --disable-shared --prefix=/usr --with-static-linked-ext --without-openssl --with
$ make
$ make install DESTDIR=$(pwd)/../../root
...
/usr/bin/ld: ../../../libcrypto.a(md5_dgst.o): relocation R_X86_64_32 against ‘a local symbol’ can not b
../../../libcrypto.a: could not read symbols: Bad value
collect2: ld returned 1 exit status
```

19.3. Exerb

* *MS Windows only!!!*

Odkazy:

- Exerb Project (http://exerb.sourceforge.jp/index.en.html) na japonském SourceForge
- Building Standalone FXRuby Applications with Exerb (http://lylejohnson.name/blog/2008/12/30/building-standalone-fxruby-applications-with-exerb/) na Lovable Lyle [2008-12-30]
-

Získání vývojové verze přímo z repositáře CVS. V průběhu přihlašování budeme vyzváni k zadání hesla. Zadáme prázdné heslo, tedy to jen odklepneme.

```
$ cvs -d:pserver:anonymous@cvs.sourceforge.jp:/cvsroot/exerb login
$ cvs -d:pserver:anonymous@cvs.sourceforge.jp:/cvsroot/exerb co exerb
```

19.4. OCRA

* ***Attributy:*** *id="ocra"*
* *Tento balíček se mi podařilo sprovoznit bez větších problémů. Dokonce funguje i když aplikuju na programy 21.1.*

Odkazy:

- ocra-1.1.3 Documentation (http://ocra.rubyforge.org/)
-

Vytváří spustitelné soubory pro MS Windows. Samotné vytváření spustitelných souborů musí být prováděno rovněž na stanice s MS Windows a funkčním vývojovým prostředím pro Ruby.

```
C:\> gem install ocra
```

Kapitola 20. Amalgalite

Odkazy:

- Amalgalite (http://rubyforge.org/projects/copiousfreetime/) na RubyForge
- amalgalite (http://copiousfreetime.rubyforge.org/amalgalite/)
-
-

Nástro pro zakompilování podpory pro SQLite3 přímo do ruby.

Poslední verze je 0.12.0 a komiluje ruby 1.8.6-p287.

Kapitola 21. Skrývání a zamlžování kódu

*

Odkazy:

- ZenObfuscate now available (http://blog.zenspider.com/zenobfuscate/) [2006-07-08] $2500
- ZenObfuscate by Eric Hodel and Ryan Davis
-
-

21.1. Ruby Encoder

* ***Attributy:*** *id="rubyencoder"*

Odkazy:

- rubyencoder (http://www.rubyencoder.com/)
-

Komerční produkt který provede zašifrování kódu skriptu a další změny aby nebylo možno se k tomoto kódu dostat.

Kapitola 22. Continuous Integration

* ***Attributy:*** *id="ContinuousIntegration"*

22.1. CruiseControl.rb

* ***Attributy:*** *id="CruiseControl.rb"*

  Odkazy:

  - CruiseControl.rb (http://cruisecontrolrb.thoughtworks.com) Continuous Integration in Ruby
  -

* *Mám problém s nasazením. Našel jsem stránku rake aborted! undefined method 'reenable' (https://answers.launchpad.net/ubuntu-on-rails/+question/79156) podle které to vypadá že se mi asi míchají programy.*

  ```
  $ gem install rake
  ```

III. Knihovny, technologie, postupy

V této části probereme některé věci do větší hloubky. Nezůstaneme na povrchu.

Kapitola 23. Démoni

Odkazy:

- daemon_controller: a library for robust daemon management (http://blog.phusion.nl/2008/08/25/daemon_controller-a-library-for-robust-daemon-management/)
- Create a daemon / server in 11 lines of Ruby (http://www.rubyinside.com/create-a-daemon-server-in-11-lines-of-ruby-58.html)
- Daemons Version 1.0.10 (http://daemons.rubyforge.org/)
- GOD A process Monitoring Framework in Ruby (http://god.rubyforge.org/)
- Ruby Daemons: Verifying Good Behavior (http://www.mindbucket.com/2009/02/24/ruby-daemons-verifying-good-behavior/) by Paul Bone [2009-02-24]
- Daemonizing (http://ruby-toolbox.com/categories/daemonizing.html)
- robustthread (http://github.com/luckythetourist/robustthread) by Nocolas Fouché [2009-08-21]
-

Démon je program bežící na pozadí systému který není připojen k žádnému terminálu. Podle toho jak je naprogrmován provádí v opakovaných časech nějaké akce, nebo je řízen událostmi. Klasickým případem démona je internetový superdémon inetd. Dalšími, známějšími démony jsou apache, proftpd, sshd a další. Démon nemusí být naprogramován jen v jazyce C a přeložen do binárního kódu, démona můžeme naprogramovat i v skriptovacím jazyce jako je Bash, Perl, Python a v našem případě Ruby.

23.1. simple-daemon

Odkazy:

- simple-daemon (http://github.com/bryanl/simple-daemon) na GitHub od thewoolleyman [2008-12-23]
- simple-daemon (http://simple-daemon.rubyforge.org/) na RubyForge
- Google skupina simple-daemon (http://groups.google.com/group/simple-daemon)
- jzajpt-simple-daemon (0.1.4) ()
-

```
$ gem install simple-daemon
```

23.2. Backdrop

Odkazy:

- cicloid / backdrop (http://github.com/cicloid/backdrop)
- ikanusim / backdrop (http://github.com/ikanusim/backdrop)
- peritor / backdrop (http://github.com/peritor/backdrop)
-

Má návaznosti na další gemy jako například:

```
$ gem install cicloid-backdrop --source http://gems.github.com
```

23.3. Daemons

Odkazy:
- Daemons Version 1.0.10 (http://daemons.rubyforge.org/) by Thomas Uehlinge
- Daemons Version 1.0.11 (http://github.com/ghazel/daemons/) by Greg Hazel
- Making sure Ruby Daemons die (http://blog.rapleaf.com/dev/?p=19#comment-629)
- The Daemons are Dead (long live the Daemons) (http://www.reevoo.com/labs/2009/11/the-daemons-are-dead-long-live-the-daemons/) [2009-11-24]
- Daemons Version 1.0.10 (http://github.com/shadowaspect/daemons) by Matt House [2010-02-11]
-
-

23.4. Daemonize

Odkazy:
- Writing very simple daemon in Ruby (http://blog.sosedoff.com/2009/01/24/writing-very-simple-daemon-in-ruby/) by Dan Sosedoff (2009-01-24)
-

```
require 'daemonize'
# Do stuff. When you're ready to daemonize your process:
Daemonize::daemonize
```

23.5. daemon-spawn

Odkazy:
- alexvollmer / daemon-spawn (http://github.com/alexvollmer/daemon-spawn) na github [2010-02-26]
- daemon-spawn 0.2.0 (http://rubygems.org/gems/daemon-spawn) na RubyGems.org
-

Instalaci provedem bud'to pomocí gem

```
$ gem install daemon-spawn
```

Nebo si naklonujeme zdroje přímo z GitHub

```
$ git clone http://github.com/alexvollmer/daemon-spawn.git
```

Protože jsem použil zdroje z GitHub aktuální ke dni 2010-04-09, které jsou nadepsány jako verze 0.2.0, nemusí být následující informace aktuální. Program je natolik jednoduchý že v něm mohou proběhnout velké změny.

Při startu démona pomocí metody `DaemonSpawn.start` vytvořen/otevřen deník příkazy:

```
log = File.new(daemon.log_file, "a")
log.sync = daemon.sync_log
```

Následně jsou přeotevírány standardní deskriptory:

```
STDIN.reopen "/dev/null"
STDOUT.reopen log
```

```
STDERR.reopen STDOUT
```

Tedy `STDIN` je odpojen přesměrováním na `/dev/null`, `STDOUT` je přesměrován do deníku který jsme předtím otevřeli a `STDERR` je přesměrován do `STDOUT`, tedy do stejného deníku.

23.6. Servolux

Odkazy:

- Serv-O-Lux (http://codeforpeople.rubyforge.org/servolux/) documentation
- Serv-O-Lux (http://yardoc.org/docs/TwP-servolux/file:README.rdoc)
-

Kapitola 24. Message Queue

*

24.1. Starling

*

Odkazy:

- Github starling / starling (http://github.com/starling/starling) [2010-01-20]
-

Starling - a light weight server for reliable distributed message passing.

Kapitola 25. Deníky a logování

25.1. Logging

*

Odkazy:

- Logging dokumentace (http://logging.rubyforge.org/) na Rubyforge
-

Kapitola 26. Síťové programování

*

V ruby můžem psát i síťové programy. Přímo v základu máme řadu tříd která nám to usnadní.

Tabulka 26-1.

třída	popis, určení
UDPSocket	
UDPServer	

26.1. UDP komunikace

*

```
require 'socket'
$port = 4321

sThread = Thread.start do       # run server in a thread
    server = UDPSocket.open
    server.bind(nil, $port)
    2.times {
        p server.recvfrom(64)
    }
end

# Ad-hoc client
UDPSocket.open.send("ad hoc", 0, 'localhost', $port)

# Connection vased client
sock = UDPSocket.open
sock.connect('localhost', $port)
sock.send("connection-based", 0)
sThread.join
```

26.2. HTTP/HTTPS

Odkazy:

- Custom HTTP/HTTPS GET/POST queries in Ruby (http://snippets.dzone.com/posts/show/788)
- Dokumentace k Net::HTTP (http://www.ruby-doc.org/stdlib/libdoc/net/http/rdoc/classes/Net/HTTP.html)
-
-

Ukázka použití knihovny net/http a net/https. V této ukázce posílám metodou POST data na vzdálený server. Parametr *url* obsahuje utl na které se něco posílá, například

`https://server.example.com/applikace/data`. Parametr *`vars`* pak obsahuje hash pojmenovaných hodnot.

```
require 'uri'
require 'net/http'
require 'net/https'

# Informace potřebné pro přihlášení k serveru metodou basic_auth
USERNAME = 'uzivatel'
PASSWORD = 'jehoheslo'

# POST the vars to url
def post url, vars
  uri = URI.parse url
  req = Net::HTTP::Post.new uri.path
  req.basic_auth USERNAME, PASSWORD
  req.set_form_data(vars)
  http = Net::HTTP.new uri.host, uri.port
  if uri.port == 443 then
    http.use_ssl = true
    http.verify_mode = OpenSSL::SSL::VERIFY_NONE
    #http.verify_mode = OpenSSL::SSL::VERIFY_PEER
    #http.ca_file = File.join(File.dirname(__FILE__), "cacert.pem")
    # Systémové certifikáty jsou v souborech v adresáři /etc/ssl/certs/
  end
  http.start { |http|
    res = http.request req
    puts res.body
  }
end

if $0 == __FILE__
  hodnoty_k_odeslani = {'agent' => 'true',
                        'passw' => 'heslo',
                        'teplota' => '21.4'}
  post 'https://server.example.org/teplomer/teplota', hodnoty_k_odeslani
end
```

Kapitola 27. Různé

* *chapter id="ruzne" xreflabel="Různé"*

Abstrakt kapitoly, je-li

Zde uvádím různá fakta zatím nezařazená do jiné kapitoly. Rovněž se zde mohou vyskytovat rozpracované kapitoly a podkapitoly jejichž konečným umístněním v knize si nejsem jist.

27.1. Ladění

27.1.1. Sledování běhu programu

Někdy potřebujeme sledovat stav zpracování programu v jeho průběhu. K tomuto můžeme s úspěchem použít funkci `set_trace_func` jenž nastavuje sledovací (*trace*) funkci.

```
set_trace_func proc { |event, file, line, id, binding, classname|
    printf "%8s %s:%-2d %10s %8s\n", event, file, line, id, classname
}
```

Kapitola 28. EventMachine

* ***Attributy:*** *id="EventMachine"*

  Odkazy:

  - EventMachine (http://rubyeventmachine.com/)
  - 59.7.1
  - Github eventmachine / eventmachine (http://github.com/eventmachine/eventmachine)
  - An EventMachine Tutorial (http://20bits.com/articles/an-eventmachine-tutorial/)
  - Ruby EventMachine - The Speed Demon (http://www.igvita.com/2008/05/27/ruby-eventmachine-the-speed-demon/)
  - EventMachine: scalable non-blocking i/o in ruby (http://timetobleed.com/eventmachine-scalable-non-blocking-io-in-ruby/)
  -
  -

* *EventMachine: fast, simple event-processing library for Ruby programs*

  Instalace

  ```
  $ gem install eventmachine
  ```

  Tabulka 28-1. Přehled API

EM.run	
EM.reactor_running?	
EM.stop	
EM.next_tick	
EM::TickLoop	
EM.schedule	

28.1. Obsluha spojení

*

  Odkazy:

  - General Introduction (http://wiki.github.com/eventmachine/eventmachine/general-introduction)
  -

  Obsluha spojení v bloku.

```
EventMachine::connect '0.0.0.0', 3210 do |connection|
    def connection.receive_data(data)
        p data
    end
end
```

  Obsluha spojenní definovaná v modulu. Tento způsob je nejvariabilnější pro budoucí rozšíření.

```
module EchoServer
    def receive_data(data)
        p data
        p get_peername[2,6].unpack "nC4"
```

```
            send_data "odpoved"
        end
    end

    EventMachine::connect ’0.0.0.0’, 3210, EchoServer
```

Obsluha spojení zapsaná v třídě.

```
    class EchoServer < EventMachine::Connection
        def initialize(*args)
            super
            # naše inicializace
        end

        def receive_data(data)
            p data
        end
    end

    EventMachine::connect ’0.0.0.0’, 3210, EchoServer
```

28.2. Časovače

*

Použití časovače blokem

```
    time = Time.now
    EventMachine::add_timer(1) do
        puts "Ahoj, jednu vteřinu po #{time}!"
    end
```

EventMachine::Timer.new(1, proc {puts ’hoj’})

28.3. Ukázky použití

*

```
    #!/usr/bin/env ruby
    require ’rubygems’
    require ’eventmachine’

    module PongServer
      def post_init
        puts "client connected!"
      end

      def receive_data(data)
        p data
        p get_peername[2,6].unpack "nC4"
        send_data "pong\n"
      end
```

```
end

EM.run do
  # EventMachine.epoll
  EM.open_datagram_socket '0.0.0.0', 3178, PongServer
  puts 'running Pong on port 3178'
end
```

28.4. Vybrané moduly, třídy a metody `EventMachine`

*

28.4.1. `EventMachine::Connection`

* ***Attributy:*** *id="EventMachine.Connection"*

```
class Echo < EventMachine::Connection
    def initialize(*args)
        super
        # stuff here...
    end

    def receive_data(data)
        p data
        send_data data
        close_connection_after_writing
    end

    def unbind
        p ' connection totally closed'
    end
end
```

28.4.2. open_datagram_socket

*

open_datagram_socket(address, port, handler=nil, *args)

Připojí *`handler`* k obsluze přicházejících UDP paketů na adrese *`address`* a UDP portu *`port`*.

Handler může být například modul. Pak v tomto modulu můžeme definovat metody:

- `receive_data -- Connection#receive_data`

V modulu můžeme používat volání:

- `send_data(data) -- Connection#send_data`
- `send_datagram(data, recipient_address, recipient_port)`
- `get_peername`

*

* *Connection#send_datagram(data, recipient_address, recipient_port)*

Kapitola 29. Přehled jazyka

Text kapitoly před sekcemi.

29.1. Konfigurace aktuálně spuštěného programu `ruby`

Jedná se o konfigurační parametry známé a zadávané v době překladu `ruby`. Tyto parametry jsou uloženy v modulu `rbconfig` a jsou nám přístupny po zadání `require 'rbconfig'`

```
FIXME: vložit malou ukázku
```

```
require "rbconfig.rb"
include Config
CONFIG["host"] ? "i686-pc-linux"
CONFIG["LDFLAGS"] ? "-rdynamic"
```

Hodnoty těchto parametrů a jejich úplný seznam je jsou v souboru `.../lib/ruby/1.8/i586-linux/rbconfig.rb`. Protože jejich výčet je dlouhý, uvedu zde jen některé.

CONFIG['MAJOR'], CONFIG['MINOR'], CONFIG['TEENY']

Hlavní, vedlejší číslo verze a *patchlevel* instalovaného ruby. Například ve stabilní verzi `1.8.0` mají tyto parametry hodnoty:

```
# $Id: rbconfig-version.ses,v 1.1 2003/01/22 21:15:37 radek Exp $
require 'rbconfig'
true
%w(MAJOR MINOR TEENY).each do |var|
    p Config::CONFIG[var]
end
"1"
"8"
"7"
["MAJOR", "MINOR", "TEENY"]
```

`CONFIG['DESTDIR']`

FIXME:

`CONFIG['srcdir']`

FIXME:

`CONFIG['prefix']`

FIXME:

`CONFIG['ruby_install_name']`

FIXME:

`CONFIG['SHELL']`

FIXME:

Kapitola 30. Operátory

Popis všech operátorů.

30.1. Unární operátor *

Symbol unárního operátoru `*` zastupuje dva unární operátory *splat* a *unsplat*. Pokud je použit při definici metody, má význam operátoru *unsplat*. Způsobí že do argumentu označeném tímto operátorem, který musí být posledním argumentem, se dosadí pole vytvořené ze všech zbylých argumentů při volání metody.

```
def bar first, *rest
    p first, rest
end
bar 1,2,3,4
```

```
$ irb
irb(main):001:0> def bar prvni, *zbytek
irb(main):002:1> p prvni, zbytek
irb(main):003:1> end
nil
irb(main):004:0> bar 1,2,3,4
1
[2, 3, 4]
nil
irb(main):005:0> bar 1
1
[]
nil
irb(main):006:0>
```

Při volání metody však funguje opačne, jako operátor *splat*

```
def foo a, b
    p a, b
end
foo *['don', 'key']  # je to samé jako
foo 'don', 'key'
```

30.1.1. splat, expanze polí *array expansion*

```
radek@kvark:~$ irb
irb(main):001:0> foo = [1, 2, 3, *[4, 5, 6]]
[1, 2, 3, 4, 5, 6]
irb(main):002:0>
```

Kapitola 31. Objekty a třídy

* *FIXME: ZRUŠIT!!!*

Zjednosušený zápis definice třídy vypadá takto

```
class jméno_třídy
    def název metody
        příkazy  # tělo metody
    end
    ... definice dalších metod
end
```

Jak je i na tomto zjednodušeném příkladu vidět, definujeme jen metody, nikoliv atributy objektu.

K dispozici máme několik konstruktorů přistupových metod pro atributy objektu. Ve zkratce jsou to

- `attr_reader` - vytváří metodu pro čtení atributu
- `attr_writer` - vytváří zápisovou metodu pro atribut
- `attr_accessor` - vytváří jak metodu pro zápis tak pro čtení atributu
- `attr` - ???

Zjednodušené zavedení atributů instance a jejich přístupových metod.

```
class Song
    attr_reader :name
    attr_writer :duration
    attr        :volume
    attr_accessor :date, :symptom, :solution
    attr_.....
end
```

Použití konstruktoru `attr_accessor`

```
class Obj
    attr_accessor :foo
end
```

je ekvivalentní definici metod `foo` a `foo=`

```
class Obj
    def foo
        return @foo
    end
    def foo=(newValue)
        @foo = newValue
    end
end
```

31.1. Viditelnost metod

Řízení přístupu k metodám objektu *Access Control*

Při návrhu rozhraní třídy můžeme určit jak mnoho, a jakým způsobem má být viditelné pro okolní svět.

K dispozici máme tři úrovně ochrany metod.

*public *wordasword**

veřejné metody, mohou být volány kýmkoliv. Toto je implicitní ochrana všech metod s výjimkou metody `initialize`, která je vždy soukromá (*private*)

protected

chráněná metoda, není pro svět viditelná. Je přístupná jen pro ostatní metody v právě definované třídě a pro metody podtříd. Tedy tříd jenž jsou v dědické linii definované třídy.

private

soukromé metody, nejsou viditelné pro vnější svět. FIXME: doplnit

Poznámka: Ruby se liší od ostatních OO jazyků v jedné důležité věci. Přístupová ochrana je zajišťována dynamicky, za běhu programu, nikoliv staticky.

Při zápisu třídy se používaji pro urční ochrany kličová slova `protected`, `private` a `public`

```
class Aclass
    def method1 ...
  protected
    def protm1 ...
    def protm2 ...
  private
    def privm1 ...
    def privm2 ...
  public
    def pubm1 ...
end
```

Uvedený zápis je ekvivalentní zápisu

```
class Aclass
    def method1 ...
    def protm1 ...
    ...

    public :method1, :pubm1
    protected :protm1, :protm2
    private :privm1, :privm2
end
```

31.2. Supertřída `Class`

- Programming Ruby, class Class (http://www.rubycentral.com/book/ref_c_class.html)

Třídy v Ruby jsou objekty první kategorie. Každá je instancí třídy `Class`.

Když vytváříme novou třídu (typicky konstrukcí

```
class Name
    ...
end
```

je vytvořen objekt třídy `Class` a přiřazen do globální konstanty (v tomto případě `Name`).

Příklad 31-1. Předefinování metody `new` třídy `Class`

```
class Class
    alias oldNew new
    def new(*args)
        print "Creating a new ", self.name, "\n"
        oldNew(*args)
    end
end

class Name
end

n = Name.new

# produces
Creating a new Name
```

Chráněné a veřejné metody

```
class Aclass
    protected
    def faclass1
        puts "faclass1"
    end
    public
    def faclass2
        puts "faclass2"
    end
end
```

Metody třídy

- `inherited`*`aSubClass`*
- `new`(*`aSuperClass`*`=Object`)

Metody instance

- `new`([args]) ⟶ anObject

  Vytváří nový objekt třídy, poté zavolá metodu `initialize` tohoto objektu a předá jí parametry `args`.
- `superclass` ⟶ aSuperClass or `nil`

  Vrací rodičovskou třídu nebo nil.

31.3. Třída `Object`

Metody instance

- ...
- `==`
- `===`
- `=~`
- `__id__` ⟶ aFixnum

  Synonymum pro `Object#id`.
- `__send__`(aSymbol [, args]+) ⟶ anObject

  Synonymum pro `Object#send`.
- `class`
- `clone`
- `display`
- `dup`
- `eql?`
- `equal?`
- `extend`
- `freeze`
- `frozen?`
- `hash`
- `id`
- `inspect`
- `instance_eval`
- `instance_of?`
- `instance_variables`
- `is_a?`
- `kind_of?`
- `method`
- `method_missing`
- `methods`
- `nil?`
- `private_methods`
- `protected_methods`
- `public_methods`
- `respond_to?`
- `send`
- `singleton_methods`
- `taint`
- `tainted?`
- `to_a`
- `to_s`
- `type`
- `untaint`

Kapitola 32. Vlákna

Multitasking

ftipný epygrav

Popis datových typů.

32.1. Nezpracovaný materiál

32.1.1. Ruby timer or timed event handling?

From Lyle Johnson

I’ll bet there’s a better way, but what about creating a thread sleeps thirty seconds in between doing its thing?

```
th = Thread.new do
    loop do
        puts "I just did something, going back to sleep now!
        sleep(5)
    end
end
```

Kapitola 33. Jazyk Ruby

* *chapter id="ruby-language" status="draft"*

  Odkazy:

  - **FIXME:** ()
  - **FIXME:** ()

  Experimentální kapitola. Zkouším vytvořit základní strukturu sekcí.

33.1. Přehled

* *chapter id="chapter.section.template"*

  .

33.2. Getting Ruby

.

33.3. Spouštíme ruby

* *chapter id="chapter.section.template"*

  Ruby spustíme jako jakýkoliv podobný program, **perl**, **python**, či **awk**. Do příkazového řádku napíšeme:

```
$ ruby [přepínače] skript
```

  Tímto spustíme přímo skript psaný v Ruby. My bychom si ale rádi trochu pohráli a zkoušeli ruby aniž bychom své pokusy nejdřív zaznamenávali do souborů se skripty. Pro tento případ máme k dispozici interaktivní verzi ruby, **irb**. Tato se spouští podobně, tedy

```
radek@yoda:~: 0 $ irb
irb(main):001:0> 1 + 2
=> 3
irb(main):002:0> exit
radek@yoda:~: 0 $
```

33.3.1. Ruby v interaktivním režimu

.

33.4. Creating Ruby programs

.

33.5. Ruby basics

.

33.6. Ruby language

.

33.6.1. Lexicology

.

33.6.1.1. Identifiers

.

33.6.1.2. Comments

.

33.6.1.3. Embeded Documentation

.

33.6.1.4. Reserved Words

.

33.6.1.5. Expressions

.

33.6.2. Variables and Constants

.

33.6.3. Literals

.

33.6.4. Operators

.

33.6.5. Control Structures

.

33.6.5.1. Conditional Branches

.

33.6.5.1.1. ***if***

.

33.6.6. Method Calls

.

33.6.7. Classes

.

33.6.8. Reference

.

33.7. Modules

.

Kapitola 34. Fronta zpráv (*Message Queue*)

Odkazy:

- lwqueue: Lightweight cross-language message queue system (http://www.petercooper.co.uk/archives/001236.html)
- Linux Clustering with Ruby Queue: Small is Beautiful (http://www.artima.com/rubycs/articles/rubyqueue.html)
- posix_mq - POSIX Message Queues for Ruby (http://bogomips.org/ruby_posix_mq/)
- Message queues in ruby (http://www.rubyfindings.com/2007/12/27/message-queues-in-ruby)
- Reliable Messaging for Ruby (http://labnotes.org/2005/11/17/reliable-messaging-for-ruby/) na Labnotes [2005-11-17]
- Reliable Messaging for Ruby (http://202.102.92.10/ruby/latest_gems/doc_root/reliable-msg-1.1.0/rdoc/index.html)
-

Kapitola 35. Extrémní programování

* ***Attributy:*** *id="xp" xreflabel="Extrémní programování"*

  Odkazy, zdroje:

  - XP in Cincinnati (http://onestepback.org/articles/tdddemo/fulltoc.html)

  Poznámka: FIXME: Pár slov o extrémním programovaní jako takovém.

* *para condition="author"*

  Extrémní programování je sourn pravidel a návodů, jenž zaručují že známe přesně v každém okamžiku stav projektu a při úplném poctivém užití minimalizují množství chyb v projektu.

  Nejdříve něco „teorie“.

  Pravidla jenž se používají:

  Do the Simplest Thing That Will Work

  Toto pravidlo zajišťuje že kód bude co nejjednodušší.**FIXME:**

35.1. Testování

První věcí o které bych rád pohovořil je testování jako princip. Proč testujeme? Testujeme proto abychom si ověřili podmínky za kterých program běží. Existuje vícero druhů testování.

* *para condition="author"*

  Kam umístnit testy? Jedno z otázek je kam unit testy umístnit. Je možno je psát do souborů kde jsou jednotlivé moduly i třídy definovány a spouštět je přes konstrukci

```
if $0 == __FILE__ then
    # run tests
end
```

  Tento způsob je ovšem proti nekterým pravidlům XP. Například nám nezaručuje že nedojde v průběhu vývoje a ladění ke změnám v kódu testů, ať už úmyslným či nikoli. Druhý způsob je psát testy do vlastních souborů. Tento nám dovoluje nastavit testům po „odladění“ příznak `ReadOnly`, spočítat si k nim kontrolní součty, archivivat je či různě zkombinovat uvedené možnosti.

35.1.1. Assertion testing

Jeden z nejjednodušších druhů testování. Testujeme zdali jsou splněny invariantní podmínky v průběhu vykonávání programů. Nejčastěji používáme pro testování vstupních hodnot metod.

35.1.2. Design by contract

FIXME:

35.1.3. Unit testing

FIXME:

35.2. RubyUnit

`rubyunit` je jedním z modulů realizujících unit testy.

Příklad 35-1. Příklad použití `rubyunit`

```
require 'rubyunit'

class TestThing < RUNIT::TestCase
    def testOne
        ...
        assert_equal(0, v)
    end
    def testTwo
        ...
    end
end

if $0 == __FILE__
    require 'runit/cui/testrunner'
    #RUNIT::CUI::TestRunner.quit_mode = false
    RUNIT::CUI::TestRunner.run(TestThing.suite)
end
```

V testu musíme zažádat o soubory testovacího modulu

```
require 'runit/testcase'
require 'runit/cui/testrunner'
require 'runit/testsuite'
require 'myclass'
```

Poté definujeme vlstní třídy testů

```
class Testing_class < RUNIT::TestCase
    ...
end
```

Názvy metod ve třídě tesů musí začínat `test`. Každá metoda může provést jeden či více testů. Pro testování používáme `assert` metody `RUNIT::TestCase`

* *Podrobně projít mtody, opravit, dopsat, a vyhledat rozdíly ve verzích.*

Metody `RUNIT::TestCase`

- `assert_fail`(message)
- `assert`(boolean, message)
- `assert_equal`(message)
- `assert_equal_float`(message)
- `assert_same`(message)
- `assert_nil`(message)
- `assert_not_nil`(message)
- `assert_respond_to`(message)

- `assert_kind_of`(message)
- `assert_instance_of`(message)
- `assert_match`(message)
- `assert_matches`(message)
- `assert_not_match`(message)
- `assert_exception`(message)
- `assert_no_exception`(message)
- `assert_operator`(message)
- `asserts`(message)
- `assert_send`(message)

Ve třídě testů můžeme definovat dvě speciální metody `setup` a `teardown`. První se spustí před každým testem a připraví prostředí, druhá po každém testu a provede nezbytný úklid.

* *Ověřit správnost tvrzení.*

Šablona pro testování

```
if __FILE__ == $0
    require 'runit/testcase'
    require 'runit/cui/testrunner'
    require 'runit/testsuite'

    class Testing_class < RUNIT::TestCase
        def test_feature1
            mine = My_class.new
            # ... etc
        end
        # ...
    end

    RUNIT::CUI::TestRunner.run(Testing_class.suite)
end
```

S RubyUnit se dodává skript `c2t.rb`. Tento skript se spouští s názvem souboru a třídy a vytvoří šablonu testovacího kódu.

35.2.1. Metody

assert(condition)

FIXME: doplnit

assert_equal(value1, value2)

FIXME: doplnit

35.3. Přechod z RubyUnit na Test::Unit

RubyUnit již není udržována a protože Test::Unit je s ní kompatibilní, je vhodné používat již jen Test::Unit. Staré testy převedeme pod nový testovací modul takto:

1. Zaměníme řádek

```
require 'runit'
```

za

```
require 'test/unit'
```

2. Zaměníme

```
class TestXYZ < RUNIT::TestCase
```

za

```
class TestXYZ < Test::Unit::TestCase
```

3. Zaměníme testovací metody.

původní	**nová**
`assert_equals`	`assert_equal`
`assert_exception`	`assert_raises`

4. Definujeme-li metody `setup`, `teardown`, ... Použijeme nové názvy

původní	**nová**
`setup`	`set_up`
`teardown`	`tear_down`

35.4. TestUnit

* *id="testunit" xreflabel="TestUnit" condition="author"*

Odkazy, zdroje:

- http://www.rubygarden.org/ruby?UsingTestUnit
- http://testunit.talbott.ws/
- http://www.b13media.com/dev/ruby/mock.html

Ukázka testu

```
require 'test/unit'                                          ❶
class TC_StringWrapper < Test::Unit::TestCase                ❷
    def test_wrap                                            ❸
        wrapper = StringWrapper.new
        assert_equal("This is a\nwrapped\nline.",            ❹
                     wrapper.wrap("This is a wrapped line.", 9),
                     "The line should have been wrapped to 9 columns")
    end
end
```

❶ Potřebujeme `'test/unit'`.

❷ Každá třída testů musí být podtřídou (dědicem) třídy `Test::Unit::TestCase`

❸ Třída testů obsahuje jednotlivé testy jako metody. Jména testů musejí začínat na `test`

❹ Pomocí metod třídy testů srovnáváme očekávané a skutečné výsledky.

Test spustíme

```
$ ruby tc_string_wrapper.rb
```

35.4.1. Instalace

Instalace není složitá. Nejdříve jsem si nahrál balíček `testunit-0.1.6.tar.gz`

```
$ cd $HOME/arch/lang/ruby/testunit
$ wget wget http://www.talbott.ws/testunit/packages/testunit-0.1.6.tar.gz
```

rozbalil jej

```
$ cd $HOME/source
$ tar xzf $HOME/arch/lang/ruby/testunit/testunit-0.1.6.tar.gz
```

Instalace byla jednoduchá. Balíček `testunit-0.1.5.tar.gz` jsem rozbalil, přepnul se do vytvořeného adresáře, nakonfiguroval a nainstaloval do poslední verze `ruby` kompilované z cvs:

```
$ cd $HOME/source
$ tar xzf $HOME/arch/lang/ruby/testunit/testunit-0.1.5.tar.gz
$ cd testunit-0.1.5
$ export ROOT=$HOME/opt/ruby-1.8.0-2003.01.07
$ $ROOT/bin/ruby setup.rb config  -- --bindir=$ROOT/bin \
                                   --rb-dir=$ROOT/lib/ruby \
                                   --so-dir=$ROOT/lib/ruby
$ ruby setup.rb setup
$ ruby setup.rb install
```

35.4.2. Spouštěče testů

Test Runners

K dispozici máme tyto spouštěče testů

- `Test::Unit::UI::Console::TestRuner`
- `Test::Unit::UI::GTK::TestRuner`
- `Test::Unit::UI::Fox::TestRuner`

35.4.3. Spouštění všech testů

Následující skript vyhledá všechny testy v aktuálním adresáři a spustí je.

Příklad 35-2. Spuštění všech testů v adresáři

```
#!/usr/bin/env ruby
# $Id: test_all.rb,v 1.1 2004/01/13 13:08:12 radek Exp $
# $Source: /home/radek/cvs/ruby-book/example/xp/test_all.rb,v $
#
# From: Simon Strandgaard

require 'test/unit'

class TestAll
    def TestAll.suite
 suite = Test::Unit::TestSuite.new
 Object.constants.sort.each do |k|
     next if /^Test/ !~ k
     constant = Object.const_get(k)
     if constant.kind_of?(Class) &&
      constant.superclass == Test::Unit::TestCase
  suite << constant.suite
     end
 end
 suite
    end
end

if __FILE__ == $0
    Dir.glob("test_*.rb").each do |file|
 require "#{file}"
    end
    require 'test/unit/ui/console/testrunner'
    Test::Unit::UI::Console::TestRunner.run(TestAll)
end
```

35.4.4. Popis modulů, třída a metod

Vše se nachází v modulu `Test/Unit`.

třída `AssertionFailedError`
modul `Assertions`
třída `Error`
třída `Failure`
třída `TestCase`
třída `TestResult`
třída `TestSuite`
modul `UI`

Ve třídě `Assertions` se nacházejí testovací metody.

- `assert_block(message="")` — Testování/předpoklad ne kterém jsou založeny všechny ostatní. Projde jestliž blok *yields* true.
- `assert(boolean, message="")` — projde, je li hodnota `boolean` pravdivá
- `assert_equal(expected, actual, message=nil)` — projde jestliže `expected == actual`

- `assert_raises(expected_exception_klass, message="")` — projde jsetliže blok vyvolá výjimku.
- `assert_instance_of(klass, object, message="")` — projde jestliže `object.class == klass`
- `assert_nil(object, message="")` — projde jestliže `object.nil?`
- `assert_kind_of(klass, object, message="")` — projde jestliže `object.kind_of?(klass)`
- `assert_respond_to(object, method, message="")` — projde když objekt implementuje metodu `method` — `object.respond_to?(method)`
- `assert_match(regexp, string, message="")` — projde když `string =~ regularExpression`.
- `assert_same(expected, actual, message="")` — projde když `actual.equal?(expected)` t.j. jedná se o stejnou instanci.
- `assert_operator(object1, operator, object2, message="")` — porovnává dva objekty na uvedený operátor. projde když `object1.send(operator, object2)` je true.
- `assert_nothing_raised(*args)` — projde když blok nevyvolá výjimku.
- `flunk(message="")` — neprojde nikdy, vždy selže.
- `assert_not_same(expected, actual, message="")` — projde když `!actual.equal?(expected)`.
- `assert_not_equal(expected, actual, message="")` — projde když `expected != actual`.
- `assert_not_nil(object, message="")` — projde když `!object.nil?`.
- `assert_does_not_match(regexp, string, message="")` — projde když `string !~ reguralExpression`.
- `assert_throws(expected_symbol, message="", &proc)` — projde když blok vyvolá (hodí) symbol.
- `assert_nothing_thrown(message="", &proc)` — projde když blok nevyvolá (nehodí) symbol.
- `assert_in_delta(expected_float, actual_float, delta, message="")` — projde když se očekávané číslo a aktuální číslo liší o méně než delta.
- `assert_send(send_array, message="")` — projde když ... FIXME:.

35.5. ZenTest

* *id="zentest" condition="author"*

Zdroje a odkazy:

- http://sourceforge.net/projects/zentest/
- http://freshmeat.net/projects/zentest/

FIXME:dopsat

35.6. Cucumber

*

Odkazy:

- David Chelimsky: The RSpec Book, Behaviour Driven Development with RSpec, Cucumber, and Friends
- BenMabey.com (http://BenMabey.com)
- GitHub (http://github.com/bmabey)
- Twitter: bmabey
- IRC: mabes

Testovací nástroj v kterém píšeme testovací scénáře.

```
project_root/
|
‘-- features
    |-- awesomeness.feature
    |-- greatest_ever.feature
    ‘-- support
        |-- env.rb
        ‘-- other_helpers.rb
    |-- step_definitions
    |   |-- domain_concept_A.rb
    |   ‘-- domain_concept_B.rb
```

STEP: Given a widget

Definition:

```
Given /^a widget$/ do
    # codes go here
end
```

Příklad 35-3. `features/manage_my_wishes.feature`

```
Feature: manage my wishes

  In order to get more stuff
  As a greedy person
  I want to manage my wish list for my family memebers to view

  Scenario: add wish
    Given I am logged in
    When I make a "New car" wish
    Then "New car" should appear on my wish list
```

```
$ cucumber features/manage_my_wishes.feature:7
```

7 je řádek scénáře

cucumber some.feature --language en-lol

Příklad 35-4. `features/setp_definitions/user_steps.rb`

```
Given /^I am logged in$/ do
    @current_user = create_user(:email_confirmed => true)

    visit new_session_path
    fill_in "Email", :with => @current_user.email
    fill_in "Password", :with => valid_user_attributes["password"]
    click_button
    # make sure we have actually logged in- so we fail fast if not
    #-- session[:user_id].should == @current_user.id
    #-- controller.current_user.should == @current_user
end
```

Fixture Replacement, Fixjour, Factory Girl, etc

Příklad 35-5. `spec/fixjour_builders.rb`

```
Fixjour do
    define_builder(User) do |klass, overrides|
        klass.new(
                  :email => "user#{counter(:user)}@email.com",
                  :password => 'password',
                  :password_confirmation => 'password'
        )
    end
end
```

```
$ gem install thoughtbot-clearance
$ ./script generate clearance
$ ./script generate clearance_features
```

Tables

Step tables

```
Scenario: view members list
  Given the following wishes exist
    | Wish          | Family Memeber |
    | Laptop        | Thomas         |
    | Nintendo Wii  | Candace        |
    | CHEEZBURGER   | FuzzBuzz       |

  When I view the wish list for "Candace"

  Then I Should see the following wishes
    | Wish          |
    | Nintendo Wii  |
```

35.7. Vanity

* ***Attributy:*** *id="Vanity"*

Odkazy:

- Vanity (http://vanity/labnotes.org/) Experiment Driven Development
-

35.8. ABingo

*

Odkazy:

- ABingo (http://www.fingocardcreator.com/abingo/installation) Rails A/B Testing
- github ryanb / abingo (http://github.com/ryanb/abingo)
- Railscast 214 ()
-

IV. Knihovny

FIXME: povídání o knihovnách.

Do této části řadím jak knihovny tříd, tak také různé aplikační prostředí pro realizaci tak rozsáhlých témat jako jsou grafické uživatelské rozhraní nebo prostředí pro vývoj a běh webových aplikací.

Kapitola 36. Programy

Programy, nástroje

* *chapter id="programy" xreflabel="Programy"*
* *Přemístnit relevantní texty do jiných kapitol a sekcí. Zrušit tuto kapitolu.*

36.1. DbTalk

Talk to your database servers

- DbTalk main site (http://www.epot.cz/dbtalk/)
- Autor

Kapitola 37. Šifrování a hesla

Odkazy:

- HowtoAuthenticate (http://wiki.rubyonrails.org/rails/pages/HowtoAuthenticate)
- Apache HTTP Server Version 2.2 Password Formats (http://httpd.apache.org/docs/2.2/misc/password_encryptions.html)
-

```
require 'digest/sha1'
require 'base64'
'{SHA}' + Base64.encode64(Digest::SHA1.digest(password))
```

Crypt

```
$ openssl passwd -crypt myPassword
$ openssl passwd -crypt -salt rq myPassword
$ htpasswd -nbd myName myPassword
```

MD5

```
$ openssl passwd -apr1 -salet r31..... myPassword
$ openssl passwd -apr1 myPassword
$ htpasswd -nbm myName myPassword
```

SHA1

```
$ htpasswd -nbs myName myPassword
```

* ***To be done.***

37.1. HTAuth (apr1)

* ***Attributy:*** *id="htauth"*

Odkazy:

- HTAuth (http://copiousfreetime.rubyforge.org/htauth/)

Knihovna HTAuth realizuje v Ruby všechny operace s hesly a jejich hashi jako původní programy od Apache **htdigest** a **htpasswd**.

Knihovna je dostupná jako gem balíček a její instalace je tedy velmi snadná.

```
# gem install htauth
Successfully installed htauth-1.0.3
```

Příklad 37-1. Ukázka získání apr1 hashe hesla, `apr1.rb`

```
#!/usr/bin/env ruby

require 'rubygems'
require 'htauth'
heslo = ARGV[0]
apr1=HTAuth::Md5.new.encode(heslo)
```

```
puts "heslo=#{heslo}"
puts "apr1=#{apr1}"
$ ./apr1.rb heslo
heslo=heslo
apr1=$apr1$t69wKyx3$Ci1IiXyBjnMrK1Ibc1G5C1
```

Příklad 37-2. Získání apr1 hashe hesla

```
# File: session/apr1.ses
require 'rubygems'
true
require 'htauth'
true
apr1 = HTAuth::Md5.new.encode('heslo')
"$apr1$Gv1AJL8q$0HU08QWAjdaq/sXjARpKn/"
```

Příklad 37-3. Kontrola hesla

```
# File: session/htauth.authenticate.ses
require 'rubygems'
true
require 'htauth'
true

line = 'radek:$apr1$SsPBHx9k$Yk5CifOFhRqKBRySquO3P1'
"radek:$apr1$SsPBHx9k$Yk5CifOFhRqKBRySquO3P1"
entry = HTAuth::PasswdEntry.from_line(line)
#<HTAuth::PasswdEntry:0x7f2abffb9e60 @algorithm=#<HTAuth::Md5:0x7f2abffb96b8 @salt="SsPBHx9k">, @user="r
entry.authenticated?('heslo')
true
```

Kapitola 38. Databáze

* *chapter id="database" xreflabel="Databáze"*

Tato kapitola je o použití databází v Ruby. Jedná se o databáze od jedoduchých, přes SQL až po specializované jako je například LDAP.

- Ruby/DBI (http://ruby-dbi.sf.net/)
- Ruby/DBI (http://ruby-dbi.sourceforge.net/)

* *Následující příklad pochází z Ruby-Talk 56115 od Teda*

Než se o čemkoliv zmíním, krátky příklad předem.

```
#!/usr/bin/env ruby
require 'dbi'
begin
    DBI.connect('DBI:pg:DeMolay', 'user', 'password') do |dbh|
        ARGV.each do |file|
            query = '/* '+$0+':'+__LINE__.to_s+'('+file+
                    '['+File.size(file).to_s+"]) */\n"    ❶
            query = "/*#$0:#{__LINE__}(#{file}[#{File.size(file)}])*/\n"
            File.open(file)  {|f| query << f.readlines.to_s }
            puts query
            dbh.select_all(query) do |row|
                puts row.join("\t")
                puts
            end
        end
    end
rescue => e
    puts e.to_s
    puts e.backtrace
end
```

❶ Připojení k Postsovské (`:pg:`) databázi `DeMolay`

38.1. Ruby/DBI

Ruby/DBI je modul realizující rozhraní DBI do několika databázových strojů. Výhodou je jednotné [1] API. Program je tedy psaný obecně a o připojené databázi se rozhoduje až konfigurací.

Seznam ovladačů v DBI

- `ADO` - ActiveX Data Objects
- `DB2` - DB2
- `InterBase` - InterBase
- `mSQL` - mSQL
- `Mysql` - MySQL
- `ODBC` - ODBC

- `Oracle` - Oracle 7, Oracle 8/8i
- `Pg` - PostgreSQL
- `Proxy` - Proxy/Server
- `SQLite` - SQLite
- `SQLRelay` -
- `Sybase` - Sybase

38.1.1. API

Poznámky k API DBI

38.1.1.1. Výjimky (Exceptions)

Modul DBI může vyvolat následující výjimky.

Waring < RuntimeError

FIXME:

`Error` < RuntimeError

FIXME:

`InterfaceError < Error`

FIXME:

`NotImplementedError < InterfaceError`

FIXME:

`DatabaseError < Error`

FIXME:

`DataError < DatabaseError`

FIXME:

`OperationalError < DatabaseError`

FIXME:

`IntegrityError < DatabaseError`

FIXME:

`InternalError < DatabaseError`

FIXME:

`ProgrammingError < DatabaseError`

FIXME:

`NotSupportedError < DatabaseError`

FIXME:

38.1.1.2. Funkce modulu `dbi`

DBI.connect

Jméno

`DBI.connect` — připojení k databázovému stroji daným ovladačem

Prototyp

```
DBI.connect(driver_url, user=nil, auth=nil, params=nil);
```

Popis

Připojíme se přes zadaný ovladač, k databázovému stroji. Ovladač je určen parametrem *`driver_url`* a má tvar

```
dbi:ovladač:databáze
```

kde *`ovladač`* je jeden z

Pg
MySQL
...

příklady

```
"dbi:Oracle:oracle.neumanm"
"dbi:Pg:dbname=testdb;host=sql"
```

Metoda vrací objekt typu `DBI::DatabaseHandle`

38.1.1.3. Metody

connect

Jméno

`connect` — připojení k databázi

Popis

Připojíme se přes daný ovladač k databázi.

Použití ovladače PostgreSQL

FIXME:

dbname nebo database

: název databáze

host

: počítač na kterém databázový stroj běží

port

: port na kterém databázový stroj čeká na spojení

options

: FIXME:

tty

: FIXME:

Příklad 38-1. Připojení k databázi PostgreSQL

```
require 'dbi'

db = DBI.connect("DBI:Pg:jim", user, password)  ❶
# Use the database
db.disconnect   # When done
```

❶ Připojení k datbázi „jim“ s použitím ovladače databáze PostgreSQL.

38.1.2. Kompilace

FIXME:

Před vlastním překladem Ruby/DBI si přeložím knihovnu sqlite

```
$ # Překlad sqlite z cvs.  Version=2.7.5, date=2003-01-09
$ cd $HOME/source
$ mkdir sqlite-2.7.5-2003.01.09
$ cd sqlite-2.7.5-2003.01.09
$ cp -a $HOME/mirror/cvs/sqlite/* .
$ find . -name CVS -exec rm -r {} \;
$ cd $HOME/tmp
$ mkdir sqlite-bld
```

```
$ cd sqlite-bld
$ mkdir $HOME/opt/sqlite-2.7.5-2003.01.09
$ $HOME/source/sqlite-2.7.5-2003.01.09/configure --prefix=$HOME/opt/sqlite-2.7.5-2003.01.09
$ make              0:13:39
```

Pro kompilaci Rubi/DBI jsem si připravil skript. Tento kompiluje podel zadání ze stažených zdrojů, nebo ze staženého cvs stromu.

Příklad 38-2. Kompilace Rubi/DBI

```
#!/bin/sh
# $Id: compile-ruby-dbi,v 1.2 2003/01/05 19:30:00 radek Exp $
# Kompilace a instalace Rubi/DBI
# Copyright (C) 2002 Radek Hnilica
# All rights reserved.

BINDIR=$HOME/bin
RUBYDIR=$HOME/lib/ruby
TMPDIR=$HOME/tmp
CVSDIR=$HOME/mirror/ruby/ruby-dbi
DOWNLOAD_DIR=$HOME/download/ruby/ruby-dbi
COMPILE_DIR=$TMPDIR/ruby-dbi

PKGS=dbi,dbd_sqlite,dbd_pg,dbd_mysql


# Print script usage
function usage() {
    cat - <<EOF
usage: $0 [version|cvs]
EOF
    exit 0
} # usage



# Unpack source tarball of given version to $COMPILE_DIR/src
# directory
function unpack_source() {
    VERSION=$1
    FILE=$DOWNLOAD_DIR/ruby-dbi-all-${VERSION}.tar.gz

    if [ -f $FILE ]; then
 cd $COMPILE_DIR
 tar xzvf $DOWNLOAD_DIR/ruby-dbi-all-${VERSION}.tar.gz
 mv ruby-dbi-all src
    else
 echo "Source tarball ${FILE} doesn’t exist"
 exit 0
    fi
} #unpack_souce
```

```
# Copy source from local CVS mirror to $COMPILE_DIR/src
# directory.
function copy_cvs() {
    cd $COMPILE_DIR
    cp -a $CVSDIR/src .
    find . -name 'CVS' -exec rm -r {} \;
} #copy_cvs



### MAIN
# FIXME: test number of arguments if wrong then do usage
VERSION=$1

# Create temporary directory for compiling
cd $TMPDIR
# FIXME: if [ -d ruby-dbi ]; then rm -r ruby-dbi; fi
rm -r ruby-dbi
mkdir ruby-dbi
cd ruby-dbi

case $VERSION in
    cvs)    copy_cvs;;
    [0-9]*) unpack_source $VERSION;;
    *)      usage
esac

# Compile and install
cd $COMPILE_DIR/src
ruby setup.rb config --with=$PKGS --without=dbd_sybase \
    --bin-dir=$BINDIR --rb-dir=$RUBYDIR --so-dir=$RUBYDIR
ruby setup.rb setup
ruby setup.rb install

# Cleanup
rm -r $COMPILE_DIR
```

38.1.3. Příklady

Příklad 38-3. `list-records.rb`

```
#!/usr/bin/env ruby
# $Id: list_records.rb,v 1.3 2002/11/06 05:47:51 radek Exp $
# $Source: /home/radek/cvs/ruby-book/example/dbi/list_records.rb,v $

require 'dbi'

DBI.connect('DBI:Pg:dbname=testdb;host=sql', 'radek') do |dbh|
    puts "selecting..."
    dbh.select_all('SELECT Id,Nazev FROM Firma') do |row|
```

```
        p row
    end
end
```

38.1.4. SQLite

Odkazy

- DAD-IT Ruby-SQLite (http://ruby-lua.unolotiene.com/ruby-sqlite.whtm) - Knihovna pro přímý přístup využívá API SQLite.
- DAD-IT Ruby-SQLite (http://domingo.dad-it.com/ruby-sqlite.whtm) - Knihovna pro přímý přístup využívá API SQLite.

Příklad 38-4. Příklad použití Ruby-SQLite

```
require 'sqlite'

def getDAD(str)
    str + ' sardina'
end

sq = SQLite.new('test.db')
print(getDAD('dddd'),"\n")
sq.setFunc('getDAD',1,self,'getDAD')
print(sq.exec("select getDAD('robertp') as duo;"),"\n")
# should print : roberto sardina
sq.close()
```

Příklad 38-5. Druhý příklad použití Ruby-SQLite

```
require 'sqlite'

def my_func(str)
    '[[' + str + ']]'
end

sq = SQLite.new('my_db_file.db')
sq.setFunc('wrap_str',1,self,'my_func')
print sq.exec("select wrap_str(name) from users;")
# should print : [[Paul]]
sq.close()
```

38.2. DBI tutoriál

Malý tutoriálek jak používat DBI.

38.2.1. Autorské poznámky k tutoriálu

* *section condition="author"*

Dle možností dalé nazanořovat sekce, nechat ploché.

Návrh sekcí

- Připojení k databázi, SQL serveru
- Dotazy **SELECT**
- Použití transakcí

38.2.2. Připojení k databázi

K databázi se připojujeme voláním metody connect modulu DBI

Příklad 38-6. Připojení k databázi s použitím DBI

```
require 'dbi'
DBI.connect('DBI:Pg:dbname=testdb;host=sql', 'user', 'password') do |dbh|
    ... dbh je databázový ovladač otevřeného připojení k databázi
end
```

38.2.3. Dotazy na tabulku

K jednotlivým polím výsldku můžeme přistupuvat buď přes index, první pole má index `0`, nebo přes jména sloupců které použijeme jako indexy.

```
val = row[2]
val = row['height']
```

Pokud neznáme jména sloupců a ani jejich počet, prostě provádíme dotaz na neznámou tabulku, můžeme použít iterační metodu `each_with_name`

```
dbh.execute("SELECT * FROM mytable") do |row|
    row.each_with_name do |val, name|
        #... pole name má hodnotu val
        printf "%s: $s, ", name, val.to_s
    end
    print "\n"
end
```

Pokud potřebujeme zjistit jména polí, například chceme vytvořit tabulku ať již v HTML nebo v prostém textu kde v hlavičce uvedeme jména sloupců a pod nimi hodnoty použijeme **FIXME:**

```
FIXME: vyřešit a dopsat příklad
sth = dbh.execute("SELECT f1, f2, f3 FROM mytable ORDER by f1;")
sth.column_info.each ...
sth.each do |row|
    puts "<tr>"
    row.each do |col|
```

```
        puts "<td>#{col}</td>"
    end
    row.collect{|fld| "<td>#{fld.to_s}</td>"}.join
    puts "</tr>\n"
end
sth.finish
```

38.2.4. Transkace

DBI nabízí možnost použití transakcí, nicméně nezaručuje. Je nutno vědět zdali použitý databázovy „backend“ transakce podporuje.

* ***FIXME:*** *Dopsat seznam databázových backendů podporujících transakce.*

```
dbh['AutoCommit'] = true
dbh['AutoCommit'] = false
```

Příklad použití ve kterém si řídíme transakce sami:

Příklad 38-7. Ruční řízení transakcí v DBI

```
dbh['AutoCommit'] = false
begin
    dbh.do("... SQL příkaz ...")
    dbh.do("... SQL příkaz ...")
    dbh.commit
rescue
    puts "Transakce neuspěla"
    dbh.rollback
end
```

Jiný přístup v řízení transakcí vychází z použití metody `transaction`

Příklad 38-8. Ruční řízení transakcí v DBI metodou `transaction`

```
dbh['AutoCommit'] = false
dbh.transaction do |dbh|
    dbh.do("... SQL příkaz ...")
    dbh.do("... SQL příkaz ...")
end
```

38.3. Mnemonic

An Object Prevalence Layer for the Ruby Language

Odkazy a literatura

- Mnemonic
- Mnemonic Home

Poskytuje „Transparent persistence, fault-tolerance and load-balancing of the execution of the business logic of an information system through the use of state snapshoots as well as command and query queuing or logging.“

This technikue is orders of magnitude *faster* and *simpler* than using an DBMS. Write *plain* ruby classes: no pre or post-processing required; no ineritance from base-class required.

38.4. ruby-ldap

* *section id="ruby-ldap" xreflabel="ruby-ldap"*

Ruby/LDAP je rozšiřující knihovna pro přístup k LDAPu. Používá API tak jak je popsáno v RFC1823. Autorem je Takaaki Tateishi. Knihovna se nachází na . (http://ruby-ldap.sourceforge.net)

Překlad ze zdrojů je velmi jednoduchý. Je třeba mít jen nainstlovány vývojářské verze některé z knihoven ldap. Při konfiguraci pak parametrem oznámíme kterou že to knihovnu máme nainstalovánu. Možné parametry jsou

`--with-openldap1`	OpenLDAP1
`--with-openldap2`	OpenLDAP2
`--with-netscape`	NetscapeSDK libraries
`--with-wldap32`	Windows2000 (or ADSI)

V mém případě to byla knihovna `OpenLDAP2`

```
$ ruby extconf.rb --with-openldap2
$ make
$ ruby install.rb
```

Dostupné konstanty, metody a třídy modulu Ruby/LDAP

`LDAP::LDAP_VERSION`

FIXME:

`LDAP::LDAP_MAX_VERSION`

FIXME:

`LDAP::VERSION`

FIXME:

`LDAP.err2string(errcode)`

FIXME:

```
LDAP.dn2ufn(dn)
LDAP.mod(mod_op, mod_type, mod_vals) (= LDAP::Mod.new)
LDAP.hash2mods(mod_op, hash)
LDAP.entry2hash(entry) (= entry.to_hash)
LDAP::Conn.new(host="localhost", port=LDAP::LDAP_PORT)
        : conn (raise LDAP::Error)
```

38.4.1. Přehled objektů a metod

38.4.1.1. třída Conn

```
LDAP::Conn.new(host = "localhost", port = LDAP::LDAP_PORT)
                : conn (raise LDAP::Error)
LDAP::Conn.open(host = "localhost", port = LDAP::LDAP_PORT)
                : conn (raise LDAP::Error)
LDAP::Conn#simple_bind(dn = nil, password = nil){ ... }
                : conn (raise LDAP::ResultError)
LDAP::Conn#bind(dn = nil, password = nil,
                 method = LDAP::LDAP_AUTH_SIMPLE){|conn| ... }
                 (raise LDAP::ResultError)
LDAP::Conn#bind(dn = nil, password = nil,
                 method = LDAP::LDAP_AUTH_SIMPLE) : conn
                 (raise LDAP::ResultError)
LDAP::Conn#unbind() (raise LDAP::ResultError)
LDAP::Conn#perror(str)
LDAP::Conn#result2error(ldap_msg) : errcode
LDAP::Conn#err2string(errcode) : errmsg
LDAP::Conn#get_errno : errcode [if available]
LDAP::Conn#search(basedn, scope, filter, attrs = nil, attrsonly = false,
                  sec = 0, usec = 0, s_attr = nil, s_proc = nil){|entry| ... }
                  : conn (raise LDAP::ResultError)
LDAP::Conn#search2(basedn, scope, filter, attrs = nil, attrsonly = false,
                   sec = 0, usec = 0, s_attr = nil, s_proc = nil){|entry_as_hash| ... }
                  : conn (if a block is given) / Array of Hash (if no block is given)
                  (raise LDAP::ResultError)
LDAP::Conn#search_ext(basedn, scope, filter, attrs = nil, attrsonly = false,
                      serverctrls, clientctrls, sec = 0, usec = 0,
                      s_attr = nil, s_proc = nil){|entry| ... }
                      : conn (raise LDAP::ResultError)
LDAP::Conn#search_ext2(basedn, scope, filter, attrs = nil, attrsonly = false,
                       serverctrls, clientctrls, sec = 0, usec = 0,
                       s_attr = nil, s_proc = nil){|entry_as_hash| ... }
                       : conn (if a block is given) / Array of Hash (if no block is given)
                       (raise LDAP::ResultError)
LDAP::Conn#add(dn, ldap_mods) : self (raise LDAP::ResultError)
LDAP::Conn#add_ext(dn, ldap_mods, serverctrls, clientctrls)
                    : self (raise LDAP::ResultError)
LDAP::Conn#modify(dn, ldap_mods) : self (raise LDAP::ResultError)
LDAP::Conn#modify_ext(dn, ldap_mods, serverctrls, clientctrls)
                      : self (raise LDAP::ResultError)
LDAP::Conn#modrdn(olddn, newdn, delete) : self (raise LDAP::ResultError)
LDAP::Conn#delete(dn) : self (raise LDAP::ResultError)
LDAP::Conn#delete(dn, serverctrls, clientctrls) : self (raise LDAP::ResultError)
LDAP::Conn#set_option(opt, data) : self (raise LDAP::ResultError)
LDAP::Conn#get_option(opt) : data (raise LDAP::ResultError)


modify(dn, ldap_mods) :self (raise LDAP::ResultError)
modify_ext(dn, ldap_mods, serverctrls, clientctrls) :self (raise
LDAP::ResultError)
```

Provádí změny v záznamu.

38.4.1.2. třída `Entry`

```
LDAP::Entry#get_dn : dn
LDAP::Entry#get_values : vals
LDAP::Entry#get_attributes : attrs
LDAP::Entry#dn (= get_dn)
LDAP::Entry#vals (= vals)
LDAP::Entry#attrs (= get_attributes)
LDAP::Entry#to_hash : Hash
```

38.4.1.3. třída `Mod`

Zapouzdřuje změny.

```
LDAP::Mod.new(mod_op, mod_type, mod_vals) : ldap_mod
LDAP::Mod#mod_op : mod_op
LDAP::Mod#mod_type : mod_type
LDAP::Mod#mod_vals : mod_vals
LDAP::Mod#mod_op=(mod_op)
LDAP::Mod#mod_type=(mod_type)
LDAP::Mod#mod_vals=(mod_vals)
```

`new(mod_op, mod_type, mod_vals)`

FIXME:

`mod_op`

FIXME:

`mod_type`

FIXME:

`mod_vals`

FIXME:

38.5. MySQL

Odkazy:

- Using the Ruby MySQL Module (http://www.kitebird.com/articles/ruby-mysql.html)

Přímé připojení k databázi MySQL

```
require 'mysql'
db = Mysql::new('server.example.com', 'USER', 'PASSWORD', 'DATABASE')
people = db.query('SELECT * FROM person')
people.each_hash do |person|
    pin = person['pin']
    ⋮
end
```

38.6. SQLite

*

Odkazy:

- YouTube SQLite Tutorial (http://www.youtube.com/watch?v=NYlCVoj4peg) 4:19 [2009-07-11]
- YouTube SQLite Programming Using Ruby (http://www.youtube.com/watch?v=Tx4QWdJD2yU) 10:11 [2009-12-18]

Instalace na MS Windows provádět podle Installing SQLite 3 on Windows for use in Ruby on Rails (http://www.bestechvideos.com/2007/02/08/installing-sqlite-3-on-windows-for-use-in-ruby-on-rails)

Poznámky

V rámci možností. Drobné odchylky se mohou vyskytnout a je třeba prostudovat podrobně dokumentaci.

Kapitola 39. Síťování

Síťové aplikace a jejich programování

* *chapter id="networking" xreflabel="Síťování"*
* ***FIXME:*** *Napsat pár obecných slov o síťování jako takovém. V kapitole pak budou zmíněny všechny významější balíky a knihovny, ktreré se tématu síťování týkají.*
* *condition="author"*

Kapitola pojednává o síťové práci s ruby.

- Část věnovaná sítím TCP/IP. Síťování na nejnižší úrovni.
- Webové servery. Integrace s Apeche, webové servery psané v Ruby, speciální webové servery..

Odkazy:

- Ruby Programming Language Enables Concise Network Programming (http://www.devx.com/enterprise/Article/28101) — simple web server
-
-

39.1. Různé

39.1.1. SSL

FIXME:

V Debian Sarge jsou k dispozici balíčky `libopenssl-ruby`, `libopenssl-ruby1.6` a `libopenssl-ruby1.8`.

```
# aptitude install libopenssl-ruby
```

39.2. Sokety *Sockets*

Sokety jsou síťovým prostředím na nejnižší úrovni. Pro programátora jsou síťovou obdobou souborů. Knihovna se sekety se jmenuje `socket`

```
require 'socket'
```

Slovníček pojmů

domain

Rodina protokolů která bude použita jako přenosový mechanismus. Může nabývat knostant `PF_INET`, `PF_UNIX`, `PF_X35`, ...

type

Typ (způsob) komunikace mezi oběma koncovými body, typicky `SOCK_STREAM`. `SOCK_DGRAM` pro datagramy.

protocol

Obvykle `0`, může být použit k identifikaci varianty protokolu v doméně protokolů.

hostName

Identifikace (adresa) počítače. Může být:

- řetězec se jménem počítače (stroj.firma.cz), ip adresa (123.456.23.67), nebo IPV6 adresa.
- řetězec "*`broadcast`*", který určuje `INADDR_BROADCAST` adresu
- prázdný řetězec který určuje `INADDR_ANY`
- číslo, interpretované jako binární adresa počítače.

port

někdy taky nazývaný *service*. Každý počítač poslouchá volání klientů na jednom či více portech. Port je celé číslo, řetězec obsahující číslo, nebo jméno služby (service).

Sokety jsou dědici třídy `IO`.

39.2.1. class BasicSocket

do_not_reverse_lookup, do_not_reverse_lookup=, lookup_order, lookup_order=, close_read, close_write, getpeername, getsockname, getsockopt, recv, send, setsockopt, shutdown

39.2.1.1. class IPsocket

getaddress, addr, peeraddr

39.2.1.2. class TCPSocket

gethostbyname, new, open, recvfrom

39.2.1.3. class TCPServer

TCPServer přijímá příchozí TCP spojení. new, open, accept Jednoduchý WWW server

```
require 'socket'
port = (ARGV[0] || 80).to_i
server = TCPServer.new('localhost', port)
while (session = server.accept)
    puts "Request: #{session.gets}"
    session.print "HTTP/1.1 200/OK\r\nContent-type: text/html\r\n\r\n"
    session.print "<html><body><h1>#{Time.now}</h1></body></html>\r\n"
end
```

39.2.1.4. class UNIXSocket

new
open
addr
path
peeraddr
recvfrom

Třída `UNIXSocket` podporuje meziprocesovou komunikaci na jednom počítači s použitím doménového protokolu Unix. Ačkoliv použitý protokol podporuje jak datagramy, tak proudové spojení, Ruby knihovna nabízí jen proudové spojení.

```
require 'socket'


$path = "/tmp/sample"


sThread = Thread.start do        # run server in a thread
  sock = UNIXServer.open($path)
  s1 = sock.accept
  p s1.recvfrom(124)
end


client = UNIXSocket.open($path)
client.send("hello", 0)
client.close


sThread.join
```

39.2.1.5. class UNIXServer

new
open
accept

Třída `UNIXServer` nabízí jednoduchý soketový server.

39.3. TCP server

```
#!/usr/bin/env ruby
# $Id: echo_server1.rb,v 1.1 2002/06/05 15:48:59 radek Exp $

require 'socket'

server = TCPServer.new 'localhost', 12345
while client = server.accept
    Thread.new(client) {|c|
 until c.eof?
     s = c.gets
     c.puts s
```

```
 end
    }
end
```

39.3.1. Nezapracované podklady

39.3.1.1. Email „Re: TCP Server“ od Bulata Ziganshina (<bulatz@integ.ru>)

```
Hello Shannon,

Friday, December 13, 2002, 6:20:29 PM, you wrote:
SF> Can anyone show me how to do this by writing a echo server which can connect
SF> multiple clients? And is there events like "on_connect" "on_disconnect"
SF> available in ruby?

from ruby distribution :)

# socket example - server side using thread
# usage: ruby tsvr.rb

require "socket"

gs = TCPserver.open(0)
addr = gs.addr
addr.shift
printf("server is on %s\n", addr.join(":"))

while TRUE
    Thread.start(gs.accept) do |s|
        print(s, " is accepted\n")
        while s.gets
            s.write($_)
        end
        print(s, " is gone\n")
        s.close
    end
end

--
Best regards
 Bulat                          mailto:bulatz@integ.ru
```

39.4. NTP

Jednoduchý klient

```
require "net/telnet"
timeserver = "www.fakedomain.org"

tn = Net::Telnet.new("Host"       => timeserver,
```

```
                        "Port"       => "time",
                        "Timeout"    => 60,
                        "Telnetmode" => false)
    local = Time.now.strftime("%H:%M:%S")
    msg = tn.recv(4).unpack('N')[0]
    # Conver to epoch
    remote = Time.at(msg - 2208988800).strftime("%H:%M:%S")
    puts "Local : #{local}"
    puts "Remote: #{remote}"
```

39.5. Poštovní protokol POP3

* *section id="pop3"*

V Ruby snadno s pomocí knihovny POP3 realizujeme poštovního klienta jenž umí vybírat tímto ptotokolem poštu. Uvedu jen několik ukázek. První prochází poštu na serveru a vypisuje subjekty zpráv.

```
require "net/pop"
pop = Net:POP3.new("pop.fakedomain.org")
pop.start("gandalf", "mellon")      # user, password
pop.mails.each do |msg|
    puts msg.header.grep /^Subject: /
end
```

Druhá ukázka je program/skript jenž maže ze serveru zprávy jenž obsahují řetězec `make money fast`. Tento se může vykytovat kdekoliv ve zprávě, jak v těle tak v hlavičkách.

```
require "net/pop"
pop = Net:POP3.new("pop.fakedomain.org")
pop.start("gandalf", "mellon")      # user, password
pop.mails.each do |msg|
    if msg.all =~ /make money fast/i
        msg.delete
    end
end
pop.finish
```

39.6. SMTP

```
require 'net/smtp'

msg = <<EOF
Subject: Many things
Text
EOF

Net:SMTP.start("smtp-server.fakedomain.com") do |smtp|
    smtp.sendmail(msg, "warlus@fake1.com", 'alice@fake2.org')
end
```

39.7. WWW, http — klient

Příklad 39-1. Jednoduchý klient

```
#!/usr/bin/env ruby
# $Id: http_client1.rb,v 1.1 2002/05/30 13:09:46 radek Exp $

require 'net/http'
Net::HTTP.start('www.linux.cz', 80) {|http|
  response, = http.get('/index.html')
  puts response.body
}
```

Příklad 39-2. Ještě jednodušší verze

```
#!/usr/bin/env ruby
# $Id: http_client2.rb,v 1.1 2002/05/30 13:09:46 radek Exp $

require 'net/http'
Net::HTTP.get_print 'www.linux.cz', '/index.html'
```

Další povídání o programování Webových aplikací je v části Programování webových aplikací

39.8. dRuby — Distributed Ruby

* *section id="druby" xreflabel="dRuby"*

- dRuby page (http://www2a.biglobe.ne.jp/~seki/ruby/druby.en.html)
- Zdrojové kódy verzí 1.3.9 (http://www2a.biglobe.ne.jp/~seki/ruby/drb-1.3.9.tar.gz), 2.0.3 (http://www2a.biglobe.ne.jp/~seki/ruby/drb-2.0.3.tar.gz), 2.0.4 (http://www2a.biglobe.ne.jp/~seki/ruby/drb-2.0.4.tar.gz)
- Intro to DRb by Chad Fowler (http://www.chadfowler.com/ruby/drb.html)
- Distributed Ruby (http://www.rubycentral.com/articles/drb.html)
- dRuby reference manual (http://www.rubyist.net/~rubikitch/RDP-en.cgi?cmd=view;name=dRuby)
- dRuby white paper (http://rwiki.jin.gr.jp/cgi-bin/rw-cgi.rb?cmd=view;name=dRuby+white+paper)
- Kniha: Distributed Programming with Ruby

dRuby, nebo taky DRb je označení pro *Distributed Ruby*. Jedná se o knihovnu která umožňuje zasílat a přijímat zprávy od vzdálených Ruby objektů přes TCP/IP.

Aplikace používající knihovnu DRb sestává ze dvou částí, serveru a klienta.

Jednoduchý server vypadá takto:

```
#!/usr/bin/env ruby
# File: example/net/drb/server.rb
require 'drb'

class TestServer
```

```
        def doit
            "Hello, Distributed World"
        end
    end

    aServerObject = TestServer.new
    DRb.start_service('druby://localhost:9000', aServerObject)
    DRb.thread.join # Don't exit just yet!
```

A klient k tomuto serveru pak

```
    #!/usr/bin/env ruby
    # File: example/net/drb/client.rb
    require 'drb'

    DRb.start_service()
    obj = DRbObject.new(nil, 'druby://localhost:9000')

    # Now use obj
    p obj.doit
```

Jiný příklad. Server:

```
    #!/usr/bin/ruby
    # File: example/net/drb/srv2.rb
    require 'drb/drb'

    class DRbEx
        def initialize
            @hello = 'hello'
        end

        def hello
            @hello
        end

        def sample(a, b, c)
            a-to_i + b.to_i + c.to_i
        end
    end

    DRb.start_service(nil, DRbEx.new)
    puts DRb.uri
    DRb.thread.join
```

a klient:

```
    #!/usr/bin/ruby
    # File: example/net/drb/cli2.rb
    require 'drb/drb'

    class DRbEx2
        include DRbUndumped

        def initialize(n)
            @n = n
        end
```

```
    def to_i
        @n.to_i
    end
end

there = ARGV.shift
DRb.start_service()
ro = DRbObject.new(nil, there)
p ro.hello
p ro.sample(DRbEx2.new(1), 2, 3)
ro.sample(1, ro.sample(DRbEx2.new(1), 2, 3), DRbEx2.new(3))
```

39.8.1. Zabezpečení

Distribuované objekty můžeme chránit před neoprávněným přístupem nastavením ACL

```
if __FILE__ == $0
    acl = ACL.new(%w(deny all
                     allow 192.168.1.*
                     allow 209.61.159.*
                     allow localhost))
    DRb.install_acl(acl)    # must be called before service starting

    DRb.start_service(nil, SongNameServer.new("/tmp/songname")
    puts DRb.uri
    DRb.thread.join
end
```

Zabezpečení před vykonáním nežádoucího kódu.

```
$SAFE = 1
```

Přístupová práva (ACL)

```
require 'drb'
require 'drb/acl'
$SAFE = 1

class HelloWorldServer
    def say_hello
        "Hello, world!"
    end
end

acl = ACL.new(%w{deny all allow 192.168.1.12 allow 192.168.1.7})
DRb.install_acl(acl)
DRb.start_service("druby://192.168.1.12:61676", HelloWorldServer.new)
DRb.thread.join
```

39.8.2. Volání

```
DRb.start_service(uri, object)
```

DRb.start_service("druby://:7777", SongNameServer.new("/tmp/songname"))

39.8.3. Distributed Ruby

Server:

```
#!/usr/bin/env ruby
# File: example/net/drb/server.rb
require 'drb'

class TestServer
    def doit
        "Hello, Distributed World"
    end
end

aServerObject = TestServer.new
DRb.start_service('druby://localhost:9000', aServerObject)
DRb.thread.join # Don't exit just yet!
```

Klient:

```
#!/usr/bin/env ruby
# File: example/net/drb/client.rb
require 'drb'

DRb.start_service()
obj = DRbObject.new(nil, 'druby://localhost:9000')

# Now use obj
p obj.doit
```

39.9. rinda

* *FIXME: dopsat, rozmyslet případné začlenění do části dRuby.*

? Jmenný server pro dRuby

39.10. Wiki

Zdroje a odkazy:

- http://sourceforge.net/projects/aswiki/
- http://sourceforge.net/projects/rdoc-wiki/
- http://rwiki.jin.gr.jp/cgi-bin/rw-cgi.rb
- http://tiki.is.os-omicron.org/tiki.cgi?c=v&p=Tiki

V Ruby je napsáno několik Wiki serverů.

39.10.1. AsWiki

Používá RCS

FIXME:

39.10.2. RDocWiki

* *Je možno získat z Source Forge (http://sourceforge.net/projects/rdoc-wiki). K dnešnímu dni (2002-12-09) je ve stavu 2 - Pre-Alpha.*

> **Varování**
>
> Je k dispozici na Source Forge (http://sourceforge.net/projects/rdoc-wiki) jen v cvs.

A Wiki clone using RDoc's (Ruby documentation format) markup language. It features pluggable storage backends (databases, file-system), pluggable versioning backend (diff, rcs, cvs ...), templating, extensibility of markup.

39.10.3. rwiki

FIXME:

RWiki is yet another WikiWiki Clone using RD format.

```
* fixed: RWiki-1.2.3 security hole
* RDtool-0.6.10 aware
* fixed: RWiki-1.2.2 security hole
* fixed: RWiki-1.2.1 security hole
```

39.10.4. tiki

FIXME:

Tiki is one of WikiEngines; WikiWiki is famous Web colaboration system. See http://c2.com/ .

- CGI program. (HTTP Server Common Gateway Interface) available under mod_ruby
- Basic Wiki grammer and natural extension.
- Text file based article management, not use database facility.
- Simple backup management. Builtin diff functionality and simple revision management.
- Digit and lowcase characters, Japanese kanji character supported as WikiName.
- Easy customization.
- User editable/defined InterWiki is invented and supported.
- Interactive action like adding comment on Wiki is supported.
- Plugin framework is supported to develop customized funcitionality of not only static but interactive action.
- Frame is supported to create context oriented page.
- Multiple nodes support enables multiple wiki sites run by one installation
- BSD type license

Seznam použité literatury pro kapitolu.

* *Pokusně použitý tag* `bibliography` *na konci kapitoly.*

[1] *IOWA.*

http://beta4.com/iowa/index.html

WEB (http://beta4.com/iowa/)

Kapitola 40. Grafická rozhraní, GUI

* *chapter id="gui" xreflabel="Grafická uživatelská rozhraní"*
* *Napsat krátké povídání o GUI obecně a zmínit možná/známá GUI jenž je možné z Ruby použít.*

40.1. wxRuby

* ***Attributy:*** *id="wxRuby"*

  Odkazy:

  - WxRuby (http://wxruby.org) — doména na prodej ???
  - wxRuby (http://rubyforge.org/projects/wxruby/)
  - wxRuby Documentation: Class Reference (http://wxruby.rubyforge.org/doc/)
  -
  - wxWidgets (http://wxwidgets.org/) Cross-Platform GUI Library
  -

  WxRuby je knihovna Ruby využívající knihovnu wxWidgets (http://wxwidgets.org/).

40.1.1. Instalace

```
# aptitude install libwxgtk2.8-0
# gem install wxruby
```

Ověření.

```
$ irb
irb(main):001:0> require 'rubygems'
=> true
irb(main):002:0> require 'wx'
=> true
```

Příklady se nacházejí `/var/lib/gems/1.8/gems/wxruby-2.0.0-x86_64-linux/samples`. Dokumentace pak v `...` dokumentace neexistuje :(.

Dalším způsobem odzkoušení je minimální aplikace která má tento kód:

```
#!/usr/bin/env ruby
require 'rubygems'
require "wx"
include Wx

class MinimalApp < App
  def on_init
    Frame.new(nil, -1, "The Bare Minimum").show()
  end
end

MinimalApp.new.main_loop
```

40.1.1.1. Ubuntu 10.04

*

Instalace WxRuby na Ubuntu bez Ruby. T.j. instalujeme úplně všechno.

```
# apt-get install ruby
# ruby -v
ruby 1.8.7 (2010-06-23 patchlevel 299) [i686-linux]
# apt-get install rubygems
# gem -v
1.3.7
# aptitude install libwxgtk2.8-0 libwxgtk2.8-dev
# gem install wxruby
Successfully installed wxruby-2.0.1-x86-linux
1 gem installed
Installing ri documentation for wxruby-2.0.1-x86-linux...
Installing RDoc documentation for wxruby-2.0.1-x86-linux...

$ irb
irb(main):001:0> require 'rubygems'
=> true
irb(main):002:0> require 'wx'
LoadError: libwx_gtk2u_media-2.8.so.0: cannot open shared object file: No such file or directory - /var/
 from /var/lib/gems/1.8/gems/wxruby-2.0.1-x86-linux/lib/wxruby2.so
 from /usr/lib/ruby/1.8/rubygems/custom_require.rb:31:in `require'
 from /var/lib/gems/1.8/gems/wxruby-2.0.1-x86-linux/lib/wx.rb:12
 from /usr/lib/ruby/1.8/rubygems/custom_require.rb:36:in `gem_original_require'
 from /usr/lib/ruby/1.8/rubygems/custom_require.rb:36:in `require'
 from (irb):2
 from /usr/lib/ruby/1.8/rubygems.rb:123
```

Postup končí chybou. Po chvíli hledání jsem našel příslušný bug report. ([#28372] require 'wxruby' failes on Ubuntu 10.04 (http://rubyforge.org/tracker/?func=detail&atid=218&aid=28372&group_id=35)). Dále postupujeme například instalací opravených balíčků z repositáře Maria Steele podle [wxruby-users] New Debian/Ubuntu repository! (http://rubyforge.org/pipermail/wxruby-users/2010-June/005496.html). Začneme odinstalováním špatných gem balíčků.

```
# gem uninstall wxruby


# wget -q http://repo.trilake.net/apt/repo.gpg -O- | sudo apt-key add -
# sudo wget http://repo.trilake.net/apt/sources.list.d/lucid.list \
        -O /etc/apt/sources.list.d/trilake.list

# apt-get update
# apt-get install wxruby

$ irb
irb(main):001:0> require 'rubygems'
=> true
irb(main):002:0> require 'wx'
LoadError: libwx_gtk2u_media-2.8.so.0: cannot open shared object file: No such file or directory - /var/
 from /var/lib/gems/1.8/gems/wxruby-2.0.1-x86-linux/lib/wxruby2.so
 from /usr/lib/ruby/1.8/rubygems/custom_require.rb:31:in `require'
 from /var/lib/gems/1.8/gems/wxruby-2.0.1-x86-linux/lib/wx.rb:12
 from /usr/lib/ruby/1.8/rubygems/custom_require.rb:36:in `gem_original_require'
 from /usr/lib/ruby/1.8/rubygems/custom_require.rb:36:in `require'
 from (irb):2
```

```
from /usr/lib/ruby/1.8/rubygems.rb:123
```

Pořád nefunguje!

```
# apt-get purge ruby
# apt-get autoremove
# apt-get purge wxruby
# apt-get install ri ruby-dev
# apt-get install wxruby
# # apt-get install rubygems
```

40.1.2. Tutoriál

*

Odkazy:

- WxRuby Tutorial (http://wxruby.rubyforge.org/wiki/wiki.pl?WxRuby_Tutorial)
- What is WxRuby? About WxWidgets and WxRuby (http://ruby.about.com/od/gui/a/wxwidgets.htm)
- Arranging Widgets in WxRuby (http://ruby.about.com/od/wxruby/qt/wxrubylayout.htm)
- What resources exist for WxRuby: documentation, tutorials, samples? (http://stackoverflow.com/questions/1662683/what-resources-exist-for-wxruby-documentation-tutorials-samples)
- Getting Started with the wxRuby GUI Toolkut (http://rubyonwindows.blogspot.com/2007/11/getting-started-with-wxruby-gui-toolkit.html)
-

V tomto tutoriálu postupně naprogramuji jednoduchý editor textu.

40.1.2.1. První okno, "Hello World"

Odkazy:

- Creating a "Hello World" Program With WxRuby (http://ruby.about.com/od/gui/qt/wxrubyworld.htm)

V první části si napíšeme základní kostru aplikace. Začneme obyčejným "hello" programem, jako je tento.

Příklad 40-1. Hello program ve wxRuby

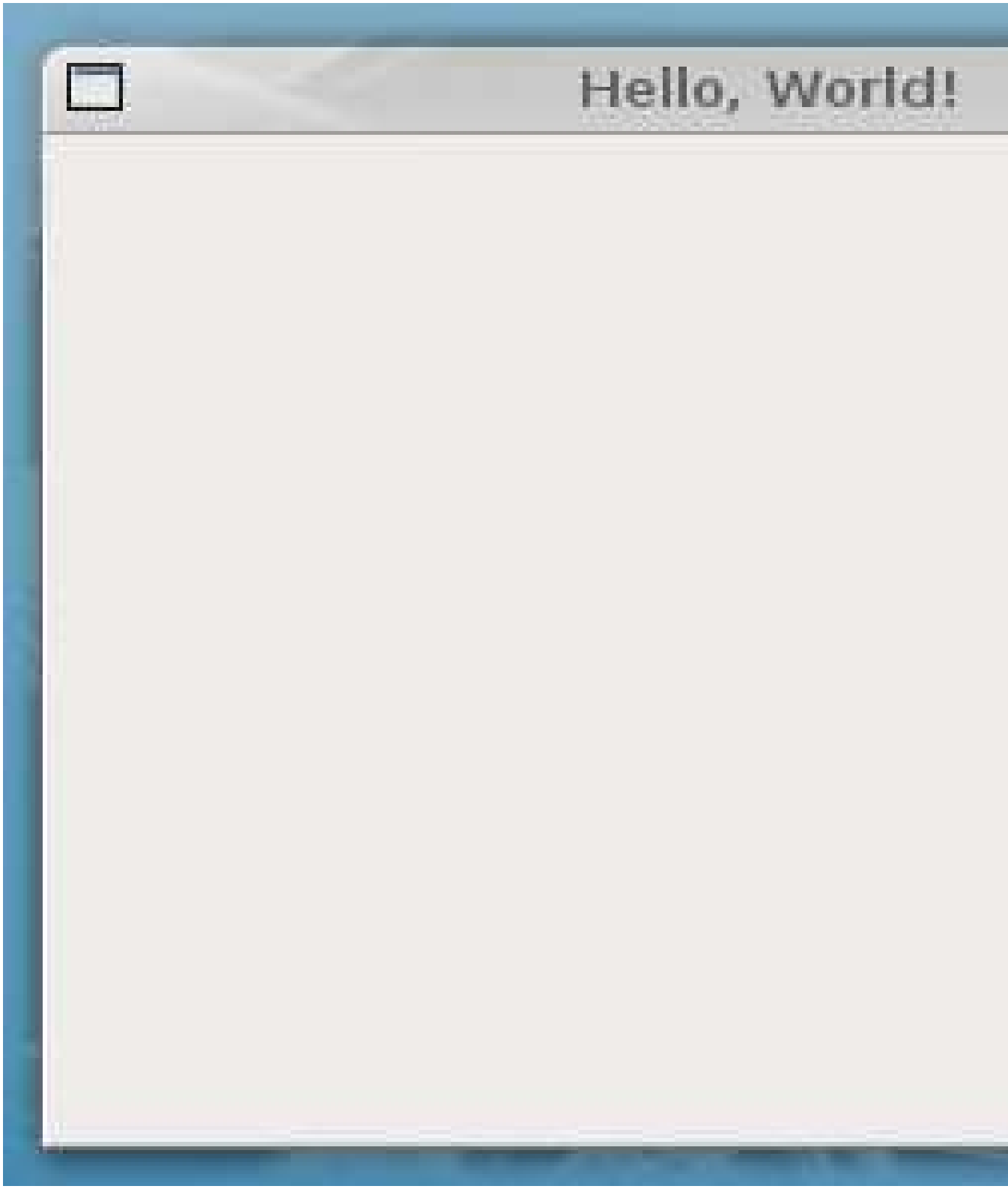
```
#!/usr/bin/env ruby
require 'rubygems'
require 'wx'

class MyApp < Wx::App
  def on_init
    frame = Wx::Frame.new(nil, -1,
                          :title => 'Hello, World!')
    frame.show
  end
end

app = MyApp.new
app.main_loop
```

A teď pár slov, jak program funguje. První řádek je známý `!#` řádek z Unixových systémů. Operační systém pozná že tento program má spouštět pomocí ruby. Následují dva řádky ve kterých říkáme, že je potřeba `rubygems` knihovna a samotná `wx` knihovna.

Program píšeme jako podtřídu třídy `Wx::App` které reprezentuje WxRuby aplikace. Do metody `on_init` naší třídy umístníme kód který se spouští při inicializaci aplikce. V tomto kódu vytvoříme pomocí volání `Wx::Frame.new` okno. První parametr každého volání je číslo rodičovského grafického prvku. Protože vytváříme samostatné okno, uvedeme jako hodnotu `nil`. Druhý paraametr každého volání je jedinečné ID objektu. Pokud použijeme `-1`, říkáme tím wxRuby že nemáme na ID objektu žádné zvláštní požadavky. wxRuby si vybere ID podle sebe. Z dalších parametrů jsem použil jen pojmenovaný parametr `:title` kterým nastavím oknu titulek.

A takto to vypadá.

Obrázek 40-1. Hello.rbw



A teď program trochu upravíme. Nejdříve vyjmeme hlavní rámec aplikace z metody `on_init` a napíšeme pro něj vlastní třídu. Protože tento rámec časem nebude mnoho metod, je to první krok jak je zvládnout.

```
class AppFrame < Wx::Frame
    def initialize
        super nil, :title => 'Nazdar světe!'
    end
end
```

Ve třídě aplikace, při spouštění programu, pak tento rámec použijeme.

```
def on_init
    @frame = AppFrame.new
    @frame.show
end
```

Po provedených úpravách vypadá kód takto:

Příklad 40-2. Hello program jinak

```
#!/usr/bin/env ruby
# -*- coding: utf-8; -*-
require 'rubygems'
require 'wx'

class AppFrame < Wx::Frame
  def initialize
    super nil, :title => 'Nazdar světe!'
  end
end

class MyApp < Wx::App
  def on_init
    @frame = AppFrame.new.show
  end
end

if $0 == __FILE__
  MyApp.new.main_loop
end
```

40.1.2.2. Menu

Odkazy:

- How Do You Create a Menu in WxRuby? (http://ruby.about.com/od/gui/a/wxrubymenu.htm)
- Appending Sub-Menus in WxRuby (http://ruby.about.com/od/wxruby/qt/wxsubmenu.htm)

Pro tvorbu menu se používá objekt `Wx::MenuBar`. Tento objekt obsahuje všechna menu a související objekty `Wx::Menu` pro roletky.

Tabulka 40-1. Zvláštní předdefinovaná ID

symbol	popis
`Wx::ID_ABOUT`	Žádost o otevření informační obrazovky
`Wx::ID_ANY`	Libovolná událost.

symbol	popis
`Wx::ID_EXIT`	Ukončení programu.
`Wx::ID_OPEN`	Otevření existujícího souboru
`Wx::ID_SAVE`	Uložení souboru.
`Wx::ID_LOWEST`	Nejnižší ID které WxRuby používá interně.
`Wx::ID_HIGHEST`	Nejvyšší ID které WxRuby používá interně.

Příklad 40-3. Menu program ve wxRuby

```
#!/usr/bin/env ruby
require 'rubygems'
require 'wx'

class MyApp < Wx::App
  def on_init
    @frame = Wx::Frame.new(nil, -1,
                           :title => 'Hello, World!')
    @frame.show

    menu = Wx::MenuBar.new
    file = Wx::Menu.new
    file.append(Wx::ID_ANY, "&Open\tAlt-O", "Open File")
    file.append(Wx::ID_EXIT, "E&xit\tAlt-X", "Quit")
    menu.append(file, "&File")
    @frame.menu_bar = menu

    evt_menu(Wx::ID_EXIT, :on_quit)
  end

  def on_quit
    @frame.close
  end
end

app = MyApp.new
app.main_loop
```

Příklad 40-4. Menu program jinak

```
#!/usr/bin/env ruby
require 'rubygems'
require 'wx'

class AppFrame < Wx::Frame
  def initialize
    super(nil, :title => 'Hello, World!')

    # Create Menu
```

```
        menu = Wx::MenuBar.new
        file = Wx::Menu.new
        file.append(Wx::ID_ANY, "&Open\tAlt-O", "Open File")
        file.append(Wx::ID_EXIT, "E&xit\tAlt-X", "Quit")
        menu.append(file, "&File")
        self.menu_bar = menu

        evt_menu(Wx::ID_EXIT, :on_quit)
      end

      def on_quit
        close
      end
    end

    class MyApp < Wx::App
      def on_init
        @frame = AppFrame.new
        @frame.show
      end
    end

    app = MyApp.new
    app.main_loop
```

Pokud potřebujeme menu znepřístupnit, použijeme metodu `enable` objektu `MenuItem`. Tato metoda má jen jeden parametr typu Boolean.

enable(Boolean enable = true)

```
    @save_menu_item = file_menu.append Wx::ID_SAVE
    @save_menu_item.enable false
```

Pro aktivaci a deaktivaci můžeme ještě použít volání metody `enable` objektu `MenuBar` nebo `Menu`. Tato metoda má dva parametry. Prvním je číslo volby menu (ID) a druhým je logická hodnota Boolean.

enable(Integer id, Boolean enable)

```
    self.menu_bar.enable(Wx::ID_SAVE, true)
```

40.1.2.3. Status

Odkazy:

- Adding a Status Bar to Your WxRuby Applications (http://ruby.about.com/od/wxruby/qt/wxstatusbar.htm)

Status Bar je lišta, která může být v každém panelu (`Wx::Frame`), v jeho spodní části. V souladu s názve slouží ke zobrazování stavových informací a krátkých hlášení. Vytváří se pomocí objektu `Wx::StatusBar` a do panelu se vkládá pomocí attributu `status` objektu `Wx::Frame`.

```
class AppFrame < Wx::Frame
    def initialize
        ...
        status = Wx::StatusBar.new(self)
        self.status_bar = status

        status.push_status_text "Status bar test"
```

```
    end
end
```

Příklad 40-5. Program s malým menu a status bar

```
#!/usr/bin/env ruby
require 'rubygems'
require 'wx'

class AppFrame < Wx::Frame
  def initialize
    super(nil, :title => 'Hello, World!')
    self.status_bar = Wx::StatusBar.new(self)
    self.status_bar.push_status_text "Status bar test"

    self.menu_bar = Wx::MenuBar.new
    file = Wx::Menu.new
    file.append(Wx::ID_EXIT)
    self.menu_bar.append(file, "&File")
  end
end

class MyApp < Wx::App
  def on_init
    @frame = AppFrame.new
    @frame.show
    evt_menu(Wx::ID_EXIT, :on_quit)
  end

  def on_quit
    @frame.close
  end
end

app = MyApp.new
app.main_loop
```

40.1.2.4. Scintilla

*

Odkazy:

- Wx::StyledTextCtrl (http://wxruby.rubyforge.org/doc/styledtextctrl.html)

Tak a dostali jsme se ke komponentě textového editoru Scintilla. Tento objekt se jmanuje `Wx::StyledTextCtrl`. Scintilla je sofistikovaný editor pro editaci textu. Má rozsáhlé možnosti nastavení a dovede takové věci jako: barevnou syntaxi, počítání řádků, sbalování a rozbalování bloků kód a mnohé další.

Příklad 40-6. Nejjednodušší editor se Scintillou

```
#!/usr/bin/env ruby
require 'rubygems'
require 'wx'

class Editor < Wx::Frame
  def initialize
```

```
      super nil
      @scintilla = Wx::StyledTextCtrl.new self
    end
  end

  class App < Wx::App
    def on_init
      frame = Editor.new
      frame.show
    end
  end

  app = App.new.main_loop
```

Takto jednoduchý program není použitelný. Neimplementovali jsme ani nejzákladnější dvě oprace, tedy načtění textu ze souboru a zápis textu zpět do souboru.

Nahrávání a zápis do souborů (Load and save to file (http://wxruby.rubyforge.org/doc/styledtextctrl.html#load_and_save_to_file)).

```
StyledTextCtrl.load_file
StyledTextCtrl.save_file
```

Samotný text mohu do objektu dostat metodami:

```
StyledTextCtrl.add_text
StyledTextCtrl.append_text
StyledTextCtrl.insert_text
StyledTextCtrl.clear_all
```

Jednoduché vložení textu pomocí `insert_text` je v následující ukázce. Nejdříve vymažu veškerý stávající obsah pomocí `clear_all` a poté vložím nový.

```
def initialize
    super nil
    @scintilla = Wx::StyledTextCtrl.new self
    @scintilla.clear_all
    @scintilla.insert_text(0, 'ěšč Předloha')
end
```

A teď zkusíme načíst soubor. Zatím zadáme jméno přímo do programu.

```
def initialize
    super nil
    @scintilla = Wx::StyledTextCtrl.new self
    @scintilla.load_file 'redit.rbw'
end
```

Než pokročím dál, potřebuji se naučit odchytávat událost ukončení editoru, a správně na ni zareagovat uložením rozpracovaného souboru.

```
def initialize
    ...
    # Bind events to code and/or methods
    evt_close :on_close
end

def on_close
    puts "Zavírám rámec editoru!"
```

```
    destroy
end
```

> **Varování**
>
> Pokud spouštíme aplikaci na MS Windows pomoci Ruby(GUI), není standardní výstup definován a například příkaz **puts** způsobí havárii programu.

40.1.3. AUI

* *id="wxRuby-AUI"*

Odkazy:

- AUI in WxRuby (http://www.prodevtips.com/2008/05/18/aui-in-wxruby/)
-
-
-

Pokročilé grafické rozhraní (AUI) je příklad použití wxRuby. V této části se jej pokusím rozebrat na části a ty diskutovat.

Spouštění aplikace v AUI, a samotná aplikace ja programována jako třída `AuiDemoApp`. Tato třída je vytvářena jako podtřída `Wx::App`. Ve třídě je definována jediná metoda `on_init`, která vytvoří hlavní rámec aplikace a zobrazí jej.

```
class AuiDemoApp < Wx::App
    def on_init
        frame = AuiFrame.new(...)
        set_top_window(frame)
        frame.show
        return true
    end
end

AuiDemoApp.new.main_loop()
```

40.1.4. Aplikace

* ***Attributy:*** *id="wxruby.App"*

Třída `Wx::App` reprezentuje celou aplikaci. Je to kontejner uvnitř nehož běží celý GUI kód. Používá se k udžování vlastností celé aplikace, implementuje událostní smyčku *event loop*, inicializuje aplikaci a umožňuje defaultní zpracování událostí které neošetřují objekty aplikace.

Několik způsobů zápisu programu pomocí třídy `Wx::App`.

```
Wx::App.run do
    frame = Wx::Frame.new(nil, :title => 'Jednoduchá aplikace'
    frame.show
end

class MyApp < Wx::App
    def on_init
        frame = Wx::Frame.new(nil, :title => 'Jednoduchá aplikace'
```

```
            frame.show
        end
    end

    app = MyApp.new
    app.main_loop
```

Seznam metod třídy `Wx::App`:

- `App.new`
- `App.run`
- `App#dispatch`
- `App#exit_main_loop`
- `App#filter_event`
- `App#get_app_name`
- `App#get_class_name`
- `App#get_exit_on_frame_delete`
- `App#get_top_window`
- `App#get_use_best_visual`
- `App#get_vendor_name`
- `App#is_active`
- `App#is_main_loop_running`
- `App#main_loop`
- `App#on_assert_failure`
- `App#on_exit`
- `App#on_init`
- `App#on_run`
- `App#pending`
- `App#set_app_name`
- `App#set_class_name`
- `App#set_exit_on_frame_delete`
- `App#set_top_window`
- `App#set_use_best_visual`
- `App#set_vendor_name`
- `App#yield`

40.1.5. Testování v aplikaci (*XP*)

*

Odkazy:

- BDD WxRuby applications with Cucumber and Nobbie (http://bryan-ash.blogspot.com/2009/02/bdd-wxruby-applications-with-cucumber.html) [2009-02-14]
- Wx-Nobbie (http://github.com/bryan-ash/wx-nobbie/tree) — An implementation of the Nobbie GUI Driving API for WxRuby2
-
-
-

40.1.6. Sizers

*

Přehled

- BoxSizer
- GridSizer
- FlexSizer
-

40.1.6.1. BoxSizer

*

Nejjednodušší *sizer*, rozmístňuje objekty v jednom sloupci nebo jednom řádku.

40.1.7. Události (*Events*)

* ***Attributy:*** *id="wxRuby.Events"*

Seznam událostí:

- `ActivateEvent`
- `CalendarEvent`
- `CalculateLayoutEvent`
- `ChildFocusEvent`
- `CloseEvent`
- `CommandEvent`
- `ContextMenuEvent`
- `DateEvent`
- `EraseEvent`
- `Event`
- `FindDialogEvent`
- `FocusEvent`
- `KeyEvent`
- `IconizeEvent`
- `IdleEvent`
- `ListEvent`
- `MenuEvent`
- `MouseEvent`
- `MoveEvent`
- `NavigationKeyEvent`
- `NotifyEvent`
- `PaintEvent`
- `QueryLayoutInfoEvent`
- `RichTextEvent`
- `ScrollEvent`
- `ScrollWinEvent`
- `SizeEvent`
- `SpinEvent`
- `SplitterEvent`

- `TextUrlEvent`
- `TimerEvent`
- `TreeEvent`
- `UpdateUIEvent`
- `WindowCreateEvent`
- `WindowDestroyEvent`
- `WizardEvent`

40.1.8. Popisy některých objektů

*

Odkazy:
- wxRuby Documentation: Class Reference (http://wxruby.rubyforge.org/doc/)
-

40.1.8.1. Wx::Frame

*

Odkazy:
- Frame.new (http://wxruby.rubyforge.org/doc/frame.html#Frame_new)
-

```
Frame.new(Window parent, Integer id, String title, Point size = DEFAULT_SIZE, Integer style = DE
```

Parametry:
- *`parent`* rodič (předek) okna. Tento parametr může mít hodnotu `nil`. Není-li `nil`, bude okno zobrazeno nad rodičovským oknem.
- *`id`* identifikátor okna. Může mít hodnotu `-1`, což znamená že bude použita implicitní hodnota.
- *`title`*
- *`pos`*
- *`size`*
- *`style`*
- *`name`*

40.2. Ruby Gnome2

*

Odkazy:
- Oficiální stránka projektu Ruby-GNOME2 (http://ruby-gnome2.sourceforge.jp/)
-

Rozmístňování prvků se provádí pomocí:

- Gtk::HBox
- Gtk::VBox
- Gtk::Table
-

40.3. FXRuby

* *section id="fxruby" xreflabel="FXRuby"*

Odkazy a zdroje:

- Hlavní sídlo FXRuby na SourceForge (http://fxruby.sourceforge.net/)
- FOX Toolkit (http://www.fox-toolkit.org/)
- Hugh Sasse's FXRuby Page (http://www.eng.cse.dmu.ac.uk/~hgs/ruby/FXRuby/)
- Dokumentace API FXRuby (http://fxruby.sourceforge.net/doc/api/)
- FXBook: the FOX Toolkit Documentation Project (http://fxbook.sourceforge.net/)
- Lyle Johnson: Developing Graphical User Interfaces with FXRuby (http://dev.faeriemud.org/FXRuby/)
- Defuse, jednoduchá hra používající FXRuby (http://www.laukamm.de/lars/defuse.htm)
- Tutorial 1 Skeleton App (http://www.fifthplanet.net/cgi-bin/wiki.pl?Tutorial_1_Skeleton_App)
- ___ (http://___)

FXRuby je rozšiřující modul s rozhraním do FOX Toolkitu (http://www.fox-toolkit.org/)

40.3.1. Instalace

Instal

40.3.1.1. Debian Lenny

Odkazy:

-

```
# aptitude install libfox-1.6-dev
# aptitude install g++
# aptitude install libxrandr-dev
# gem install fxruby
Building native extensions.  This could take a while...
Successfully installed fxruby-1.6.19
1 gem installed
Installing ri documentation for fxruby-1.6.19...
```

V případě jakýchkoliv problémů se místo hlášek o úspěšné instalaci gemu zobrazí informace o chybě. Přesněji informace o souboru ve kterém je protokol o překladu.

```
Gem files will remain installed in /var/lib/gems/1.8/gems/fxruby-1.6.19 for inspection.
Results logged to /var/lib/gems/1.8/gems/fxruby-1.6.19/ext/fox16/gem_make.out
```

Po odstranění/vyřešení problému se můžeme znovu pokusit o nainstalování gemu.

Ve skutečném životě se mi to na první pokus taktéž nepodařilo. K instalaci balíčků, které se nacházejí před vlastní instalací gemu, mě navedly právě chaybové výstupy. S trochou googlování jsem pak přišel na to co mi chybí.

Poznámka: Použitý gem je standardní z Debianu.

Knihovna libxrandr je X11 RandR extension library. Poskytuje klientovi přístup k RandR rozšíření X protokolu. Toto rozšíření umožňuje konfigurovat za běhu vlastnosti displeje jako jsou rozlišení, otočení, zrcadlení.

Úspešnou instalaci si můžeme ověřit například z irb

```
$ irb
irb(main):001:0> require 'rubygems'
=> true
irb(main):002:0> require 'fox16'
=> true
```

Knihovna fox16 je tedy dostupná. Hned vyzkoušíme tento malý program.

```
#!/usr/bin/env ruby
require 'rubygems'
require "fox16"
include Fox

application = FXApp.new
mainWindow = FXMainWindow.new(application, "Ahoj")
FXLabel.new(mainWindow, "Ahoj světe")
application.create
mainWindow.show(PLACEMENT_SCREEN)
application.run
```

40.3.1.2. Debian Etch

Odkazy

- Debian + FXRI + Rubygems is Englightenment! (http://angryruby.blogspot.com/2007/08/debian-fxri-rubygems-is-englightenment.html)

V Debian Etch není přímo balíček s FXRuby. K instalaci můžeme použít gem. Nejdříve si připravíme potřebné vývojové knihovny.

```
# aptitude install libfox-1.6-dev ruby1.8-dev
```

A poté nainstalujem samotný gem. Ten se při instalaci přeloží s použitím vývohových knihoven které jsme si nainstalovali předem.

```
$ gem install fxruby
```

Malá ukázka použití FXRuby.

```
#!/usr/bin/env ruby

require 'rubygems'
require 'fox16'

include Fox
application = FXApp.new

main = FXMainWindow.new(application, "Dnešní datum")
button = FXButton.new(main, "Hello, world!")
button.connect(SEL_COMMAND) { application.exit }

application.create
main.show(PLACEMENT_SCREEN)
application.run
```

40.3.1.3. Instalace ze zdrojových kódů

40.3.1.3.1. Jak získat FOX

Knihovnu gui FOX je možno nainstalovat z balíčků, například na Debian Woody je jako balíček `libfox0.99`, nebo stáhnout zdrojové kódy z http://www.fox-toolkit.org a skompilovat si knihovnu sám.

Stažení zdrojových kódů knihovny.

```
$ cd $HOME/arch/gui/fox
$ wget http://www.fox-toolkit.org/ftp/fox-1.0.29.tar.gz
```

Překlad aktuální verze 1.0.29 (2003-01-07) knihovny ze stabilní řady 1.0.x.

```
$ cd $HOME/source
$ tar xzf ~/arch/gui/fox/fox-1.0.29.tar.gz
$ cd fox-1.0.29
$ mkdir $HOME/opt/fox-1.0.29
$ ./configure --prefix=$HOME/opt/fox-1.0.29        0:01:29
$ make                                             cca 3hod
$ make install
```

40.3.1.3.2. Jak získat FXRuby

Zajisté existují i binární distribuce FXRuby, a bylo by dobré se o nich zmínit.

Před vlastní kompilací a instalací FXRuby jsem zkomiloval nejdříve FOX GUI.

Postup při překladu byl následující. Nejdříve jsme stáhl a rozbalil FXRuby verzi 1.0.16 jenž jsem získal na Source Forge (http://fxruby.sourceforge.net/). V rozbaleném adresáři jsem pak spustil

```
$ cd source/ruby/FXRuby-1.0.16
$ ruby install.rb config -- --with-fox-include=$HOME/include \
                           --with-fox-lib=$HOME/lib
$ ruby install.rb setup
$ ruby install.rb install
```

Úspěšnost instalace si ověříme v irb

```
# $Id: fxruby-test.ses,v 1.1 2002/12/16 20:34:12 radek Exp $
require 'fox'LoadError: no such file to load -- fox
 from (irb):1:in `require'
 from (irb):1
```

* *condition="author"*

Popis překladu FXRuby-1.0.17 do Ruby cvs 2002-12-17

```
12345678901234567890123456789012345678901234567890123456789012345678901234567890
$ cd $HOME/source
$ tar xzf ~/arch/lang/ruby/fxruby/FXRuby-1.0.17.tar.gz
$ cd FXRuby-1.0.17/
$ ruby install.rb config -- --with-fox-include=$HOME/include \
                           --with-fox-lib=$HOME/lib 0:00:24
$ ruby install.rb setup       2:57:23
$ ruby install.rb install     0:00:44
```

Výsledný produkt nefunguje. Po pokusu o test skončí chybou.

```
$ cd test
$ ruby TS_All.rb
/home/radek/opt/ruby-2002.12.17/lib/ruby/site_ruby/1.7/i586-linux/fox.so: /home/radek/opt/ruby-2002.12.1
        from ./TC_FXFileStream.rb:2
        from TS_All.rb:21:in ‘require’
        from TS_All.rb:21
        from TS_All.rb:20:in ‘each’
        from TS_All.rb:20
Loaded suite TS_All
Started

Failure!!!
run:
No tests were run.

Finished in 0.071742 seconds.
0 tests, 0 assertions, 1 failures, 0 errors
radek@kvark:~/source/FXRuby-1.0.17/tests$
```

Varování

Domnívám se, že FXRuby verze 1.0.x, aktuálně 1.0.17 lze používat jen z Fox verze 1.0.x a nikoliv z vývojovou řadou 1.1.x

40.3.2. Malý tutoriál

* *section id="fxruby.tutorial" xreflabel="Malý tutoriál"*

Toto je takový malý tutoriál. Postupně procházím jednotlivé komponenty které mne zajímají a popisuji jejich užití na příkladech.

40.3.2.1. Kostra aplikace

* *section id="fx-app-skeleton" xreflabel="Kostra aplikace"*

První věcí kterou proberu je základní kostra aplikace. Bez té nemá smysl se věnovat dalšímu. Tak tedy aplikace může být posloupností příkazů jenž provedou konstrukci nezbytných objektů. Ukážeme si to na velmi jednoduché aplikaci. Nejdříve si vše odzkoušíme interaktivně:

```
# $Id: fxruby-hello.ses,v 1.1 2003/11/03 08:05:10 radek Exp $
require ’rubygems’
true
require ’fox16’
LoadError: no such file to load -- fox16
 from /home/radek/lib/rubygems/lib/rubygems/custom_require.rb:31:in ‘gem_original_require’
 from /home/radek/lib/rubygems/lib/rubygems/custom_require.rb:31:in ‘require’
 from (irb):2
include Fox
NameError: uninitialized constant Fox
 from (irb):3
```

```
app = FXApp.new
NameError: uninitialized constant FXApp
 from (irb):5
win = FXMainWindow.new(app, "Ahoj", nil, nil, DECOR_ALL, 0, 0,
88, 21)
NameError: uninitialized constant FXMainWindow
 from (irb):6
FXButton.new(win, "Konec", nil, app, FXApp::ID_QUIT)
NameError: uninitialized constant FXButton
 from (irb):8
app.create
NoMethodError: undefined method ‘create’ for nil:NilClass
 from (irb):9
win.show(PLACEMENT_SCREEN)
NameError: uninitialized constant PLACEMENT_SCREEN
 from (irb):10
app.run
NoMethodError: undefined method ‘run’ for nil:NilClass
 from (irb):11
```

Po spuštění aplikace příkazem **app.run** se objeví okno s tlačítkem. Zmáčknutím tlačítka se okno ukončí.

Vyzkoušeli jsme si že FXRuby funguje a nyní si napíšeme jednoduchou aplikaci Ahoj. Aplikace zobrazí ve svém formuláři nápis „Ahoj světe“.

```
#!/usr/bin/env ruby
# $Id: hello.rb,v 1.1 2003/10/31 08:45:16 radek Exp $
# $Source: /home/radek/cvs/ruby-book/example/gui/fxruby/hello.rb,v $
require "fox"                                                    ❶
include Fox                                                      ❷

application = FXApp.new
application.normalFont = FXFont.new(application,                 ❸
  "-*-helvetica-bold-r-normal-*-12-*-*-*-*-*-iso8859-2")
mainWindow = FXMainWindow.new(application, "Ahoj", nil, nil, DECOR_ALL,❹
                              0, 0, 88, 21)
FXLabel.new(mainWindow, "Ahoj světe")
application.create
mainWindow.show(PLACEMENT_SCREEN)
application.run                                                  ❺
```

❶ Potřebujeme knihovnu `fox`.

❷ Z této knihovny přímo includujeme modul Fox.

❸ Protože používáme české znaky je třeba nastavit font v kódování `ISO-8859-2`.

❹ Vytvoříme hlavní okno aplikace.

❺ Spustíme smyčku vyhodnocování událostí.

Zkusme nyní kód zjednodušit.

```
#!/usr/bin/env ruby
# $Id: hello3.rb,v 1.1 2005/10/04 08:52:07 radek Exp $
# $Source: /home/radek/cvs/ruby-book/example/gui/fxruby/hello3.rb,v $
require "fox"
include Fox

class MyApp < FXApp
    def initialize
        super
        @normalFont = FXFont.new(app,
                "-*-helvetica-bold-r-normal-*-12-*-*-*-*-*-iso8859-2")
        init ARGV
    end
end

class MyWin < FXMainWindow
    def initialize(app)
        super(app, "Ahoj3", nil, nil, DECOR_ALL, 0, 0, 88, 21)
        FXLabel.new(self, "Ahoj sv<65533>te")
    end

    def create
        super
        show(PLACEMENT_SCREEN)
    end
end

FXApp.new do |app|
    win = MyWin.new(app)
    app.create
    app.run
end;
```

Protože při konstrukci složitějších objektů začne být kód méně přehledný, zapouzdříme jednotlivé části do objektů.

```
#!/usr/bin/env ruby
# $Id: helloapp.rb,v 1.3 2003/11/10 09:46:43 radek Exp $
# $Source: /home/radek/cvs/ruby-book/example/gui/fxruby/helloapp.rb,v $
require 'rubygems'
require "fox16"
include Fox

class MyApp < FXApp
    def initialize
 super("Hello App", "radek")
 init(ARGV)
 win = MyWin.new(self)
 create
```

```
        end
    end

    class MyWin < FXMainWindow
        def initialize(app)
            super(app, "Ahoj", nil, nil, DECOR_ALL, 0, 0, 88, 21)
            FXButton.new(self, "Konec", nil, app, FXApp::ID_QUIT)
        end

        def create
            super
            show(PLACEMENT_SCREEN)
        end
    end

    if __FILE__ == $0
        MyApp.new.run
    end;
```

40.3.2.2. Jednoduchý vstup

Zatím jsme si ukázali jak vytvořit okno. Teď si předvedeme jak realizovat jednoduchý vstup. Použijeme komponentu `Fox::FXTextField`

```
    #!/usr/bin/env ruby
    # $Id: textfield.rb,v 1.1 2003/11/03 18:22:20 radek Exp $
    # $Source: /home/radek/cvs/ruby-book/example/gui/fxruby/textfield.rb,v $
    require "fox"
    include Fox

    class FormApp < FXApp
    end

    class FormWin < FXMainWindow
        def initialize(app)
            super(app, "TextField", nil, nil, DECOR_ALL, 0, 0, 120, 32)

            FXTextField.new(self, 16).connect(SEL_COMMAND) do
                |sender, selector, data|
                puts data
                exit
            end
        end

        def create
            super
            show(PLACEMENT_SCREEN)
        end
    end

    formApp = FormApp.new
    formApp.init(ARGV)
    formWin = FormWin.new(formApp)
    formApp.create
    formApp.run;
```

40.3.2.3. Rozdělení plochy formuláře

* *section id="fxruby.layout" xreflabel="Správci rozvržení"*

Odkazy a zdroje:

- Placing Widgets (http://www.fox-toolkit.org/layout.html)
- ___ (http://___)

Jednou z důležitých komponenet je správce rozložení plochy formuláře *Layout Manager*. Ten určuje přesné umístněnní jednotlivých prvků na formuláři podle námi zadaných kritérií. Co to znamená si ukážme na jednotlivých příkladech.

Na výběr máme řadu správců, v krátkosti jsou to

- `Fox::FX4SPlitter` — rozděluje plochu na čtyři čtvrtiny
- `Fox::FXGroup` — FIXME:
- `Fox::FXHorizontalFrame` — FIXME:
- `Fox::FXMatrix` — FIXME:
- `Fox::FXPacker` — FIXME:
- `Fox::FXSplitter` — FIXME:
- `Fox::FXSwitcher` — FIXME:
- `Fox::FXTopWindow` — FIXME:
- `Fox::FXVerticalFrame` — FIXME:

* *para condition="author"*

Následující texty jsou převzaty z dokumentace (http://fxruby.sourceforge.net/doc/api/) a Layout Managers (http://www.fox-toolkit.org/layout.html)

Správci rozvržení

`Fox::FX4Splitter`

The four-way splitter is a layout manager which manages four children like four panes in a window. You can use a four-way splitter for example in a CAD program where you may want to maintain three orthographic views, and one oblique view of a model. The four-way splitter allows interactive repartitioning of the panes by means of moving the central splitter bars. When the four-way splitter is itself resized, each child is proportionally resized, maintaining the same split-percentage.

The four-way splitter is a layout manager which manages four children like four panes in a window. You can use a four-way splitter for example in a CAD program where you may want to maintain three orthographic views, and one oblique view of a model. The four-way splitter allows interactive repartitioning of the panes by means of moving the central splitter bars. When the four-way splitter is itself resized, each child is proportionally resized, maintaining the same split-percentage. The four-way

splitter widget sends a SEL_CHANGED to its target during the resizing of the panes; at the end of the resize interaction, it sends a SEL_COMMAND to signify that the resize operation is complete.

`Fox::FXGroupBox`

An FXGroupBox widget provides a nice raised or sunken border around a group of widgets, providing a visual delineation. Typically, a title is placed over the border to provide some clarification. Radio buttons placed inside a group box automatically assume mutually exclusive behaviour, i.e. at most one radio button will be checked at any one time.

The GroupBox is a layout manager that provides identical layout facilities as the Packer. In addition, the GroupBox draws an attractive border around its contents, and provides an optional caption. Finally, if its children are RadioButtons, it forces at most one of these to be checked.

`Fox::FXHorizontalFrame`

The HorizontalFrame layout manager packs its children horizontally from left to right (or right to left).

The horizontal frame layout manager widget is used to automatically place child-windows horizontally from left-to-right, or right-to-left, depending on the child windows? layout hints.

`Fox::FXMatrix`

The FXMatrix layout manager automatically arranges its child windows in rows and columns. If the matrix style is MATRIX_BY_ROWS, then the matrix will have the given number of rows and the number of columns grows as more child windows are added; if the matrix style is MATRIX_BY_COLUMNS, then the number of columns is fixed and the number of rows grows as more children are added. If all children in a row (column) have the LAYOUT_FILL_ROW (LAYOUT_FILL_COLUMN) hint set, then the row (column) will be stretchable as the matrix layout manager itself is resized. If more than one row (column) is stretchable, the space is apportioned to each stretchable row (column) proportionally. Within each cell of the matrix, all other layout hints are observed. For example, a child having LAYOUT_CENTER_Y and LAYOUT_FILL_X hints will be centered in the Y-direction, while being stretched in the X-direction. Empty cells can be obtained by simply placing a borderless FXFrame widget as a space-holder.

The Matrix layout manager arranges its children in rows and columns. An FXMatrix widget can operate in both column-oriented as well as row-oriented mode. Normally, the Matrix layout manager operates row-wise. Based on the number of rows, the Matrix layout determines the width of each column and the height of each row, then arranges the children in the space allotted, observing the child's layout hints as much as possible.

`Fox::FXPacker`

FXPacker is a layout manager which automatically places child windows inside its area against the left, right, top, or bottom side. Each time a child is placed, the remaining space is decreased by the amount of space taken by the child window. The side against which a child is placed is determined by the LAYOUT_SIDE_TOP, LAYOUT_SIDE_BOTTOM, LAYOUT_SIDE_LEFT, and LAYOUT_SIDE_RIGHT hints given by the child window. Other layout hints from the child are observed as far as sensible. So for example, a child placed against the right edge can still have LAYOUT_FILL_Y or LAYOUT_TOP, and so on. The last child may have both LAYOUT_FILL_X and LAYOUT_FILL_Y, in which case it will be placed to take all remaining space.

The Packer layout widget places its GUI elements in its interior rectangle, placing each child against one of the four sides. As each child is placed, the size of the rectangle is reduced by the space taken up by the child. If a child is placed against the left or right, the interior rectangle's width is reduced; if the child is placed against the top or bottom, the height is reduced. Children may be of any type, including other layout managers.

`Fox::FXSplitter`

The Splitter layout manager divides some area of the screen horizontally or vertically. The divider bars can be repositioned by the user, so that depending on what the user is doing, he or she may give one or the other partition more screen space.

Splitter window is used to interactively repartition two or more subpanes. Space may be subdivided horizontally or vertically. When the splitter is itself resized, the right-most (bottom-most) child window will be resized unless the splitter window is reversed; if the splitter is reversed, the left-most (top-most) child window will be resized instead. The splitter widget sends a SEL_CHANGED to its target during the resizing of the panes; at the end of the resize interaction, it sends a SEL_COMMAND to signify that the resize operation is complete. Normally, children are resizable from 0 upwards; however, if the child in a horizontally oriented splitter has LAYOUT_FILL_X in combination with LAYOUT_FIX_WIDTH, it will not be made smaller than its default width, except when the child is the last visible widget (or first when the option SPLITTER_REVERSED has been passed to the splitter). In a vertically oriented splitter, children with LAYOUT_FILL_Y and LAYOUT_FIX_HEIGHT behave analogously. These options only affect interactive resizing.

`Fox::FXSwitcher`

The FXSwitcher layout manager automatically arranges its child windows such that one of them is placed on top; all other child windows are hidden. Switcher provides a convenient method to conserve screen real-estate by arranging several GUI panels to appear in the same space, depending on context. Switcher ignores all layout hints from its children; all children are stretched according to the switcher layout managers own size. When the SWITCHER_HCOLLAPSE or SWITCHER_VCOLLAPSE options are used, the switcher?s default size is based on the width or height of the current child, instead of the maximum width or height of all of the children.

The Switcher layout manager places its children exactly on top of each other; it ignores most of the layout hints provided by each child. You typically use a layout manager like the switcher to save screen real-estate, by placing for example several control panels on top of each other, and bringing the right one on top depending on the context.

`Fox::FXTopWindow`

TopWindow operates like an FXPacker window. For simple dialogs and toplevel windows, no additional layout managers may be needed in many cases, as the TopWindow's layout characteristics may be sufficient.

Abstract base class for all top-level windows.

`Fox::FXVerticalFrame`

The VerticalFrame layout manager packs its children vertically, from top to bottom or vice versa. It behaves similar to the HorizontalFrame layout manager.

Vertical frame layout manager widget is used to automatically place child-windows vertically from top-to-bottom, or bottom-to-top, depending on the child window's layout hints.

40.3.2.3.1. Fox::FXHorizontalFrame

* *section id="FXHorizontalFrame" xreflabel="Fox::FXHorizontalFrame"*

Umístňuje jednotlivé komponenty vodorovně (horizontálně) z leva do prava nebo opačně.

```
#!/usr/bin/env ruby
# $Id: horizontalframe1.rb,v 1.1 2003/11/03 18:22:20 radek Exp $
```

```
# $Source: /home/radek/cvs/ruby-book/example/gui/fxruby/horizontalframe1.rb,v $
require "fox"
include Fox

class MyApp < FXApp
end

class MyWin < FXMainWindow
    def initialize(app)
        super(app, "Horizontal Frame", nil, nil, DECOR_ALL, 0, 0, 190, 33)
        FXHorizontalFrame.new(self) do |frame|
            FXLabel.new(frame, "Hodnota:")
            FXTextField.new(frame, 16).connect(SEL_COMMAND) do
                |sender, selector, data|
                puts data
                exit
            end
        end
    end

    def create
        super
        show(PLACEMENT_SCREEN)
    end
end

app = MyApp.new
app.init(ARGV)
form = MyWin.new(app)
app.create
app.run
```

Výsledek poté vypadá takto

40.3.2.3.1.1. Třída `Fox::FXHorizontalFrame`

Metody třídy

`new(`*`p`*`, `*`opts`*`=0, `*`x`*`=0, `*`y`*`=0, `*`w`*`=0, `*`h`*`=0, `*`pl`*`=DEFAULT_SPACING, `*`pr`*`=DEFAULT_SPACING, `*`pt`*`=DEFAULT_SPACING, `*`pb`*`=DEFAULT_SPACING, `*`hs`*`=DEFAULT_SPACING, `*`vs`*`=DEFAULT_SPACING, ){|theHorizontalFrame| ... }`

Jednotlivé parametry znamenají:

- *`p`* — rodičovské okno komponenty
- *`opts`* — volby rámce Integer
- *`x`*, *`y`* — počáteční pozice Integer

- `w`, `h` — šířka a výška Integer
- `pl`, `pr`, `pt`, `pb` — vnitřní výplň (mezera) vlevo, vpravo nahoře a dole v bodechInteger
- `hs`, `vs` — vodorovná (horizontální) a svislá (vertikální) mezera mezi komponentami, uvedeno v bodech Integer

40.3.2.3.2. `Fox::FXVerticalFrame`

Manažer rozvržení `Fox::FXVerticalFrame` je obdobou manažeru Fox::FXHorizontalFrame jediný rozdíl je ve směru umístňování komponent. Zatímco Fox::FXHorizontalFrame rozmístňuje vodorovně (horizontálně), `Fox::FXVerticalFrame` provádí rozmísntění ve směru svislém (vertikálním). Opět je možno si určit bude-li se tak dít shora dolů, nebo zdola nahoru.

V následující ukázce umístníme na formulář pod sebe čtyři prvky: nápis, vstupní pole a dvě tlačítka. Prvky jsou zarovnány na levý okraj.

```
#!/usr/bin/env ruby
# $Id: verticalframe1.rb,v 1.1 2003/11/03 18:22:20 radek Exp $
# $Source: /home/radek/cvs/ruby-book/example/gui/fxruby/verticalframe1.rb,v $
require "fox"
include Fox

class MyApp < FXApp
end

class MyWin < FXMainWindow
    def initialize(app)
        super(app, "Horizontal Frame", nil, nil, DECOR_ALL, 0, 0, 155, 104)
        FXVerticalFrame.new(self) do |frame|
            FXLabel.new(frame, "Hodnota:")
            FXTextField.new(frame, 16).connect(SEL_COMMAND) do
                |sender, selector, data|
                puts data
                exit
            end
            FXButton.new(frame, "Konec", nil, app, FXApp::ID_QUIT)
            FXButton.new(frame, "Taky konec", nil, app, FXApp::ID_QUIT)
        end
    end

    def create
        super
        show(PLACEMENT_SCREEN)
    end
end

app = MyApp.new
app.init(ARGV)
form = MyWin.new(app)
app.create
app.run
```

Jak je vidět na obrázku, jednotlivé komponenty jsou umísntěny pěkně pod sebou.

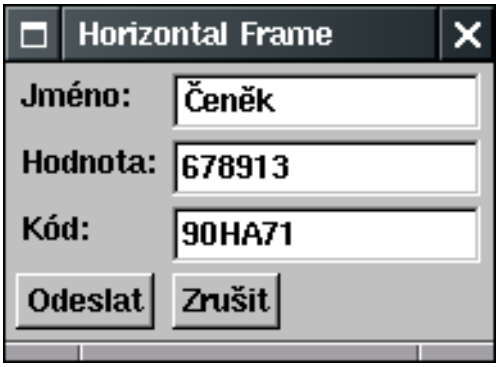
40.3.2.3.3. `Fox::FXMatrix`

Manažer rozložení komponent `Fox::FXMatrix`, jak již název napovídá rozmístňuje komponenty do pravidelné mřížky a umožňuje nám vytvořit formulář kde komponenty „zařezávají“ jak vodorovně tak svisle.

```
#!/usr/bin/env ruby
# $Id: matrix1.rb,v 1.2 2003/11/04 10:00:18 radek Exp $
# $Source: /home/radek/cvs/ruby-book/example/gui/fxruby/matrix1.rb,v $
require "fox"
include Fox

class MyApp < FXApp
end

class MyWin < FXMainWindow
    def initialize(app)
        super(app, "Horizontal Frame", nil, nil, DECOR_ALL, 0, 0, 190, 108)
        FXMatrix.new(self, 2, MATRIX_BY_COLUMNS|LAYOUT_BOTTOM) do |frame|
            FXLabel.new(frame, "Jm<65533>no:")
            FXTextField.new(frame, 16).connect(SEL_COMMAND) do
                |sender, selector, data|
                puts data
            end

            FXLabel.new(frame, "Hodnota:")
            FXTextField.new(frame, 16).connect(SEL_COMMAND) do
                |sender, selector, data|
                puts data
            end

            FXLabel.new(frame, "K<65533>d:")
            FXTextField.new(frame, 16).connect(SEL_COMMAND) do
                |sender, selector, data|
                puts data
            end

            FXButton.new(frame, "Odeslat", nil, app, FXApp::ID_QUIT)
            FXButton.new(frame, "Zru<65533>it", nil, app, FXApp::ID_QUIT)
        end
    end

    def create
```

```
            super
            show(PLACEMENT_SCREEN)
        end
    end

    if __FILE__ == $0
        MyApp.new do |app|
            app.normalFont = FXFont.new(app, "-*-helvetica-bold-r-normal-*-12-*-*-*-*-*-iso8859-2"
            app.init(ARGV)
            MyWin.new(app)
            app.create
            app.run
        end
    end
```

Na obrázku je vidět jak jsou jednotlivé komponenty rozmístněny.



40.3.2.3.3.1. Třída `Fox::FXHorizontalFrame`

Metody třídy

`new(` *`parent`*`,` *`n`*`=1,` *`opts`*`=MATRIX_BY_ROWS,` *`x`*`=0,` *`y`*`=0,` *`width`*`=0,` *`height`*`=0,` *`padLeft`*`=DEFAULT_SPACING,` *`padRight`*`=DEFAULT_SPACING,` *`padTop`*`=DEFAULT_SPACING,` *`padBottom`*`=DEFAULT_SPACING,` *`hSpacing`*`=DEFAULT_SPACING,` *`vSpacing`*`=DEFAULT_SPACING) {|theMatrix| ...}`

The FXMatrix layout manager automatically arranges its child windows in rows and columns. If the matrix style is MATRIX_BY_ROWS, then the matrix will have the given number of rows and the number of columns grows as more child windows are added; if the matrix style is MATRIX_BY_COLUMNS, then the number of columns is fixed and the number of rows grows as more children are added. If all children in a row (column) have the LAYOUT_FILL_ROW (LAYOUT_FILL_COLUMN) hint set, then the row (column) will be stretchable as the matrix layout manager itself is resized. If more than one row (column) is stretchable, the space is apportioned to each stretchable row (column) proportionally. Within each cell of the matrix, all other layout hints are observed. For example, a child having LAYOUT_CENTER_Y and LAYOUT_FILL_X hints will be centered in the Y-direction, while being stretched in the X-direction. Empty cells can be obtained by simply placing a borderless FXFrame widget as a space-holder. Matrix packing options MATRIX_BY_ROWS: Fixed number of rows, add

columns as needed MATRIX_BY_COLUMNS: Fixed number of columns, adding rows as needed Jednotlivé parametry znamenají:

- `p` — rodičovské okno komponenty
- `opts` — volby rámce Integer
- `x`, `y` — počáteční pozice Integer
- `w`, `h` — šířka a výška Integer
- `pl`, `pr`, `pt`, `pb` — vnitřní výplň (mezera) vlevo, vpravo nahoře a dole v bodechInteger
- `hs`, `vs` — vodorovná (horizontální) a svislá (vertikální) mezera mezi komponentami, uvedeno v bodech Integer

Construct a matrix layout manager with n rows or columns

40.3.2.4. Menu

* *section id="fxruby.menu" xreflabel="Menu"*

Další věcí které bych rád věnoval svou pozornost je systém menu.

`Fox::FXMenubar`

FIXME:

40.3.2.4.1. Menubar

Začneme tím že si zkonstrujeme jednoduché menu.

```
#!/usr/bin/env ruby
# $Id: menubar.rb,v 1.1 2003/11/03 18:22:20 radek Exp $
# $Source: /home/radek/cvs/ruby-book/example/gui/fxruby/menubar.rb,v $
require "fox"
include Fox

class MyApp < FXApp
end

class MyWin < FXMainWindow
    def initialize(app)
        super(app, "Menu", nil, nil, DECOR_ALL, 0, 0, 150, 36)

 # Vytvo<65533><65533>me menu
 menu = FXMenubar.new(self, LAYOUT_SIDE_TOP|LAYOUT_FILL_X)

 filemenu = FXMenuPane.new(self)

 FXMenuTitle.new(menu, "&File", nil, filemenu)
 FXMenuTitle.new(menu, "&Options")
 FXMenuTitle.new(menu, "&Help")

 FXMenuCommand.new(filemenu, "&Quit", nil, getApp(), FXApp::ID_QUIT)
    end

    def create
        super
        show(PLACEMENT_SCREEN)
```

```
        end
    end

    app = MyApp.new
    app.init(ARGV)
    win = MyWin.new(app)
    app.create
    app.run
```

po spuštění vypadá naše jednoduché menu takto

40.3.2.4.2. `Fox:FXMenuCommand`

Construct a menu command `new(parent, text, icon=nil, target=nil, selector=0, opts=0) {|theMenuCommand| ...}`

40.3.2.5. `Fox::FXFont`

* *section id="FXFont" xreflabel="FXFont"*

Práce s fonty.

```
    #!/usr/bin/env ruby
    # $Id: font1.rb,v 1.1 2005/10/04 08:52:07 radek Exp $
    # $Source: /home/radek/cvs/ruby-book/example/gui/fxruby/font1.rb,v $
    require "fox"
    include Fox

    class MyApp < FXApp
    end

    class MyWin < FXMainWindow
        def initialize(app)
            super(app, "Ahoj", nil, nil, DECOR_ALL, 0, 0, 188, 121)
            FXLabel.new(self, "<65533><65533><65533><65533><65533><65533><65533>")
            FXButton.new(self, "Konec", nil, app, FXApp::ID_QUIT)
        end

        def create
            super
            show(PLACEMENT_SCREEN)
        end
    end
```

```
if __FILE__ == $0
    MyApp.new do |app|
        app.normalFont = FXFont.new(app, "-*-helvetica-bold-r-normal-*-12-*-*-*-*-*-iso8859-2"
        app.init(ARGV)
        MyWin.new(app)
        app.create
        app.run
    end
end
```

Výsledek poté vypadá takto

40.3.2.5.1. Popis třídy `Fox::FXFont`

Tabulka 40-2. Font style hints with influence the matcher

konstanta	popis
FONTPITCH_DEFAULT	Default pitch
FONTPITCH_FIXED	Fixed pitch, mono-spaced
FONTPITCH_VARIABLE	Variable pitch, proportional spacing
...	...

Tabulka 40-3. Řezy fontů (*Font Slant*)

konstanta	popis
FONTSLANT_DONTCARE	Všechny řezy, na řezu nezáleží
FONTSLANT_REGULAR	Regulární přímý, bezpatkový
FONTSLANT_ITALIC	Kurzíva
FONTSLANT_OBLIQUE	FIXME: Oblique
FONTSLANT_REVERSE_ITALIC	obrácená kurzíva
FONTSLANT_REVERSE_OBLIQUE	FIXME: obrácený oblique

Tabulka 40-4. Kódování znaků (*Character Encoding*)

konstanta	popis
FONTENCODING_DEFAULT	výchozí kódování
FONTENCODING_ISO_8859_1	západoevropské kódování ISO-8859-1
FONTENCODING_ISO_8859_2	středoevropské kódování ISO-8859-2

konstanta	popis
FONTENCODING_ISO_8859_3	kódování ISO-8859-3
FONTENCODING_ISO_8859_4	kódování ISO-8859-4
FONTENCODING_ISO_8859_5	kódování ISO-8859-5
FONTENCODING_ISO_8859_6	kódování ISO-8859-6
FONTENCODING_ISO_8859_7	kódování ISO-8859-7
FONTENCODING_ISO_8859_8	kódování ISO-8859-8
FONTENCODING_ISO_8859_9	kódování ISO-8859-9
FONTENCODING_ISO_8859_10	kódování ISO-8859-10
FONTENCODING_ISO_8859_11	kódování ISO-8859-11
FONTENCODING_ISO_8859_12	kódování ISO-8859-12
FONTENCODING_ISO_8859_13	kódování ISO-8859-13
FONTENCODING_ISO_8859_14	kódování ISO-8859-14
FONTENCODING_ISO_8859_15	kódování ISO-8859-15
FONTENCODING_ISO_8859_16	kódování ISO-8859-16
FONTENCODING_KIO8	kódování KOI-8
FONTENCODING_KIO8_R	ruské kódování KOI-8
FONTENCODING_KIO8_U	ukrajinské kódování KOI-8
FONTENCODING_KIO8_UNIFIED	FIXME: KOI-8
FONTENCODING_LATIN1	západoevropské kódování ISO-8859-1
FONTENCODING_LATIN2	východoevropské kódování ISO-8859-2
FONTENCODING_LATIN3	jihoevropské kódování ISO-8859-3
FONTENCODING_LATIN4	kódování ISO-8859-4
FONTENCODING_LATIN5	kódování ISO-8859-9 (Turečtina)
FONTENCODING_LATIN6	kódování ISO-8859-10 (Nordic)
FONTENCODING_LATIN7	kódování ISO-8859-13 (Baltic Rim)
FONTENCODING_LATIN8	kódování ISO-8859-14 (Celtic)
FONTENCODING_LATIN9	kódování ISO-8859-15 (Latin 0)
FONTENCODING_LATIN10	kódování ISO-8859-16 (Latin 10)
FONTENCODING_USASCII	kódování ISO-8859-1 (Latin 1)
FONTENCODING_WESTEUROPE	kódování ISO-8859-1 (Latin 1)
FONTENCODING_EASTEUROPE	kódování ISO-8859-2 (Východní Evropa)
FONTENCODING_SOUTHEUROPE	kódování ISO-8859-3 (Jižní Evropa)
FONTENCODING_NORTHEUROPE	kódování ISO-8859-4 (Severní Evropa)
FONTENCODING_CYRILLIC	kódování ISO-8859-5 (Azbuka, Cyrilice)
FONTENCODING_RUSSIAN	kódování KOI-8 (Azbuka, Cyrilice)
FONTENCODING_ARABIC	kódování ISO-8859-6 (Arabština)
FONTENCODING_GREEK	kódování ISO-8859-7 (Řečtina)
FONTENCODING_HEBREW	kódování ISO-8859-8 (Hebrejština)
FONTENCODING_TURKISH	kódování ISO-8859-9, Latin 5 (Turečtina)
FONTENCODING_THAI	kódování ISO-8859-11, Thajsko
FONTENCODING_BALTIC	kódování ISO-8859-13, Pobaltské země
FONTENCODING_CELTIC	kódování ISO-8859-14, Latin 8 (Keltské písmo)

40.3.3. API, seznam tříd, metod, ...

Seznam tříd s krátkým popisem

modul `Fox`

Tento modul zapouzdřuje všechny metody, třídy a konstanty které FXRuby používá.

třída `Fox::FXApp`

Třída aplikace.

třída `Fox::FXBitmap`

Bitmapa - obrázek.

třída `Fox::FXButton`

Tlačítko s textem nebo obrázkem nebo obojím.

třída `Fox::FXCanvas`

Plocha na které mohou být objekty.

třída `Fox::FXComposite`

FIXME:

třída `Fox::Font`

Třída fontů.

třída `Fox::FXMatrix`

Layout Manager

třída `Fox::FXPacker`

Layout Manager

třída `Fox::___`

FIXME:

40.3.3.1. Třída `Fox::FXApp`

Objekt Aplikace

Objekt třídy `FXApp` neumí přijímat události ale může je posílat objektům.

Druhy událostí

Timers

Posílá cílovým objektům zprávy `SEL_TIMEOUT`.

Chores

Posílá cílovým objektům zprávy `SEL_CHORE`.

Inputs

Posílá cílovým objektům zprávy `SEL_IO_READ`, `SEL_IO_WRITE` a `SEL_IO_ECEPT`.

Signals

Posílá cílovým objektům zprávy `SEL_CHORE`.

addChore, `addInput`, `addSignal`, `addTimeout`, `beep`, `beginWaitCursor`, `closeDisplay`, `copyright`, `create`, `destroy`, `detach`, `disableThreads`, `dumpWidgets`, `enableThreads`, `endWaitCursor`, `exit`, `flush`, `forceRefresh`, `getDefaultCursor`, `getDragTypeName`, `init`, `instance`, `modal?`, `new`, `openDisplay`, `peekEvent`, `refresh`, `registerDragType`, `removeChore`, `removeInput`, `removeSignal`, `removeTimeout`, `repaint`, `run`, `runModal`, `runModalFor`, `runModalWhileShown`, `runOneEvent`, `runPopup`, `runUntil`, `runWhileEvents`, `setDefaultCursor`, `stop`, `stopModal`, `stopModal2`, `threadsEnabled?`

`addChore`

FIXME:

```
new(appName = "Application", vendorName = "FoxDefault");
```

Volání `new` vytváří nový objekt třídy `Fox::FXApp`

`init(`*`argv`*`, `*`connect`*`=true)`

FIXME:

`run()`

FIXME:

`runUntil(condition)`

FIXME:

`runWhileEvents(window=nil)`

FIXME:

40.4. Shoes

Shoes (http://shoooes.net/) je malý grafický toolkit umožňující jednoduše realizovat velmi jednoduché aplikace.

Odkazy:

- SHOES (http://shoooes.net/)
- THE SHOEBOX (http://the-shoebox.org/) A PLACE FOR LIL' APPS TO LIVE IN HARMONY.
-

40.5. JTTui

* *section id="jttui" xreflabel="JTTui" condition="author"*

40.5.1. Překlad a instalace

```
$ $HOME/source
$ tar xjf $HOME/arch/lang/ruby/jttui/jttui.200205082055.tar.bz2
```

40.5.2. Použití JTTui

Příklad 40-7. Příklad použití JTTui

```
#!/usr/bin/env ruby
# $Id: jttui_example1.rb,v 1.1 2002/06/05 15:48:59 radek Exp $
# Jakub Travnik's tui example

$:.unshift '/home/radek/lib/ruby'
require 'jttui/jttui'
require 'jttui/jttuistd'


JTTui.run do |root|
    d1 = JTTDialog.new(root, 'Dialog Window', 0,0,60,16,
         'Example 1 - simple dialog')
    d1.align = JTTWindow::ALIGN_CENTER
    wb1 = JTTWButton.new(d1, 'Test Button', 3,2,11,1, 'Button_1') { JTTui.beep }
    wb2 = JTTWButton.new(d1, 'Test Button', 3,3,11,1, 'Button_2') {
 JTTui.beep; sleep 1; JTTui.beep }
    wb3 = JTTWButton.new(d1, 'Test Button', 3,4,11,1, 'Button_3') {
 JTTui.beep; sleep 1; JTTui.beep; sleep 1; JTTui.beep }
    wbquit = JTTWButton.new(d1, 'Test Button', 48,13,8,1, '_Quit') {
 JTTui.addmessage nil, :close
    }

    el1 = JTTWEditline.new(d1, 'Test Editline1', 3,11,27,1,
      'Hello this is edit-line', true)
    el2 = JTTWEditline.new(d1, 'Test Editline1', 3,13,27,1,
      'This one is read only', false)

    la1 = JTTWLabel.new(d1, 'Test Label', 3,6,30,5,
    'Test _Label with text wrapping can' +
    ' select the widget bellow.  ' +
    'This edit line widget uses emacs like keys.') {
 d1.settab el1}

    ch1 = JTTWCheckbox.new(d1, 'Test Checkbox', 20,2,13,1, '_Checkbox1')
    ch2 = JTTWCheckbox.new(d1, 'Test Checkbox', 20,3,13,1, 'C_heckbox2') {
 JTTui.beep }
    ch2.states = 3

    rg1 = JTTWRadiogroup.new(d1, 'Test Radiogroup', 35,5,8,2,
        ['O_ne', 'T_wo'], -1)
    rg2 = JTTWRadiogroup.new(d1, 'Test Radiogroup', 46,5,13,4,
        ['_One', '_Two', 'Th_ree', '_Four'], 2)

    wb4 = JTTWButton.new(d1, 'Test Button', 32,11,17,1, 'Test _Messagebox') {
 m1 = JTTWMessagebox.new("Choose character for background\n" +
    "note: shapes are terminal dependent", 0,3,
```

```
        "_1: \1", "_2: \4", "_3: \7", "_4: \0",
        "_5: \11", "_6: none")

    m2 = JTTWMessagebox.new(”, 0,0, ’_I\’m happy’, ’_Try again’)
    begin
        res = m1.execute
        if res >= 0
     char = "\1\4\7\0\11 "[res]
     char = char | (res==5 ? JTTui.color_background : JTTui.color_basic)
     JTTui.rootwindow.background = char
        end
        m1.defaultnr = res # remember previous choose
    end until 0==m2.execute("You have pressed no. #{res}")

    JTTWMessagebox.new(’Thank you for trying message box widget behaviour. ’ +
         ’Notice that long messages should be in bigger window.’,
         0,0, ’_Ok’).execute
       }

       d1.addtabstop wb1
       d1.addtabstop wb2
       d1.addtabstop wb3
       d1.addtabstop ch1
       d1.addtabstop ch2
       d1.addtabstop rg1
       d1.addtabstop rg2
       d1.addtabstop el1
       d1.addtabstop el2
       d1.addtabstop wb4
       d1.addtabstop wbquit
       la1.down
       d1.cancelbutton = wbquit
       d1.defaultbutton = wb1
    end
```

40.6. Ostatní grafická prostředí

* *Zde jsou jen prázdné sekce jako připomínka.*

40.6.1. RuGUI

*

Odkazy:

- RuGUI – Ruby Graphical User Interface (http://rugui.org/)
- Github rugui / rugui (http://github.com/rugui/rugui)

40.6.2. GNUstep

Odkazy a zdroje:

- ruby-gnustep on FreePAN (http://freepan.org/ruby/by-cat/Library/GUI/ruby-gnustep/)

ToDo

1. První úkol.

40.6.3. Ruby TK

Odkazy a zdroje:

- Ruby TK (http://www.math.umd.edu/~dcarrera/ruby/0.3/chp_05/about.html)

ToDo

1. První úkol.

40.6.4. Ruby GNOME2

Odkazy a zdroje:

- Ruby-GNOME2 Project Website (http://ruby-gnome2.sourceforge.jp/)

ToDo

1. První úkol.

Bindings for Gtk+2 and Gnome2.

40.6.5. Ruby Cocoa

* *section id="cocoa" xreflabel="Ruby Cocoa" condition="author"*

Odkazy a zdroje:

- Ruby Cocoa - A Ruby/Objective-C Bridge for Mac OS X with Cocoa (http://www.imasy.or.jp/~hisa/mac/rubycocoa/index.en.html)

ToDo

1. První úkol.

RubyCocoa is a framework for Mac OS X that allows Cocoa programming in the object-oriented scripting language Ruby. RubyCocoa lets you write a Cocoa application in Ruby. It allows you to create and use a Cocoa object in a Ruby script. It’s possible to write a Cocoa application that mixes Ruby and Objective-C code.

Some useful applications of RubyCocoa:

- Exploration of a Cocoa object’s features with irb interactively
- Prototyping of a Cocoa application
- Writing a Cocoa application that combines good features of Ruby and Objective-C
- Wrapping Mac OS X’s native GUI for a Ruby script

40.6.6. Ruby FLTK

* *section id="fltk" xreflabel="Ruby/FLTK" condition="author"*

Odkazy a zdroje:

- Ruby/FLTK (http://ruby-fltk.sourceforge.net/) by Takaaki Tateishi

ToDo

1. První úkol.

Ruby/FLTK je knihovna která zprostředkovává přístup ke knihovně FLTK.

Knihovna FLTK (*Fast Light Tool Kit*) je krosplatformní grafická knihovna realizující grafické rozhraní na UNIXu/Linuxu (X11), Microsoft Windows a MacOS X. FLTK přináší funkcionalitu moderních GUI bez zátěže velkého množství kódu.

FLTK je navržena jako malá a modulární tak, že jednoduchý program `hello` staticky slinkovaný ma na architektuře Linux x86 velikost jen zhruba 97kB.

I přesto že se jedná o knihovnu malou, má v sobě takové prvky které nejsou ještě zcela běžné v ostatních grafických prostředích, jako jsou nekonečné seznamy a tabulky. Tyto prvky jsou realizovány v rozšíření FLTK jenž se jmenuje FLWM (*Fast Light Virtual Widgets*).

* *English, Author*

FLTK (pronounced "fulltick") is a cross-platform C++ GUI toolkit for UNIX?/Linux? (X11), Microsoft? Windows?, and MacOS? X. FLTK provides modern GUI functionality without the bloat and supports 3D graphics via OpenGL? and its built-in GLUT emulation. It is currently maintained by a small group of developers across the world with a central repository on SourceForge. FLTK is designed to be small and modular enough to be statically linked - the "hello" program is only 97k when compiled on an x86 Linux system! FLTK also works fine as a shared library and is now being included on Linux distributions.

Kapitola 41. Knihovny neuvedené jinde

Šablona pro nové kapitoly

* *Zde jsou spíše náměty na rozpracování do samostatných kapitol.*

V této kapitole jsou uvedeny knihovny, které jsem nepopsal v předchozích kapitolách.

41.1. Narf

NARF cgi je alternativa k dodávné knihovně cgi která podporuje HTTPUnit testy s a bez webserveru. Jako přídavek obsahuje knihovnu šablon a několik rozšíření api standardního cgi.rb.

Domovská stránka je na http://narf-lib.sourceforge.net

41.2. BigFloat

Zdroje a odkazy:

- http://www.tinyforest.gr.jp/ruby/bigfloat_en.html/

Rozšiřující knihovna pro interpreter Ruby. S použitím této knihovny můžete počítat s libovolnou přesností (s libovolným počtem desetinných míst).

41.3. Priority queue

Zdroje a odkazy:

- http://www.math.kobe-u.ac.jp/HOME/kodama/tips-ruby-pqueue.html

FIXME:

41.4. Nokogiri

*

Odkazy:

-
-

```
$ gem install nokogiri
Building native extensions.  This could take a while...
Successfully installed nokogiri-1.4.2
1 gem installed
Installing ri documentation for nokogiri-1.4.2...
Installing RDoc documentation for nokogiri-1.4.2...
```

```
#!/usr/bin/env ruby
```

```
require 'rubygems'
require 'nokogiri'
require 'open-uri'

url = "http://www.hnilica.cz/radek/book/ruby/index.html"
url = "http://www.walmart.com/search/search-ng.do?search_constraint=0&ic=48_0&search_query=arc

doc = Nokogiri::HTML(open(url))
puts doc.at_css("title").text

doc.css(".item").each do |item|
  title = item.at_css(".prodLink").text
  price = item.at_css(".PriceCompare .BodyS, .PriceXLBold").text[/\$[0-9\.]+/]
  puts "#{title} - #{price}"
  puts item.at_css(".prodLink")[:href]
end
```

V. Programování Webových aplikací

* *part id="programovani-webovych-aplikaci" xreflabel="Programování webových aplikací"*

V této části se budeme věnovat vývoji webových aplikací s pomocí Ruby a řad knihoven a aplikací.

Kapitola 42. eRuby

* *chapter id="eruby" xreflabel="eRuby"*

Odkazy a zdroje:

- mod_ruby the Apache/Ruby integration project (http://modruby.net/)

Začneme programem eRuby. Tento je na Apachi zcela nezávislou aplikací samostatně využitelnou například pro generování statických webových stránek.

> **Poznámka:** V distribuci Ruby je program Tiny eRuby (v souboru `erb.rb`) jenž je jednoduchou implementací eRuby.

eRuby je program, který umožňuje vložit Ruby kód do dokumentů. eRuby můžeme použít na čisté textové soubory (*plain text files*), XML a HTML soubory. eRuby je technologickým ekvivalentem JSP, ASP nebo PHP, který přináší všechny výhody jazyka Ruby.

42.1. Instalace a sprovoznění

1. eRuby získáme na webovských stránkách http://www.modruby.net/archive/. Stáhneme:

```
$ cd ~/arch/lang/ruby/eruby
$ wget http://www.modruby.net/archive/eruby-1.0.3.tar.gz
```

2. Rozbalíme:

```
>$ cd ~/tmpsrc
$ tar ~/arch/lang/ruby/eruby/eruby-1.0.3.tar.gz
```

3. Přeložíme a nainstalujeme:

```
$ cd eruby-1.0.3
$ ./configure.rb --enable-shared --with-charset=iso-8859-2
$ make
$ make install
```

Chceme-li používat eRuby z Apache, přidáme do části modulu `mod_ruby.c` konfiguračního souboru `/etc/apache/httpd.conf` tyto řádky:

```
## eRuby
RubyRequire apache/eruby-run
<FilesMatch ".rhtml">
        SetHandler ruby-object
        RubyHandler Apache::ERubyRun.instance
</FilesMatch>
```

42.2. Použití eruby

* *Použití eruby pro vytváření HTML stránek a jeho integrace do Apache*

eRuby, tedy embeded Ruby jak již název napovídá je Ruby zabudované do WWW stránek. Pro oddělění od ostatního html kódu jsou použity znaky `<%` a `%>`.

Blok ohraničený `<%#` a `%>` je ignorován.

Pokud potřebujeme uvést jen krátky příkaz, nemusíme jej uvádět jako blok ale stačí na začátek řádku napsat `%`

Použití eruby s apache. `eruby` doesn't automatically import the parameters. Use CGI library

```
<%
        require 'cgi'
        params = CGI.new.params
%>
```

CGI parametry jsou předávány jako slovník polí (*Hash of Arrays*)

```
<%= params['foo'][0] %>
```

Při použití multipart-requests (file upload) tak i parametry které nejsou soubory jsou objekty třídy Tempclass.

```
<%= params['foo'][0].read %>
```

Ukažme si tedy jednoduchou stránku:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<!-- $Id: hello.rhtml,v 1.1 2003/10/28 16:02:24 radek Exp $
     $Source: /home/radek/cvs/ruby-book/example/eruby/hello.rhtml,v $ -->
<% ERuby.charset = 'iso-8859-2' %>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
    <title>Hello</title>
  </head>
  <body>
    <h1>Hello</h1>
    <p><65533><65533><65533><65533><65533><65533><65533></p>
  </body>
</html>;
```

Stránka se seznamem method objektu `eRuby`:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<!-- $Id: eruby-methods.rhtml,v 1.1 2003/10/28 16:02:24 radek Exp $ -->
<% ERuby.charset = 'iso-8859-2' %>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
    <title>Metody modulu eRuby</title>
  </head>
  <body>
    <h1>Metody modulu <tt>eRuby</tt></h1>
    <p>Seznam metod: <%=ERuby.methods.sort.collect{|m| "<tt>#{m}</tt>"}.join ", "%>
  </body>
</html>;
```

Způsob přistupování k předávaným paramtrům.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<!-- $Id: eruby-params.rhtml,v 1.1 2003/10/28 16:02:24 radek Exp $ -->
<%
        require 'cgi'
        params = CGI.new.params
%>
<% ERuby.charset = 'iso-8859-2' %>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
    <title>P<65533>ed<65533>v<65533>n<65533> parametr<65533> do str<65533>nky</title>
  </head>
  <body>
    <h1>P<65533>ed<65533>v<65533>n<65533> parametr<65533> do str<65533>nky</h1>
    <table border="1">
      <tr><th>parametr</th><th>hodnota</th></tr>
<%
    params.each_key do |key|
        puts "<tr><td>#{key}</td><td>#{params[key]}</td></tr>"
    end
%>
    </table>
  </body>
</html>;
```

42.3. Mod Ruby a eRuby

Zdroje a odkazy:

- http://www.ruby-lang.org/raa/list.rhtml?name=mod_ruby
- http://www.modruby.net/

* *Podle „Session stub & a bug“, Szabolcs Szasz <sz@szasz.hu>*

Minimální `.rhtml` dokument pro sezení (session)

Příklad 42-1. Minimální *session* `.rhtml`

```
<%
require 'cgi/session'
session = CGI::Session.new( cgi = CGI.new('html4') )
begin
    session['var'] = 'xxx'

# You WILL use sessions from mod_ruby, so don't forget this
# for closing it (ie. closing the session-file, by default).
# See: http://www.modruby.net/doc/faq.en.html#label:14
ensure
    cgi.header      # *NASTY* hack to flush stuff to actually make the
                    # session live -- needed only for creating a session
                    # Any nicer way, please?? (I DO NOT want to
                    # clutter my "main stuff" (above) with cgi.calls, as
                    # I only need sessions, and conceptually, CGI has
                    # nothing to do with that.  I want a clean page ...)
```

```
    session.close
end
%>
```

mod_ruby je možno získat v archívu na http://www.modruby.net/archive/ a aktuální vývojovou verzi je možno získat z `cvs` následujícím postupem:

```
$ export CVSROOT=:pserver:anonymous@cvs.ruby-lang.org:/src
$ cvs login
Logging in to :pserver:anonymous@cvs.ruby-lang.org:2401/src
CVS password: anonymous
$ cvs checkout mod_ruby
```

42.4. Přehled objektů a metod

Metody a atributy objektu `ERuby`

atribut `noheader`

FIXME:

atribut `charset`

FIXME:

attribut `default_charset`

FIXME:

třída `Compiler`

FIXME:

Třída `ERuby::Compiler`

`sourcefile`
`sourcefile=`

FIXME:

`compile_string`

FIXME:

`compile_file`

FIXME:

42.5. Nezpracované podklady

```
<% require 'cgi' # Require the CGI library
cgi = CGI.new()  # New CGI object
```

```
username = cgi.param('username')
%><html><head><title>eRuby Test</title></head>
  <body>
    <h1>Testing eRuby</h1>
    <% if username.empty? # Print out the form asking for the username %>
      <form method="post">
        Please enter your username: <input type="text" name="username"><br>
        <input type="submit" value="Go">
      </form>
    <% else %>
      Welcome <%= username %>!
    <% end %>
  </body>
</html>
```

42.5.1. Posílání souboru

```
<% require 'cgi'; @cgi = CGI.new  %>

<% p @cgi.params['drop'].first %>
<form method='post' enctype='multipart/form-data'>
Drop Location: <input type=file name=drop>
<input type=submit name=submit>
</form>
```

Uploading was not as hard as I had thot! Attached is a file that should get you going uploading and included in text below....good luck !

Soubor `upload.rbx`

```
=begin

        upload.rbx is meant to be illustrative and might not
serve your purposes but should
        get you started if you need to upload with
Ruby/modruby

        {{{http://www.rubycentral.com/book/lib_network.html


        Multipart Form Values

        When dealing with a multipart form,
        the array returned by CGI#[] is composed of objects
of class Tempfile,
        with the following dynamically added methods:

                original_filename
                local_path
                content_type
                size

        To make this file work you'll need to...

                make it executable
```

```
                make your upload directory writable(Upload::PATH)

                ...and that _should_ be it

        }}}

=end


begin

        def main
        #{{{
                require 'cgi'

                $cgi = CGI.new("html4")

                $cgi.header('content-type'=>'text/html')

                puts '<html><body>'

                if $cgi.params.length == 0

                        Upload.form
                else

                        Upload.post

                        print Upload.getStatus

                end

                puts '</body></html>'

        end#}}}



rescue Exception=>e

        puts e

end

class Upload
#{{{ ----------------------Uploads file to server
from html form--------------------------

        #max size of file
        MAX_SIZE     = 25000

        #where the file goes / MUST HAVE WRITE PRIVS HERE
        PATH             = "/YOUR_FILE_PATH_HERE_PLEASE/"

        #how many file inputs - you can upload multiple files
at once
        FILE_COUNT       = 5
```

```
        #what file types do we allow?
        CONTENT_TYPES= ['image/jpg','image/gif']

        #how are things going?
        @@status         = []

        def self.form
        #{{{

                puts '<form method="post"
enctype="multipart/form-data">'

                FILE_COUNT.times do

                        puts '<br/><input type="file" name="myfile">'

                end

                 puts '<p/><input type="submit"></form>'

        end#}}}

        def self.post
        #{{{

                $cgi['myfile'].each do |incoming|

                        if incoming.size == 0

                                @@status << "<br/>Avoiding empty field"
                                next

                        end

                        if incoming.size > MAX_SIZE

                                @@status << "<br/>Data too large for
#{incoming.original_filename}(#{incoming.size} >
#{MAX_SIZE})"
                                next

                        end

                        #need to strip :)...trailing space...ouch
                        if not CONTENT_TYPES.include?
incoming.content_type.strip

                                @@status << "<br/>Type not allowed(type =
#{incoming.content_type}) allowed content =
#{CONTENT_TYPES.join(' | ')}"
                                next

                        end

                        # all should be ok to upload
                        path     = PATH + incoming.original_filename
```

```
                        File.new(path.untaint,'w') do |file|

                                file << incoming.read

                        end

                        @@status << "<br/>done writing #{path}"
                end

        end#}}}

        def self.getStatus
        #{{{

                @@status

        end#}}}
end#}}}

main
```

I've had to change the last upload.rbx file...hey i said i was a rookie...i've had to make these changes...

```
#File.new(path.untaint,'w') do |file|

        #file << incoming.read

#end

#to

file = File.new(path.untaint,'w')
file << incoming.read
file.close
```

apparently using the second is not allowing the write << from the read any idea why? The first worked but only created a null file. What am i missing? Was it not closing? Seems to work now but no promises ;)

42.5.2. Další poznámky k posílání souborů

```
% require 'cgi'
% @cgi = CGI.new
<html>
<body>
% if @cgi.params['drop'].first
filename:
<p><% print CGI.escapeHTML(@cgi.params['drop'].first.original_filename) %></p>
contents:
<pre><% print CGI.escapeHTML(@cgi.params['drop'].first.read) %></pre>
% end
<form action="test.rhtml" method='post' enctype='multipart/form-data'>
Drop Location: <input type="file" name="drop">
<input type=submit name=submit>
```

```
</form>
</body>
</html>
```

Kapitola 43. Camping

Odkazy:

- Camping (http://redhanded.hobix.com/bits/campingAMicroframework.html)
- Camping, a Microframework (http://code.whytheluckystiff.net/camping/)
-
-

Kapitola 44. Rack

* ***Attributy:*** *id="Rack"*

Odkazy:

- rack.rubyforge.org (http://rack.rubyforge.org)

Ukázky kódu:

- hello-thin.rb (example/hello-thin.rb)
- hello-rack.rb (example/hello-rack.rb)
- serve-file.rb (example/serve-file.rb)

Rack je konvence.

Máme li ruby object, který má metodu `call`

```
app.call(env)
```

a tato metoda vrací pole se třemi prvky

```
[200, {'Content-Type' => 'text/plain'}, 'Hello World!']
```

je možné ento objekt propojit s Rack

```
require 'thin'
Rack::Handler::Thin.run(app, :Port => 4000)
```

A tímto propojením vznikne wbová aplikace.

```
app = Proc.new do |env|
    [200, { 'Content-Type' => 'text/plain' },
     'Hello World!']
end

require 'rubygems'
require 'thin'
Rack::Handler::Thin.run(app, :Port => 4000)
```

Pole které metoda `call` vrací má tři prvky jenž mají tento váznam.

```
[200, {'Content-Type' => 'text/plain'}, "Hello #{@name}"]
```

- 200 — HTTP status kód
- {'Content-Type' => 'text/plain'} — Hash obsahující HTTP hlavičky které chceme posílat
- "Hello #{@name}" — tělo odpovědi, nemusí to být řetězec

```
[ 200,                              # HTTP status kód
  {'Content-Type' => 'text/plain'}, # Hash obsahující HTTP hlavičky které chceme posílat
  "Hello #{@name}"                  # tělo odpovědi, nemusí to být řetězec
]
```

Tělo odpovědi může být objekt který odpovídá na zprávu `:each` (`respond_to?(:each)`). Příklad použití:

```
file = File('myfile.xml')
[200, {'Content-Type' => 'application/xml'}, file]

class StreamingFile
```

```
    def initialize(file)
        @file = file
    end

    def length
        @File.size(@file)
    end

    def last_modified
        File.mtime(@file).tfc822
    end

    def each
        File.open(@file, "rb") do |file|
            while part = file.read(8192)
                yield part
            end
        File.delete(@file)
    end
end
```

Kapitola 45. Sinatra

Odkazy:

- Sinatra home (http://sinatrarb.com)
- Lightweight Web Services (http://rubyconf2008.confreaks.com/lightweight-web-services.html) by Adam Wiggins & Blake Mizerany
- MEET SINATRA (http://peepcode.com/products/sinatra) na Peep Code, cena $12

Sinatra is the easiest way to create a Fast, RESTful, web-application in Ruby with few dependancies, setup, and LOC.

Instalace s použitím RubyGems:

```
# gem install sinatra
```

Nyní si můžeme napsat první jednoduchý program:

```
#!/usr/bin/env ruby
require 'rubygems'
require 'sinatra'
get '/hi' do
  "Hello World!"
end;
```

Program spusítme a prohlížečem se podíváme na `http://localhost:4567/hi`.

```
$ ./hello.rb
== Sinatra/1.0 has taken the stage on 4567 for development with backup from Thin
>> Thin web server (v1.2.7 codename No Hup)
>> Maximum connections set to 1024
>> Listening on 0.0.0.0:4567, CTRL+C to stop
```

Na příkladu, na čtvrtém řádku, je vidět jak se mapuje kód na http dotazy. K tomuto mapování se používají příkazy `get`, `post`, `put` a `delete`. Mapovací metody mají jako parametr řetězec popisující část url. V tomto popisu můžeme použít "proměnné". Například v mapování

```
get '/hello/:name'
```

je `:name` symbol pojmenovávající parametr na daném místě. Http příkaz `GET /hello/Karel` je tedy zachycen mapováním `'/hello/:name'` a hodnotou `'Karel'` je naplněn parametr `params[:name]`.

```
get '/say/*/to/*' do
  params[:splat].inspect
end
# GET http://localhost:4567/say/hello/to/radek
# => ["hello", "radek"]
```

Parametrů, proměnných, v cestě může být více. Můžeme dokonce použít anonymní parametry. Takové parametry zastupuje znak '*'.

```
get '/say/*/to/*' do |pole|
  pole.inspect
end
# http://localhost:4567/say/hello/to/Radek
```

```
# => ["hello", "Radek"]
# warning: multiple values for a block parameter (2 for 1)
```

V této ukázce nám Sinatra vypíše varování. V cestě jsme definovali dva parametry, ale v bloku kódu přebíráme jen jeden. Sinatra se s tím vyrovná tak, že nám vrátí pole všech parametrů.

```
get '/say/:msg/to/:name' do |m,n|
  "m=#{m}, n=#{n}"
end
# http://localhost:4567/say/hello/to/Karel
# => m=hello, n=Karel
```

Mapování může provádat také regulárním výrazem

```
get %r{/hello/([\w]+)/(.*)} do
  "Hello, #{params.inspect}"
end
# http://localhost:4567/hello/Jana/Pavel
# => Hello, {"captures"=>["Jana", "Pavel"]}

get %r{/hello/([\w]+)/(.*)} do |f,m|
  "Pár: #{f} a #{m}"
end
# http://localhost:4567/hello/Jana/Pavel
# => Pár: Jana a Pavel
```

Při mapování můžeme mimo cestu zadat i podmínky. V těchto podmínkách můžeme testovat

- `:agent`
- `:host_name`
- `:provides`

Pomocné metody

45.1. git-wiki

Odkazy:

- git-wiki: because who needs cool names when you use git? (http://atonie.org/2008/02/git-wiki)
- git-wiki (http://github.com/schacon/git-wiki/tree/master) in github

Instalace git-wiki je jednoduchá, pokud víte jak na to. Vzhledem k tomu že se jedná o mladý projekt, určitě nebude v balíčku pro Debian a možná ani pro jiné distribuce. Dále je git-wiki závislý na novější verzi gitu. Budeme tedy muset instalovat git ze zdrojových kódů (../unix/git.html).

Kapitola 46. REST

* ***Attributy:*** *id="REST"*

Odkazy:

- Representational State Transfer (http://en.wikipedia.org/wiki/Representational_State_Transfer) na Wikipedii
-
-

Klientské knihovny:

- jnunemaker / httparty (https://github.com/jnunemaker/httparty) na GitHub
- maccman / nestful (https://github.com/maccman/nestful) na GitHub
- archiloque / rest-client (https://github.com/archiloque/rest-client) na GitHub
-

Operace

Tabulka 46-1. RESTful Web Service HTTP methods

	Element URI, http://example.com/resources/142	**Collection URI http://example.com/resources/**
GET	Retrieve: Prvek kolekce	List: Seznam prvků kolekce (URI)
PUT	Update or Create: aktualizace prvku kolekce nebo vytvoření	Replace: Výměna celé kolekce za jinou
POST	Create	Create
DELETE	Delete	Delete

Tabulka 46-2.

sloveso	**URI**	**použití**
POST	/resource	Vytvoření nového prvku kolekce.
GET	/resource/{id}	Získání prvku kolekce.
PUT	/resource/{id}	Aktualizace prvku kolekce novými hodnotami
DELETE	/resource/{id}	Odstranění prvku kolekce

CRUD:

- To create a resource on the server, use POST.
- To retrieve a resource, use GET.
- To change the state of a resource or to update it, use PUT.
- To remove or delete a resource, use DELETE.

```
GET /resources/?page=2 HTTP/1.1


PUT /users/Robert HTTP/1.1
Host: myserver
Content-Type: application/xml
```

```
<?xml version="1.0"?>
<user>
  <name>Bob</name>
</user>
```

46.1. POST

*

HTTP dotaz z klienta na server

```
POST /order HTTP/1.1
Host: example.com
Content-Type: application/xml
Content-Length: 239
<order xmlns="http://schemas.example.com/order">
  <location>takeAway</location>
  <items>
    <item>
      <name>latte</name>
      <quantity>1</quantity>
      <milk>whole</milk>
      <size>small</size>
    </item>
  </items>
</order>
```

Odpověd serveru klientovi

```
HTTP/1.1 201 Created
Content-Length: 267
Content-Type: application/xml
Date: Wed, 19 Nov 2008 21:45:03 GMT
Location: http://example.com/order/1234

<order xmlns="http://schemas.restbucks.com/order">
  <location>takeAway</location>
  <items>
    <item>
      <name>latte</name>
      <quantity>1</quantity>
      <milk>whole</milk>
      <size>small</size>
    </item>
  </items>
  <status>pending</status>
</order>
```

46.2. GET

*

Dotaz

```
GET /order/43221 HTTP/1.1
Host: example.com
```

Odpověď

```
HTTP/1.1 200 OK
Content-Length: 241
Content-Type: application/xml
Date: Sat, 20 Nov 2010 19:50:10 GMT

<order xmlns="http://schemas.example.com/order">
  <location>takeAway</location>
  <items>
    <item>
      <name>latte</name>
      <quantity>1</quantity>
      <milk>whole</milk>
      <size>small</size>
    </item>
  </items>
</order>
```

46.3. PUT

*

```
PUT /order/4321 HTTP/1.1
Host: example.com
Content-Type: application/xml
Content-Length: 246

<order xmlns="http://schemas.example.com/order">
  <location>takeAway</location>
  <items>
    <item>
      <milk>skim</milk>
      <name>cappuccino</name>
      <quantity>1</quantity>
      <size>large</size>
    </item>
  </items>
</order>
```

Odpověď

```
HTTP/1.1 200 OK
Content-Length: 275
Content-Type: application/xml
Date: Sun, 30 Nov 2008 21:47:34 GMT

<order xmlns="http://schemas.example.com/order">
  <location>takeAway</location>
  <items>
    <item>
      <milk>skim</milk>
```

```
      <name>cappuccino</name>
      <quantity>1</quantity>
      <size>large</size>
    </item>
  </items>
  <status>preparing</status>
</order>
```

46.4. DELETE

*

```
DELETE /order/1234 HTTP/1.1
Host: restbucks.com
```

Odpověď

```
HTTP/1.1 204 No Content
Date: Sat, 20 Nov 2010 19:56:04 GMT
```

46.5. Komunikace s REST serverem pomocí curl

*

Odkazy:

- REST-esting with cURL (http://blogs.plexibus.com/2009/01/15/rest-esting-with-curl/)
- How to Use cURL to Test RESTful Rails (http://railstips.org/blog/archives/2006/08/04/how-to-use-curl-to-test-restful-rails/)
- How to Use cURL to Test RESTful Rails (http://inquirylabs.com/blog2009/2006/08/04/how-to-use-curl-to-test-restful-rails/) by Duano Johnson
- Curl in to RESTful posters (http://blogs.openwonderland.org/2010/11/16/curl-in-to-restful-posters/)
-

POST

```
curl -i -H "Accept: application/json" -X POST -d "firstName=james" http
curl -X POST -H 'Content-type: text/xml' -d '<xml><login>john</login><password>123456</passwor
curl -i POST -u "user:password" -H "Content-Type: application/xml" -d "<Foo></Foo>" http://exa
```

PUT

```
curl -i -H "Accept: application/json" -X PUT -d "phone=1-800-999-9999" h
```

GET

```
curl -i -H "Accept: application/json" http://192.168.0.165/persons/per
```

DELETE

```
curl -i -H "Accept: application/json" -X DELETE http://192.168.0.165/per
```

Operace GET mi funguje

```
$ curl  http://cl-control.appspot.com/rest/apothecary
```

```
<?xml version="1.0" encoding="utf-8"?><list offset=""><apothecary><key>agpjbC1jb250cm9schELEgpBcG90aGVjY
```

Operace DELETE mi taky funguje

```
$ curl -X DELETE http://cl-control.appspot.com/rest/apothecary/agpjbC1jb250cm9schELEgpBcG90aGV
```

Vytvoření nového záznamu, podle postupu:

```
     * Gets the single <typeName> instance with the given <key>

# POST http://<service>/rest/<typeName>

     * Create new <typeName> instance using the posted data which should adhere to the XML Sche
     * Returns the key of the new instance by default. With "?type=full" at the end of the url,
```

Po chvilce zkoumání chyb jsem přišel s následujícím XML tvarem.

```
#!/bin/bash
URI="http://cl-control.appspot.com/rest/apothecary"

curl -X PUT $URI -d '<?xml version="1.0"?>
<apothecary>
  <stredisko>13</stredisko>
  <hostname>tester</hostname>
</apothecary>'
```

Kapitola 47. Ruby on Rails

Ruby na kolejích

* *chapter id="rails" xreflabel="Ruby on Rails"*
* *Vzhledem k rozsahu a významu této kapitoly by ji bylo asi nejlépe umístnit do samostatné části <part>.*

FIXME: Abstrakt kapitoly, je-li.

Odkazy:

- Ruby on Rails (http://www.rubyonrails.org/)
- Four Days on Rails (http://rails.homelinux.org/)
- Rolling with Ruby on Rails (http://www.onlamp.com/pub/a/onlamp/2005/01/20/rails.html)
- Seriál Ruby on Rails (http://www.root.cz/serialy/ruby-on-rails/) na serveru ROOT.CZ (http://www.root.cz)
    - Úvod (http://www.root.cz/clanky/ruby-on-rails-uvod/)
    - Jednoduchá aplikace (http://www.root.cz/clanky/ruby-on-rails-jednoducha-aplikace/)
    - Kniha hostů a blog (http://www.root.cz/clanky/ruby-on-rails-kniha-hostu-a-blog/)
    - Blog poprvé (http://www.root.cz/clanky/ruby-on-rails-blog-poprve/)
    - Blog podruhé (http://www.root.cz/clanky/ruby-on-rails-blog-podruhe/)
    - Dokončení blogu (http://www.root.cz/clanky/ruby-on-rails-dokonceni-blogu/)
    - Galerie poprvé (http://www.root.cz/clanky/ruby-on-rails-galerie-poprve/)
    - Testování (http://www.root.cz/clanky/ruby-on-rails-testovani/)
    - Renderování (http://www.root.cz/clanky/ruby-on-rails-renderovani/)
    - Co se nevešlo (http://www.root.cz/clanky/ruby-on-rails-co-se-neveslo/)

- Seriál Ruby on Rails na ZAACHI.COM
    - 1: Začínáme s Ruby (http://www.zaachi.com/cs/items/serial-ror-1-zaciname-s-ruby.html) [2010-11-15]
    - 2: První program (http://www.zaachi.com/cs/items/serial-ror-2-prvni-program.html) [2010-11-15]
    - 3: Základy syntaxe I (http://www.zaachi.com/cs/items/serial-ror-4-zaklady-syntaxe-i.html) [2010-11-15]
    - 4: Základy syntaxe II (http://www.zaachi.com/cs/items/serial-ror-4-zaklady-syntaxe-ii.html) [2010-11-15]
    - 5: Počítání s Ruby (http://www.zaachi.com/cs/items/serial-ruby-on-rails-5-pocitani-s-ruby.html) [2010-11-16]
    - 6: Datové typy: BigNum, FixNum, Float (http://www.zaachi.com/cs/items/serial-ruby-on-rails-6-datove-typy-bignum-fixnum-float.html) [2010-11-23]
    - 7: Datové typy: String (http://www.zaachi.com/cs/items/serial-ruby-on-rails-7-datove-typy-string.html) [2010-11-24]
    - 8: () [2010-12-]

- IRC síť FreeNet (FreeNode), kanál `#rubyonrails`
- Ruby on Rails forum (http://forum.rubyonrails.cz/)

Screencasts:

- First Rails 2.0 Screencast! (http://video.google.com/videoplay?docid=3210076950721253476&hl=en) na Video Google

Ruby on Rails je programový balík napsaný v Ruby jenž umožňuje rychlý návrh a vývoj webových aplikací. Tolik tedy v jedné větě, a co to znamená: Ruby on Rails (dále jen RoR) je prostředí pro vývoj a provozování webových aplikací, sestává z řady knihoven, modulů, skriptů, ... Vývoj v tomto prostředí je oproti starším nástrojům velmi urychlen. Programátor se může soustředit na samotnou logiku a řadu věcí za něj udělají knihovny a kód vygenerují skripty. V extrémním případě lze velmi jednoduché aplikace vytvářet co několik desítek minut jednu. Jak je to možné? RoR vychází z řady předpokladů, omezení, pravidel, která mu dovolují automaticky generovat kód aplikace. Například použitelná verze aplikace která slouží k editování dat v jedné datové tabulce se vytvoří prostým definováním struktury této tabulky a zavoláním generátor **script/scaffold**. Dokončení aplikace pak obnáší jen úpravy designu podle potřeby a estetického cítění.

Tento pohled na RoR je ovšem velmi zjednodušený. Samotné RoR a další nástroje jenž kolem RoR vznikly řeší a automatizují i další běžné programátorovy úlohy jako jsou:

- testování aplikace (extrémní programování)
- publikování vyvinutého kód z počítače vývojáře na cílový server (47.22.1)
- ladění aplikace, RoR má poměrně silné nástroje pro ladění aplikace

Dále něco nezapracovaných odkazů. Pokud potřebujeme vizualizovat modely či vztahy mezi některými objekty v RoR, podívejm se na projekty:

- Visualize Modesl (http://visualizemodels.rubyforge.org/) by Nils Franzen
- RailsRailroad (http://railroad.rubyforge.org/)

Ruby on Rails na YouTube od UCBerkeleyEvents

1. Hello World (http://www.youtube.com/watch?v=LADHwoN2LMM) 1:15:15 [2008-03-11]
2. Just Enough Ruby (http://www.youtube.com/watch?v=UCB57Npj9U0) 1:35:17 [2008-03-11]
3. Basic Rails (http://www.youtube.com/watch?v=LuuKDyUYFTU) 1:42:28 [2008-03-11]
4. Advanced Active Record (http://www.youtube.com/watch?v=FdeQmEY6phA) 50:29 [2008-03-11]
5. AJAX and Testing (http://www.youtube.com/watch?v=fsuuw5q4UjE) [2008-03-11]
6. Configuration and Deploy (http://www.youtube.com/watch?v=8WSf8FojTek) [2008-03-11]

47.1. Instalace Ruby on Rails

* *section id="rails.instalace" xreflabel="Instalce Ruby on Rails"*

Instalace Ruby on Rails s použitím gemu je jednoduchá, prostě jej pomocí příkazu **gem** nainstalujeme.

```
# gem install rails
```

Uvedený příkaz nám může v praktickém životě posloužit jako vtip, protože nainstaluje jen a pouze samotný Ruby on Rails. To je však pro naše použití málo. Potřebujeme nainstalovat taktéž další software. Nejde jen o závislosti na jiných balíčcích ty vyřešíme přidáním parametru `--include-dependencies` nebo `-y`. Jde například o knihovny zajišťující přístup k databázovým strojům, web server publikující naši aplikaci na webu. Ale všechno pěkně po řadě, nechceme přece nic uspěchat. Začenem tedy prostou instalací Ruby on Rails.

```
# gem install rails --include-dependencies
```

Tímto nainstalujeme nejnovější verzi, která je k dispozici. Pokud máme nějaký důvod instalovat verzi starší, můžeme s úspěchem použít parametr `--version` *n*, kde *n* je číslo požadované verze. Verzi 0.12.1 nainstalujeme tedy takto:

```
# gem install rails --version 0.12.1
```

Poznámka: Instalaci starší verze použijeme s výhodou tehdy, máme li tutoriál, knihu či jiný výukový materiál jenž tuto starší verzi používá. Například Rolling with Ruby on Rails (http://www.onlamp.com/pub/a/onlamp/2005/01/20/rails.html), kde je použita verze 0.9.4. Rozdíly mezi nainstalovanou verzí a verzí jenž je v manuálu popisována nás mohou mást.

Poznámka: Můžou se nám hodit další balíčky v našem systému, například `libopenssl-ruby`.

47.1.1. Instalace pomocí RubyGems

Odkazy a materiály:
- RubyGems
-
-

Jedním ze způsobů instalace je instalace pomocí RubyGems.

47.1.2. Instalace Rails 3.0 Beta s pomocí RubyGems

Odkazy:
- Rails 3.0: Beta release (http://weblog.rubyonrails.org/2010/2/5/rails-3-0-beta-release/)
- Rails 3.0: Release Notes (http://guides.rails.info/3_0_release_notes.html)
- How I Use Bundler (http://blog.admoolabs.com/how-i-use-bundler/)

Rails 3.0 vyžaduje Ruby verze 1.8.7 nebo vyšší. Je kompatibilní i s verzí Ruby 1.9.2.

```
$ gem install tzinfo builder memcache-client rack rack-test rack-mount erubis mail text-format
$ gem install rails --pre
```

47.1.3. Instalace Rails 3.0 Beta s pomocí bundleru

Odkazy:
- How I Use Bundler (http://blog.admoolabs.com/how-i-use-bundler/)
-

Rails 3.0 vyžaduje Ruby verze 1.8.7 nebo vyšší. Je kompatibilní i s verzí Ruby 1.9.2.

Při instalaci pomocí bundleru nám tento udržuje oddělenou sadu gemů. Zastane nám tedy podobnou funkci jak instalace gem do uživatelského prostoru.

```
$ gem install bundler
```

Nainstaloval se mi `bundler-0.9.20`. Nyní si vytvoříme adresář pro novu sadu gemů. A v něm soubor Gemfile.

```
$ mkdir -P $HOME/lib/rails3b
$ cd $HOME/lib/rails3b
$ vi Gemfile
```

```
source "http://gemcutter.org"
gem "rails", "3.0.0.beta"
```

A nyní stáhneme gem s rails 3 beta a všemi dalšími na kterých závisí.

```
$ bundle install r3b
```

Před použitím příkazu bundle mu musíme nastavit cestu k souboru Gemfile.

```
$ export BUNDLE_GEMFILE=$HOME/lib/rails3b/Gemfile
```

47.1.4. Poznámky k instalaci na Debian 5.0 Lenny

Odkazy:

- Position on RubyGems (http://pkg-ruby-extras.alioth.debian.org/rubygems.html)
- ruby, gem, rails, debian lenny, sqlite3, mysql, Days on Rails (http://maximilianou.blogspot.com/2008/10/ruby-gem-rails-debian-lenny-first-day.html)
- Installing Ruby on Rails on Debian/Ubuntu (http://wiki.rubyonrails.org/getting-started/installation/linux-ubuntu)
- Ruby on Rails on Debian (http://www.debian-administration.org/articles/329)

Nejprve musíme mít nainstalováno ruby.

```
# aptitude install ruby-full irb
```

RubyGems mají s Debianem problém. Jedním ze způsobů jak jej vyřešit je nainstalovat RubyGems v uživatelském prostoru.

* ***WORKING: Editovat.***

```
# aptitude install rubygems
```

```
# aptitude install rails
```

47.2. Vytvoření kostry aplikace

* *chapter*

RoR má pro mnoho případů generátory které za nás vytvářejí adresáře a soubry. Právě použití těchto generátorů je jednou z důležitých věcí. První generátor který si ukážeme je samotný **rails**. Tento nám vygeneruje celou šablonu aplikace. Příkaz **rails** akceptuje jako parametr adresář ve kterém vytvoří kostru aplikace. Následující příkaz vytvoří v aktuálním adresáři adresář `admin` a v něm všechny další podadresáře a soubory aplikace.

```
$ rails admin
```

Vytvořenou kostru aplikace si můžeme ihed vyzkoušet. Učiníme tak jednoduše spuštěním web serveru v adresáři aplikace s použitím jednoho z vygenerovaných (nakopírovaných) skriptů **script/server**.

```
radek@yoda:~/src/firma/mpress/admin: 0 $ script/server
=> Rails application started on http://0.0.0.0:3000
=> Ctrl-C to shutdown server; call with --help for options
```

⋮

Pokud nám server nenastartuje, a ve výpisu najdeme něco jako

```
=> Booting WEBrick...
=> Rails 2.1.0 application started on http://127.0.0.1:3000
=> Ctrl-C to shutdown server; call with --help for options
[2009-11-12 11:32:47] INFO  WEBrick 1.3.1
[2009-11-12 11:32:47] INFO  ruby 1.8.7 (2008-08-11) [x86_64-linux]
[2009-11-12 11:32:47] WARN  TCPServer Error: Address already in use - bind(2)
/usr/lib/ruby/1.8/webrick/utils.rb:73:in ‘initialize’: Address already in use - bind(2) (Errno::EADDRINU
```

Znamená to, že stadardní port 3000 je obsazen jinou aplikací. V takovém případě jednoduše spustíme server na jiném portu. Například na portu 3333.

```
$ script/server -p 3333
```

Jak vidíme server se úspěšně nastartoval a očekává dotazy na portu 3000 (nebo portu 3333 :). Zadáme tedy do prohlížeče adresu `http://localhost:3000/` a uvidíme standardní předvytvořenou stránku.

Druhá ukázka využívá Subversion

```
radek@yoda:~: 0 $ rails ~/src/firma/mpress/snimkypd -c
```

47.3. Databázové stroje

* *section id="rails.database-engines"*

Naše aplikace a tedy i Ruby on Rails potřebuje přístup k datům. Tato data jsou často důvod proč aplikaci píšeme. Míváme je uložena v nějaké databázi, nejčastěji SQL databázi. Těch je celá řada a Ruby on Rails si z řadou z nich rozumí. Pokud ne, máme možnost dopsat vlastní databázový ovladač. Jak to udělat se dočteme například v New database adapter (http://wiki.rubyonrails.com/rails/pages/New+database+adapter).

Budu se zde zabývat pouze databázemi se kterými mám zkušenost, což není mnoho. Napřed alespoň zmíním které databáze lze z Ruby on Rails použít. Jsou to:

- MySQL
- 47.3.1
- SQLite 2
- 47.3.2

Přístup k databázím je popsán v konfiguračním souboru `config/database.yml`. Zde je pro každé 47.5.1 zvlášť definován použitý databázový konektor s parametry databáze.

47.3.1. PostgreSQL

* *section id="rails.postgresql"*

Odkazy:

- Installing Ruby on Rails and PostgreSQL on OS X, Second Edition (http://www.robbyonrails.com/articles/2007/06/19/installing-ruby-on-rails-and-postgresql-on-os-x-second-edition) by Robby Russell

- PostgreSQL (http://wiki.rubyonrails.org/rails/pages/PostgreSQL)

Potřebujeme samotný server PostgreSQL verze **FIXME:**7.2 nebo vyšší. Tento se nachází v balíčku `postgresql`. Já používám SQL server na jiném stroji vyjma vývojářského notebooku kde si ho nosím sebou. Pro připojení *Ruby* budeme potřebovat balíček `libpgsql-ruby1.8`, a může se nám hodit i CLI klient `postgresql-client`. Ten použujeme při ručních změnách v databázi, zakládání databází i tabulek. Všechny uvedené balíčky jsem instaloval z Debian Sarge.

Na Debian Etch se tento balíček rovněž jmenuje libpgsql-ruby

```
# aptitude install libpgsql-ruby
```

* *Ověřit číslo verze PostgreSQL. Etch (7.4, 8.1) Leny (8.3)*

Konfigurační záznam databázového konektoru pro PostgreSQL vypadá typicky takto

```
development:
    adapter: postgresql
    database: mojedb
    encoding: UTF8
    username: karel
    password: heslo321
    host: localhost
```

Tento záznam popisuje připojení k databázi `mojedb` jenž běží lokálně na tomoto stroji (`localhost`). Databáze je v kódování `UTF8` a prihlašujeme se jménem `karel` a heslem `heslo321`.

Pokud máme k databázi povolen přístup bez kontroly hesla, nemusíme položku `password` vyplňovat. Pokud používáme identifikační metodu `ident`, nemusíme vyplňovat ani pole `username`. Databáze si naše jméno zjistí právě pomocí programu `ident`.

V uvedeném příkladu není použita položka `port` jenž určuje číslo TCP portu na kterém běží server. Implicitní hodnota je 5432.

Přehled voleb / parametrů připojení k PostgreSQL serveru:

`encoding`

Kódování užité pro kódování národních (ne ASCII) znaků v databázi.

`host`

Adresa databázového serveru ke kterému se naše aplikace připojuje.

`port`

Číslo tcp portu na kterém běží server a na který se naše aplikace připojuje.

`schema_search_path`

Tímto parametrem specifikujeme schema v databázi, pakliže schémata používáme.

`min_messages`

* *aptitude install ruby1.8-dev libpq-dev; gem install postgres-pr; aptitude search libpgsql-ruby1.8*

47.3.1.1. Vytvoření databáze

Uvedu praktický postup vytvoření databáze a konfigurace připojení.

V ukázkách budu používat pokusnou databázi rorex (rorexdev/rorextest). Vlastníkem těchto databází je uživatel 'roxana'. Zde uvádím ve zkratce postup vytvoření této databáze.

Jako uživatel root se přehlásím na vlastníka databázového stroje postgres a vytvořím uživatele i databáze. V příkazu je uvedený parametr `--cluster` kterým specifikujeme ke kterému databázovém clusteru na lokálním počítači se připojujeme.

```
# sudo -u postgres psql [--cluster 8.1/main] -d template1
template1=# CREATE USER roxana WITH ENCRYPTED PASSWORD 'cokolada' NOCREATEDB NOCREATEUSER;
template1=# CREATE DATABASE rorex WITH OWNER=roxana TEMPLATE=template0 ENCODING='utf-8';
template1=# CREATE DATABASE rorexdev WITH OWNER=roxana TEMPLATE=template0 ENCODING='utf-8';
template1=# CREATE DATABASE rorextest WITH OWNER=roxana TEMPLATE=template0 ENCODING='utf-8';
template1=# \q
```

Do souboru `/etc/postgresql/8.1/main/pg_hba.conf` databázového serveru dopíši řádky povolující uživateli roxana přístup k právě vytvořeným databázím z tohoto (lokálního) stroje.

```
# vi /etc/postgresql/8.1/main/pg_hba.conf

# Přístup k RoR databázi rorex(dev/test).
local rorex roxana md5
local rorexdev roxana md5
local rorextest roxana md5
```

Po úpravě `pg_hba.conf` je třeba oznámit postgresu změnu konfigurace:

```
# /etc/init.d/postgresql-8.1 reload

Nyní si ověříme funkčnost tím že se k databázi přihlásíme

$ psql --cluster 8.1/main -U roxana -W -d rorex
```

Po zadání správného hesla, v našem případě `cokolada` se dostaneme do databáze

```
Welcome to psql 7.4.19, the PostgreSQL interactive terminal.
⋮
rorex=>
```

47.3.2. SQLite 3

* ***Attributy:*** *id="rails.sqlite3"*

SQLite je jednoduchý, velmi odlehčený databázový stroj. Ukládá celou databázi do jednoho souboru.

```
# aptitude install ruby-dev sqlite3 libsqlite3-dev swig
# gem install sqlite3-ruby
```

47.4. Web server

* *section id="rails.webserver" xreflabel="Web server"*

Další věcí kterou budeme potřebovat je web server, který bude naši aplikaci prezentovat. Pro ladění můžeme použít WEBrick, který je standardní součástí ruby. Tot byla nejjednodušší instalace, nemusíme instalovat nic.

* *Aplikaci kterou jsem napsali, nebo píšeme potřebujeme zkoušet a provozovat. V případě zkoušení je to velmi jednoduché. Nejlépe použít webový server WEBrick. V případě vystavení aplikace k reálnému užití pak volíme obvykle jiný server. Ale nemusí to tak být. Záleží na naší úvaze, charakteru aplikace a předpokládaného používání aplikace. Rozeberme si tedy všechny možnosti které máme.*

47.4.1. WEBrick

* *section id="rails.webrick" xreflabel="WEBrick"*

Odkazy:

- WEBrick (http://www.webrick.org/)

FIXME: od verze .... je již WEBrick přímo součástí Ruby, takže jej nemusíme instalovat zvlášť.

Jak jsem se již zmínil, použití Webricku je nejjednodušší. Mezi skripty jenž byly při založení aplikace vytvořeny v adresáři `script` je jeden `script/server`, který slouží ke spuštění www serveru WEBrick s naší Rail aplikací. Použijeme-li jej bez parametrů, připojí se server na port 3000 a rail aplikaci spustí v **FIXME:**módu/režimu `development`. Tento skript je užíván hodně při ladění a vývoji aplikace, kdy se nemusíme zabývat konfigurací a nastavováním www serveru ale pohotově spustíme WEBrick bez jakéhokoliv dalšího nastavování. V adresáři aplikace zadáme příkaz

```
$ script/server
```

a spustí se server s aplikací. Server očekává dotazy na portu 3000 lokálního počítače `localhost` (ip=127.0.0.1). Pokud je port obsazen, či zjiného důvodu jej nemůžeme/nechceme použít, má startovací skript `script/server` řadu parametrů jimiž můžeme ovlivnit jeho chování. Můžeme zadat

`-p` *`číslo_portu`*
`--port=`*`číslo_portu`*

pro určení TCP portu na kterém má server naslouchat

`-b` *`ip_adresa`*
`--bind=`*`ip_adresa`*

pro navázání serveru na konkrétní ip adresu některého síťového rozhraní lokálního počítače. Standardně se server navazuje na adresu `0.0.0.0` což znamená na všechny adresy všech rozhraní počítače které existují v době startu aplikace.

`-e` *`prostředí`*
`--environment=`*`prostředí`*

parametr určuje které ze základních předdefinovaných prostředí se má použít. Jména použitých prostředí jsou sama o sobě dostatečně popisná, jedná se o hodnoty: `test`, `development`, `production`. Pokud tímto parametrem neurčíme jinak, nebo nenastavíme prostředí v konfiguraci, aplikace se standardně spouští v prostředí `development`.

```
$ script/server -e production
```

`-d`
`--daemon`

posledí volbu kterou popíši je volba `--daemon`. Server se spustí v režimu démona a odpojí se tedy od terminálu ze kterého byl spuštěn. To mimo jiné znamená že všechny výpisy jenž normálně server psal na terminál budou se již na terminálu neobjeví a jediné místo kde je můžeme hledat je deník serveru.

```
$ script/server --daemon
=> Rails application started on http://0.0.0.0:3000
[2005-07-24 16:42:27] INFO  WEBrick 1.3.1
[2005-07-24 16:42:27] INFO  ruby 1.8.2 (2005-04-11) [i386-linux]
$
```

47.4.2. Mongrel

```
# gem install mongrel
```

47.4.3. Passenger

Odkazy:

- Passenger (mod_rails for Apache) (http://www.modrails.com/)
- Configurating Passenger (mod_rails) on SliceHost with Ubuntu 7.10 (http://www.railsgarden.com/2008/04/12/configurating-passenger-mod_rails-on-slicehost-with-ubuntu-710/)

Instalace je velmi jednoduchá, nejdříve nainstalujeme gem

```
# gem install passenger
```

A poté spustíme instalaci

```
# passenger-install-apache2-module
```

Instalace je velmi chytrá a upozorní nás na programy a balíčky které jí chybí a rovněž nám řekne jakým způsobem je nainstalujeme. Stačí se tedy jen řídit jejími pokyny.

V mém případě jsem musel doinstalovat balíček `apache2-prefork-dev` a vyměnit původně použitý `apache2-mpm-worker` za `apache2-mpm-prefork`.

```
# aptitude install apache2-mpm-prefork apache2-prefork-dev
```

Poté doplníme konfiguraci apache. S výhodou jsem použil adresář `/etc/apache2/conf.d` do kterého jsem vložil soubor `passenger` s následujícím obsahem:

```
LoadModule passenger_module /usr/lib/ruby/gems/1.8/gems/passenger-1.0.5/ext/apache2/mod_passer
RailsSpawnServer /usr/lib/ruby/gems/1.8/gems/passenger-1.0.5/bin/passenger-spawn-server
RailsRuby /usr/bin/ruby1.8
```

Konfigurace webů pro jednotlivé aplikace

```
<VirtualHost *:80>
        ServerName www.yourhost.com
        DocumentRoot /somewhere/public
```

```
</VirtualHost>
```

47.4.4. Apache

* *section id="rails.apache" xreflabel="Apache"*

Jedním z nejvíce používaných webových serverů je Apache (http://apache.org/). Často jej používáme pro poskytování jak statického obsahu i aplikací. Je velmi pravděpodobné že je nasazen i na serveru kde chceme vystavit naši aplikaci.

O integrování Rails aplikací do Apache (http://apache.org/) je tato část. V principu jsou možné tři způsoby integrování aplikace do Apache (http://apache.org/)

- CGI skripty
- užití modulu `mod_ruby`
- FastCGI skripty

Další možností kterou máme je použít jako web server Apache. Tento je k dispozice na většině dostupných platforem a nezřídka již běží na našem serveru. Velmi často slouží již pro vystavování statických stránek a dalších aplikací. My do něj pouze začleníme tu naši. Vzájemné propojení a komunikace naší aplikace s www serverem Apache je možné jedním ze tří způsobů které si postupně ukážeme. První způsob kterým se budeme zabývat je aplikace jako CGI skript.

FIXME:

47.4.5. Apache CGI

FIXME:

Začleníme tedy do konfigurace apache (http://httpd.apache.org/) následující řádky. Uvedeme je přímo v konfiguračním souboru `httpd.conf` nebo v smaostatném souboru podle verze apache (http://httpd.apache.org/) a podle způsobu konfigurace.

```
<Directory /var/www/rails_aplikace/>
        Options ExecCGI FollowSymLinks
        AddHandler cgi-script .cgi
        SetEnv RAILS_ENV production
        AllowOverride all
        Order allow,deny
        Allow from all
</Directory>
```

První řádek `Directory` definuje adresář ve kterém se nachází naše aplikace z pohledu apache (http://httpd.apache.org/). Další řádek specifikuje že se mají akceptovat a vykonávat CGI skripty (`ExecCGI`), a že se mají používat a následovat symbolické odkazy (`FollowSymLinks`). Poté nastavíme ovladač cgi skriptů jenž bude rozeznávat jako skripty soubory s příponou `.cgi`. Důležitý řádek s direktivou `SetEnv` nastaví proměnnou prostředí `RAILS_ENV` jejíž hodnota určuje v kterém ze tří základních režimů/módů se aplikace spustí (`development`/`test`/`production`). V našem případe to bude produkční ražim (`production`). **FIXME:**

Použití CGI je jednoduché ale zároveň výkonově nejslabší. Celá aplikace jako CGI skript se při otveření každé nové stránky opět celá spouští a načítá do paměti. Toto velmi zatěžuje systémové zdroje. Pokud chcete tuto variantu použít, odzkoušejete si ji na vaší kokrétní konfiguraci zdali vám bude vyhovovat. Když ne, můžete použít některý z dále zmíněných způsobů.

47.4.6. Apache a `mod_ruby`

Tato varianta je postavena na využití modulu mod_ruby apache (http://httpd.apache.org/). Využíva toho, že interpret Ruby se stává součástí apache a nemusí se s každou prohlíženou stránkou znovu startovat. Je tedy z hlediska koncového uživatele rychlejší. Použití mod_ruby je má oblíbená varianta na paměťově slabších serverech. Není sice tak rychlá jako varianta s FastCGI, ale je znatelně rychlejší a svižnější než varianta s CGI. Nemá rovněž paměťové nároky FastCGI. Má-li váš server k dispozici málo paměti pro vaši aplikaci, vyzkoušejte mod_ruby, může to být ta správná volba pro vás.

Podle verze apache, kterou používáme, přinstalujeme bliček `libapache2-mod-ruby` nebo `libapache-mod-ruby`.

```
# aptitude install libapache2-mod-ruby
```

```
# aptitude install libapache-mod-ruby
```

Konfigurace aplikace je sdružena do jednoho souboru, na který se z hlavní konfigurace apache (http://httpd.apache.org/) odkážeme direktivou `Include`, nebo ji zahrneme přímo do souboru `httpd.conf`.

```
#
# Apache configuration for WebDB aplication using mod_ruby
# Copyright (c) 2005 Radek Hnilica
# All rights reserved.  Všechna práva vyhrazena.

# Following global configuration rule can be removed if it's one in
# main httpd.conf.  The RubySafeLevel should be 0 (Unsafe), because
# Rails does not work with higher value (more safe level) yet.

RubySafeLevel 0

<Directory /var/www/název-aplikace>

 Options ExecCGI FollowSymlinks
 AllowOverride None

 RubySafeLevel 0
 RubySetEnv RAILS_ENV production
 RubyRequire apache/ruby-run
 <Files *.rb>
  SetHandler ruby-object
  RubyHandler Apache::RubyRun.instance
 </Files>

 # Restricting access to application only to few users.  Users
 # should authenticate using theirs account names and passwords.

 #AuthType Basic
 #AuthName "Prokažte prosím vlastníctví účtu znalostí hesla."
 #AuthUserFile /etc/apache-ssl/passwd/user
 #Require user radek twada dusan

 # Restricting access by ip or domain names.  No restriction
 # taken.

 Order allow,deny
 Allow from all

 # Configuration of rewrite engine.
```

```
    RewriteEngine On
    RewriteRule ^$ index.html [QSA]
    RewriteRule ^([^.]+)$ $1.html [QSA]
    RewriteCond %{REQUEST_FILENAME} !-f
    RewriteRule ^(.*)$ dispatch.rb [QSA,L]

    # Specific error message for error 500.

    ErrorDocument 500 "<h2>Application fatal error</h2>Rails application\
    failed to start properly"
   </Directory>
```

FIXME:

47.4.7. Apache FastCGI

FIXME:dopsat.

47.4.8. Poznámky ke konfiguraci Apache

V předchozích případech jsme probírali konfiguraci pro případ že adresář z celou aplikací je uložen v www prostoru Apache, tedy obvykle v `/var/www`. Nyní se pokusím popsat konfiguraci kdy tomu tak není.

```
   # /etc/apache2/conf.d/rails-app
   Alias /rails-app /usr/local/share/rails-app

   <Directory /usr/local/share/rails-app>
   </Directory>

http://server/rail-app/public/
```

47.4.9. Apache 2

Odkazy:

- Using Ruby On Rails With Apache2 On Debian Etch (http://www.howtoforge.com/ruby_on_rails_debian_etch)
-

Queue:

- Ruby on Rails: dispatch.fcgi was not found on this server (http://www.eukhost.com/forums/showthread.php?t=2084)
- http://lists.unbit.it/pipermail/ml/2006-July/001460.html

```
NameVirtualHost *
<VirtualHost *:*>
    ServerAdmin giulio a troccoli.it
    DocumentRoot /home/webmaster/troccoli.it/spagnolo/public
    ServerName spagnolo.troccoli.it
    ErrorLog logs/spagnolo-error_log
    CustomLog logs/spagnolo-access_log common
```

```
        <Location /journal>
            RewriteEngine On
            # Let Apache handle purely static files like images by itself.
            RewriteCond %{REQUEST_FILENAME} !-f
            # Send Everything else to Rails
            RewriteRule ^(.*)$ dispatch.fcgi [QSA,L]
        </Location>
        <Directory /home/webmaster/troccoli.it/spagnolo/journal>
            # ExecCGI is required for mod_fcgid to work.
            Options Indexes FollowSymLinks ExecCGI
            # Disable .htaccess files.
            AllowOverride None
            Order allow,deny
            Allow from all
            # This tells mod_fcgid to run the dispatch.fcgi script as a
    FastCGI
            AddHandler fcgid-script .fcgi
        </Directory>
    </VirtualHost>
```

- [ruby-it] Re: Re: Come usare Rails con Apache? (http://lists.unbit.it/pipermail/ml/2007-February/004281.html)
- Ruby on Rails Not Found error with apache2 (http://www.mail-archive.com/debian-user@lists.debian.org/msg465539.html)
- Apache fCGI Server unter Ubuntu (http://forum.ruby-portal.de/ptopic,37187.html)
- HowtoSetupApacheWithFastCGIAndRubyBinding (http://wiki.rubyonrails.org/rails/pages/HowtoSetupApacheWithFastCG
- Getting spurious „no route found to match“ with Rails 1.2.3? (http://talideon.com/weblog/2007/07/rails1-2-3-bad-routes.cfm)
- Apache tuning for Rails and FastCGI (http://scottstuff.net/blog/articles/2005/07/20/apache-tuning-for-rails-and-fastcgi)
- So for each rails app I am running on the server I need to call that FastCGI line? like:

  ```
  FastCgiServer /rails_app_1/dispatch.fcgi -idle-timeout 120 -initial-env RAILS_ENV=produc
  FastCgiServer /rails_app_2/dispatch.fcgi -idle-timeout 120 -initial-env RAILS_ENV=produc
  ```

- http://www.cheminsdetraverse.info/index.php?2006/12/05/20-apache2-rails-en-virtualhost
- Install Ruby on Rail, Apache 2.2.3, Fastcgi 2.4.2 on Linux (http://hack.emilykwan.com/node/95)

* ***FIXME:****Popovídat o Apache 2 trochu obecně.*

```
# aptitude install ruby1.8-dev
# aptitude install apache2 libsasl2-modules
# a2enmod rewrite
# a2enmod ruby
# a2enmod ssl
```

Debian Etch

```
# aptitude install ruby libzlib-ruby rdoc irb rubygems eruby [rails]
# aptitude install libapache2-mod-ruby
```

47.4.9.1. fcgid

* *section id="rails.apache2.fcgid"*
* *RailsOnDebian (http://old.bougyman.com/RailsonDebian.html)*

Fcgid je náhrada, respektive následník mod-fcgi.

Nejdříve tedy instalace a „konfigurace“ fcgid. Nainstalujeme balíčky libapache2-mod-fcgid a libfcgi-ruby.

```
# aptitude install apache2 libapache2-mod-fcgid libfcgi-ruby
```

Pokud nemáme povolené, povolíme následující moduly v Apache a znovu nahrajeme konfiguraci Apache.

```
# a2enmod ssl
# a2enmod rewrite
# a2enmod suexec
# a2enmod include
# a2enmod fcgid
# /etc/init.d/apache2 force-reload
```

* ***# aptitude install libmysql-ruby***

A to je vše, co potřebujeme. Další konfigurace jsou již v souvislosti s konkrétními aplikacemi v Rails. Předtím, než k těmto konfiguracím přistoupím, popíši model který používám.

Potřeboval jsem mít všechny aplikace na jednom serveru. Myšleno všechny přístupné pod jednou adresou serveru například www.example.com. Každá aplikace má na tomto serveru vlastní adresář. Já adresáře aplikací umístňuji do kořene serveru, takže mají adresy jako www.example.com/app1, www.example.com/app2 atd. Vlastní program a data aplikací jsou umístněny v `/usr/local/share/rails-app/app1`, `/usr/local/share/rails-app/app2`...

Nejdříve tedy ukázková aplikace test pro odzkoušení konfigurace.

Poznámka: Pro následující řádky, příkazy zadávané za výzvou `$` jsou v kontextu uživatele www-data.

```
# mkdir -p /usr/local/share/rail-app/test
# chown www-data /usr/local/share/rail-app/test
$ rails /usr/local/share/rails-app/test
```

Nejdříve si ověříme funkčnost aplikace ve vývojvém prostředí serveru WEBrick.

```
$ cd /usr/local/share/rails-app/test
$ script/server
```

Nasměrujeme svůj prohlížeč na adresu http://localhost:3000. Zde uvidíme uvítací stránku aplikace. Nyní WEBrick server ukončíme a nakonfigurujeme Apache. Do souboru `/etc/apache2/sites-available/default` kde je konfigurace „hlavního virtuálního serveru“ vložíme řádky definující naši testovací aplikaci. Pro přehlednost nejlépe někam na konec sekce `<VirtualHost>`.

```
NameVirtualHost *
<VirtualHost *>
        ⋮
        # Konfigurace zkušební Rails aplikace test.
        Alias /test/ "/usr/local/share/rails-app/test/public/"
        Alias /test "/usr/local/share/rails-app/test/public/"
        <Directory /usr/local/share/rails-app/test/public>
                Options ExecCGI FollowSymLinks
                AllowOverride All
```

```
                Order Allow,Deny
                Allow From All
        </Directory>
</VirtualHost>
```

Upravenou konfiguraci apache nahrajeme.

```
# /etc/init.d/apache2 reload
```

A do prohlížeče zadáme adresu http://localhost/test/index.html. Nyní vidíme opět uvítací stránku aplikace. Rozkliknutím About your application's environment zjistíme že ještě není vše úplně v pořádku. Chybí nám nastavení aplikace pro FCGI a rovněž nastavení kořenového adresáře aplikace. Toto učiníme v souboru `/usr/local/share/rails-app/test/public/.htaccess`. Nejdříve FCGI. Hned na druhém řádku souboru je nastavení ovladače pro fcgi skripty. Toto upravíme z původního

```
AddHandler fastcgi-script .fcgi
```

na

```
AddHandler fcgid-script .fcgi
```

Standardne je aplikace nakonfigurována na cgi. Přehození na FCGI učiníme v pravidle

```
RewriteRule ^(.*)$ dispatch.cgi [QSA,L]
```

které přepíšeme na

```
RewriteRule ^(.*)$ dispatch.fcgi [QSA,L]
```

Protože naše není jako taková v kořenu serveru, ale ve vlastním adresáři, musíme tuto okolnost v konfiguraci také zohlednit. Vyhledáme zakomentovaný řádek

```
#   RewriteBase /myrailsapp
```

a přepíšeme na

```
RewriteBase /test
```

Pokud si nyní rozklikneme About your application's environment, vidíme již vše bez chyby.

Konfigurace aplikace tedy znamená zahrnout do sekce <VirtualHost> souboru `/etc/apache2/sites-available/default` blok příkazů:

```
# Konfigurace Rails aplikace app
Alias /app/ "/usr/local/share/rails-app/app/public/"
Alias /app "/usr/local/share/rails-app/app/public/"
<Directory "/usr/local/share/rails-app/app/public">
        Options ExecCGI FollowSymLinks
        AllowOverride All
        Order Allow,Deny
        Allow From All
</Directory>
```

Na straně aplikace pak musí být v souboru `.../rails-app/app/public/.httaccess` předefinovány či přidány řádky:

```
AddHandler fcgid-script .fcgi
RewriteBase /app
RewriteRule ^(.*)$ dispatch.fcgi [QSA,L]
```

47.4.10. Apache2 CGI

Odkazy:
- . ()
- . ()

* ***FIXME:***

Příklad 47-1. `apache2.cgi.conf`

```
<Directory /var/www/webdb>
        Options ExecCGI FollowSymLinks
        AddHandler cgi-script .cgi

        SetEnv RAILS_ENV production
        AllowOverride all

        Order allow,deny
        Allow from all
</Directory>
```

47.4.11. Apache2 FCGI

Odkazy:
- How to set up Rails with mod_fcgi (http://www.tummy.com/Community/Articles/rails-fcgi/) na http://tummy.com
- HowtoSetupApacheWithFastCGIAndRubyBindings (http://wiki.rubyonrails.com/rails/pages/HowtoSetupApacheWithFastC
- How to install Ruby on Rails on Ubuntu 5.10 (http://claudio.cicali.org/article/74/how-to-install-ruby-on-rails-on-ubuntu-510)
- Apache2 with mod_fcgid (http://www.datanoise.com/articles/2006/04/06/apache2-with-mod_fcgid)
- Debian mod_fastcgi Notes (http://wiki.rubyonrails.org/rails/pages/Debian+mod_fastcgi+Notes)
- **FIXME:** ()

Poznámka: Uvedený postup je pro Debian Sarge, toho času stable.

FIXME: Předpokládám že máme nainstalováno vše co je popsáno v článku 47.4.9. Doinstalujeme modul `fcgid` do apache a `fcgi` knihovnu k ruby.

* *Na server rapp mám enablované následující moduly apache: cgid, fcgid, rewrite, ruby, ssl, userdir.*

```
# aptitude install libfcgi-ruby1.8 libapache2-mod-fcgid
# a2enmod fcgid
```

Upravil jsem konfiguraci fcgid modulu takto:

Příklad 47-2. `fcgid.conf`

```
<IfModule mod_fcgid.c>
        AddHandler fcgid-script .fcgi
        SocketPath /var/lib/apache2/fcgid/sock
        IPCCommTimeout 120
        IPCConnectTimeout 10
```

```
        MaxProcessCount 40
        ProcessLifeTime 86400
        IdleTimeout 1800
        DefaultMaxClassProcessCount 8
        DefaultInitEnv RAILS_ENV production
</IfModule>
```

FIXME:

Příklad 47-3. `apache2.fcgi.conf`

```
# Apache 2 configuration for Kontejnery using FastCGI (fcgid).

# Documentation directory is aliased into application/doc.

Alias /kontejnery/doc /home/rails/kontejnery/doc

<Directory /home/rails/kontejnery/doc>
        AllowOverride None

        # Access rules
        Order allow,deny
        Allow from all
</Directory>


# Application is aliased into public directory where are static pages
# and dispatcher.

Alias /kontejnery /home/rails/kontejnery/public

<Directory /home/rails/kontejnery/public>
        Options ExecCGI FollowSymlinks
        AllowOverride None

        RubySafeLevel 0
        RubySetEnv RAILS_ENV production

        RubyRequire apache/ruby-run
        <Files *.rb>
                SetHandler ruby-object
                RubyHandler Apache::RubyRun.instance
        </Files>

        # Restrict access to the application on apache level.  We
        # could specify any apache authentication.

        # AuthType Basic

        Order allow,deny
        Allow from all


        # Rewrite engine configuration

        RewriteEngine On
        RewriteBase /kontejnery
```

```
        RewriteRule ^$ index.html [QSA]
        RewriteRule ^([^.]+)$ $1.html [QSA]
        RewriteCond %{REQUEST_FILENAME} !-f
        RewriteRule ^(.*)$ dispatch.fcgi [QSA,L]

        # Specific error message for error 500.

        ErrorDocument 500 "<h2>Application fatal error</h2>Rails application failed to start p
</Directory>
```

Výstraha

Může se stát že nám aplikace nefunguje, sám jsem strávil několik hodin hledáním řešení na Internetu. Po různých pokusech jsem se nakonec dostal k tomu, že jsem neměl v pořádku oprávnění na adresáři `./tmp/sessions`.

Tip: Zkuste použít pro ukládání sessions databázi.

47.4.12. Phusion Passenger

Odkazy:

- User:Paul/TipsAndTricks/Phusion-Passenger Debian Lenny (http://www.loudas.com/s/User:Paul/TipsAndTricks/Phusion-Passenger_Debian_Lenny)
- Phusion Passenger™ on Debian (http://stereonaut.net/phusion-passenger%E2%84%A2-on-debian/)
- phusion-passenger (http://code.google.com/p/phusion-passenger/) na Google Code
- Phusion Passenger users guide (http://www.modrails.com/documentation/Users guide.html)
- Ruby on rails s apache2 (http://forum.ubuntu.cz/index.php?topic=52073.0)
-
-
-

47.4.12.1. Instalace Phusion Passenger na Debian Lenny z balíčků

Odkazy:

-
-
-
-

Do souboru `/etc/apt/sources.list` vložíme řádky:

```
# Phusion Passenger
deb http://debian.tryphon.org stable main contrib
```

A můžeme instalovat

```
# aptitude update
# aptitude search passenger
```

```
p   libapache2-mod-passenger        - Rails and Rack support for Apache2
p   passenger-common                - Rails and Rack support for Apache2
p   passenger-doc                   - Rails and Rac support for Apache2 - Documentation
```

```
# aptitude install libapache2-mod-passenger
```

Instalace skončí s chybou na nesplněné závislosti.

47.4.12.2. Instalace pomocí rubygems

Jsou použité lokální gemy které nejsou v sytému.

Opět jako v předchozím případě, musíme mít nainstalovánu vývojovou verzi ruby a pár dalších balíčků. Například pro ruby 1.8 je to

```
# aptitude install ruby1.8-dev make g++ apache2-prefork-dev libapr1-dev libaprutil1-dev
```

A nyní překlad phusion-passenger modulů pro Apache a Nginx.

```
# cd /usr/local/gems
# source setvars
# gem install passenger
# passenger-install-apache2-module
```

V průbehu instlace se objeví text

```
Please edit your Apache configuration file, and add these lines:

   LoadModule passenger_module /usr/local/gems/gems/passenger-2.2.11/ext/apache2/mod_passenger
   PassengerRoot /usr/local/gems/gems/passenger-2.2.11
   PassengerRuby /usr/bin/ruby1.8

After you restart Apache, you are ready to deploy any number of Ruby on Rails
applications on Apache, without any further Ruby on Rails-specific
configuration!

Suppose you have a Rails application in /somewhere. Add a virtual host to your
Apache configuration file and set its DocumentRoot to /somewhere/public:

   <VirtualHost *:80>
      ServerName www.yourhost.com
      DocumentRoot /somewhere/public    # <-- be sure to point to 'public'!
      <Directory /somewhere/public>
         AllowOverride all              # <-- relax Apache security settings
         Options -MultiViews            # <-- MultiViews must be turned off
      </Directory>
   </VirtualHost>

And that's it! You may also want to check the Users Guide for security and
optimization tips, troubleshooting and other useful information:

  /usr/local/gems/gems/passenger-2.2.11/doc/Users guide Apache.html

Enjoy Phusion Passenger, a product of Phusion (www.phusion.nl) :-)
http://www.modrails.com/
```

Vytvořil jsem si tedy konfigurační soubor pro Apache2 `/etc/apache2/conf.d/passenger`

```
LoadModule passenger_module /usr/local/gems/gems/passenger-2.2.11/ext/apache2/mod_passenger.so
PassengerRoot /usr/local/gems/gems/passenger-2.2.11
PassengerRuby /usr/bin/ruby1.8
```

Pokud používáme RubyGems instalované v uživatelském prostoru, nebo v lokálním prostoru, tedy oddělené od systémových RubyGems, musíme provést pár úprav. Nejdříve samotné spouštění ruby. Musíme si vytvořit obálku, vlastní skript, pro spouštění ruby, ve kterém nastavíme správně cesty ke knihovnám. Tuto si můžeme uložit například do `/usr/local/gems/bin/ruby-wrapper` s následujícím obsahem.

```
#!/bin/bash
export RUBYLIB=/usr/local/gems/lib
exec "/usr/bin/ruby1.8" "$@"
```

Můžeme si předefinovat více parametrů, jako je na stránce Passing environment variables to Ruby from Phusion Passenger (http://blog.phusion.nl/2008/12/16/passing-environment-variables-to-ruby-from-phusion-passenger/).

Poznámka: Nemělo by v `RUBYLIB` být `/usr/local/gems/lib:/usr/lib/ruby/1.8` ?

Další úpravou je konfigurace passenger modulu v Apache2. Zde musíme spouštět ruby přes náš **ruby-wrapper**. V souboru `/etc/apache2/conf.d/passenger` upravíme direktivu `PassengerRuby`.

```
# Load Passenger module and configure it.
LoadModule passenger_module /usr/local/gems/gems/passenger-2.2.11/ext/apache2/mod_passenger.so
PassengerRoot /usr/local/gems/gems/passenger-2.2.11
PassengerRuby   /usr/local/gems/bin/ruby-wrapper
```

Poslední úpravou je nastavení prostředí v konfiguraci virtuálního webu

```
<VirtualHost *:80>
        ...
        SetEnv GEM_HOME /usr/local/gems
        ...
<VirtualHost>
```

47.4.13. Lighttpd

* *section id="rails.lighttpd" xreflabel="Lighttpd" status="draft"*

FIXME:

47.4.14. Nginx + Passenger

*

Odkazy:

- Příprava Linuxového serveru pro hosting Ruby on Rails aplikací (http://tmatejicek.blogspot.com/2010/11/priprava-linuxoveho-serveru-pro-hostnig.html)
-

47.5. Konfigurace aplikace

Konfigurace aplikace napsané v Rails se nachází v adresáři `config`. Zde se mimo soubor `database.yml`, jenž popisuje použité databáze, nachází soubor `environment.rb` a adresář `environments` popisující prostředí aplikace.

47.5.1. Prostředí aplikace

* *section id="rails.environments"*

Odkazy:

- Environments in Rails 1.1 (http://glu.ttono.us/articles/2006/05/22/guide-environments-in-rails-1-1)
- Configuring Rails Environments: The Cheat Sheet (http://glu.ttono.us/articles/2006/05/22/configuring-rails-environments-the-cheat-sheet)

FIXME:

Aplikace může být spuštěna v několika různých prostředí. Prostředí ve smyslu vývojového prostředí, testovacího prostředí a produkčního prostředí. Tato tři uvedená prostředí jsou standardně vytvářena, nic nám ovšem nebrání vytvořit si prostředí další.

Z pohledu aplikace je prostředí vlastně konfigurací, sadou parametrů jenž ovlivňují běh aplikace.

A nyní si ukážeme, jak jsou jednotlivá prostředí popsána.

Při spouštění aplikace je nastaven parametr `ENV['RAILS_ENV']`, jenž obsahuje jméno prostředí. Jako první se zavádí soubor `config/environment.rb`. V něm jsou věci globální, společné všem prostředím. Konfigurace zde nastavená může být přepsáná konfigurací v souboru konkrétního prostředí.

Nastavujeme zde třeba `Inflector`.

V prostředí máme také možnost ovlivnit chování 47.26.1

47.5.2. Připojení k databázi

Připojení k databázi je popsáno v konfiguračním souboru `config/database.yml`. Zde je pro každé prostředí definován jeden záznam popisující připojení k databázovému serveru. Záznam definuje v první řadě použitý databázový konektor (adapter) a podle druhu databáze pak další potřebné údaje.

Ukázky konfigurací jednotlivých databázových konektorů a popis údajů které používají se nachází v části 47.3.

Při práci s konfiguračním souborem `config/database.yml` dodržujeme několik zásad.

- Soubor neukládáme do systému správy verzí (např. Subversion), protože jsou v něm hesla.
- Omezíme přístup k tomuto souboru pro ostatní uživatele ze stejného důvodu, jsou v něm hesla.
- Do systému správy verzí (např. Subversion) uložíme kopii tohoto souboru pojmenovanou `config/database.example` z které pečlivě odstraníme všechny citlivé údaje (hesla, jména účtů připadně i názvy serverů). Soubor připravíme tak aby se pouhým přejmenováním a dopsání přihlašovacích údajů dal použít jako `config/database.yml`.

47.6. Ladění aplikace

- Rails debugging - a (slightly) better approach? (http://lists.rubyonrails.org/pipermail/rails/2006-July/051456.html)
- HowtoDebugWithBreakpoint (http://wiki.rubyonrails.org/rails/pages/HowtoDebugWithBreakpoint) — deprecated
- HowtoDebugWithRubyDebug (http://wiki.rubyonrails.org/rails/pages/HowtoDebugWithRubyDebug)
- HowtoDebugViews (http://wiki.rubyonrails.com/rails/pages/HowtoDebugViews)
- Rails Debug Popup - Makes easy to debug info. (http://snippets.dzone.com/posts/show/697)
- Debugging Rails application (http://www.datanoise.com/articles/2006/9/15/debugging-rails-application)

FIXME:

Nejdříve v bodech.

- Máme k dispozici metodu `debug` jenž vypisuje v pohledu do html stránky své parametry. Nejvhodnější je ji použí v pohledu aplikace `app/views/layouts/application.rhtml`.
- Skript `script/breakpointer`.
- **FIXME:**
- BREAKPOINT_SERVER_PORT = 42531

47.6.1. Nezapracované texty/poznámky

Zapracovat tyto texty do sekce Ladění aplikace

Ladicí „tisk“ proměnné do html stránky. Prostě použijeme kód jako `<%= debug(@post) %>` pro zobrazení obsahu `@post`.

47.7. Databáze

Budování aplikace začneme jádrem, tedy databází. To znamená vytvoření struktury dataových tabulek a napsání předpisu pro jejich vytvoření v jazyce SQL. Pro věci kolem databáze používám adresář `app/db` který je třeba vytvořit.

```
$ mkdir app/db
```

47.8. Adresářová struktura a rake

FIXME:

./app
./components
./config
./db pracovní adresář

./doc výstupní adresář
./lib deníky běžící aplikace
./misc
./public
./script
./test testy
./vendor

```
rdoc.rdoc_files.include('app/**/*.rd')
```

47.9. Struktura aplikace

47.9.1. Databázový model

* *section id="rails.model" xreflabel="model"*

Odkazy:

- UnderstandingModels (http://wiki.rubyonrails.com/rails/pages/UnderstandingModels)
- . ()

Implementuje datový model / obchodní objekty. Rail používá pro reprezentaci datového modelu třídu `ActiveRecord`.

V RoR je pro práci s datovám modelem použita třída `ActiveRecord`. Tato třída má určité požadavky na strukturu databáze a datové tabulky.

- Objekt třídy `ActiveRecord` vytváří jméno datové tabulky z vlastního jména třídy. Předpokládá že jméno třídy reprezentující datovou tabulku podstatné jmnéno v jednotném tvaru (singulár) v anglickém jazyce. U datové tabulky pak předpokládá odpovídající jméno v množném tvaru (plurál). Pokud tomu tak není, lze jméno datové tabulky předefinovat pomocí příkazu `set_table_name` nebo nastavením proměnné `table_name`.

  ```
  class Call < ActiveRecord::Base
      set_table_name "rad_radios"
      ⋮
  end
  ```

  nebo

  ```
  class Call < ActiveRecord::Base
      table_name = "rad_radios"
      ⋮
  end
  ```

  Obě varianty jsou v použití rovnocenné.
- Dalším skrytým předpokladem třídy `ActiveRecord` je název sloupce s jednoznačným identifikátorem (klíčem). `ActiceRecord` předpokládá že tento sloupec se jmenuje `id`. Jeho jméno můžeme opět předefinovat příkazem `set_primary_key` nebo nastavením proměnné `primary_key`.

  ```
  class Call < ActiveRecord::Base
      set_table_name "rad_radios"
      set_primary_key "rad_id"
      ⋮
  end
  ```

nebo

```
class Call < ActiveRecord::Base
    table_name = "rad_radios"
    primary_key = "rad_id"
    ⋮
end
```

Obě varianty jsou v použití rovnocenné.

- **FIXME:**Některé databázové servery a nebo jejjich starší verze neznají automaticky generované sekvenční čísla, jednou z takových databází je Firebird.

```
class Call < ActiveRecord::Base
    set_sequence_name "rad_radios_generator"
    ⋮
end
```

nebo

```
class Call < ActiveRecord::Base
    sequence_name = "rad_radios_generator"
    ⋮
end
```

Pokud názvy našich tabulek v databázi nejsou v angličtině a nevyhovují nárokum ActiveRecord, je třeba provést před vytvořením modelu úpravy v inflectoru. Zavedeme tedy do konfigurace `config/environment.rb` tvar jednotného a množného čísla jména tabulky. Například pro tabulku středisek jenž se jmenuje `strediska` zavedeme jednotné a množné číslo tohoto jména.

```
Inflector.inflections do |inflect|
    inflect.irregular 'stredisko', 'strediska'
end
```

Hned si ověříme, jestli vše funguje jak potřebujeme.

```
$ script/console
Loading development environment
>> Inflector.pluralize 'stredisko'
=> "strediska"
>> Inflector.singularize 'strediska'
=> "stredisko"
```

Teď teprve můžeme přistoupit k vygenerování modelu. Model generujeme příkazem **script/generate model**.

script/generate model [volby] *`název_modelu`*

V našem případě, kd máme tabulku středisek pojmenovanou `strediska`, vytvoříme model pojmenovaný jednotným číslem `Stredisko`.

```
$ script/generate model --svn Stredisko
      exists  app/models/
      exists  test/unit/
      exists  test/fixtures/
      create  app/models/stredisko.rb
A         app/models/stredisko.rb
      create  test/unit/stredisko_test.rb
A         test/unit/stredisko_test.rb
      create  test/fixtures/strediska.yml
A         test/fixtures/strediska.yml
```

```
      exists  db/migrate
      create  db/migrate/002_create_strediska.rb
A         db/migrate/002_create_strediska.rb
```

Například model pro datovou tabulku `hosts` vytvoříme a přidáme do subversion příkazem

```
$ scrip/generate model --svn Host
      exists  app/models/
      exists  test/unit/
      exists  test/fixtures/
      create  app/models/host.rb
A         app/models/host.rb
      create  test/unit/host_test.rb
A         test/unit/host_test.rb
      create  test/fixtures/hosts.yml
A         test/fixtures/hosts.yml
      create  db/migrate
A         db/migrate
      create  db/migrate/001_create_hosts.rb
A         db/migrate/001_create_hosts.rb
$
```

47.10. Migrace / Migrations

* *section id="rails.migrations"*

Migrace jsou nástroj pomocí kterého můžeme udržovat strukturu své databáze. Jsou pro databázi tím, čím je pro programový kód systém správy verzí. Jednotlivé migrační soubory jsou uloženy v adresáři `db/migrate` a jejich jména začínají třemi číslicemi oddělenými od názvu migrace znakem '_'. Název migrace je libovolný popisný text který má vypovídat co migrace dělá. Tímto číslem, číslování začíná od čísla 1 jsou migrace jednoznačně určeny a toto číslo současně označuje verzi databázové struktury. Obsah migračních souborů tvoří dvě metody, `self.up` a `self.down`, popisující jakým způsobem se mění struktura při upgrade z předchozí verze na označenou verzi a při downgrade z označené verze na verzi předcházející. Postupným prováděním jednotlivých migrací můžeme upgradovat nebo downgradovat strukturu databáze na libovolnou verzi. Můžeme se tedy vrátit k libovolné verzi databázové struktury od okamžiku kdy jsme začali migrace používat. K tomu aby migrace správně fungovali, potřebují znát aktuální verzi databáze. Toto číslo je poznamenáno v jediném řádku tabulky `schema_info` obsahující jediný sloupec `version` typu integer. Dále migrace udržují ještě jeden soubor a to `db/schema.rb` popisující aktuální strukturu databáze stejným způsobem jako v samotných migracích.

* *Přeformulovat přdchozí odstavec.*

Prvním krokem k používání migrací který uděláme je, že si vytvoříme již zmíněný soubor `db/schema.rb` popisující aktuální strukturu databáze.

```
$ rake db:schema:dump
```

Pokud je naše databáze prázdná, výsledkem bude soubor který její strukturu popisuje takto:

```
ActiveRecord::Schema.define() do

end
```

Pok

* ***WORKING: Editovat.***

Vytvoříme databáze, vytvoříme náš projekt a nakonfigurujeme přístup k databázi v konfiguračním souboru `config/database.yml`. Jestli jsme neudělali chybu si ověříme příkazem

```
$ rake db:schema:dump
```

Tento příkaz se připojí k databázi a vytvoří soubor `db/schema.rb` popisující strukturu databáze. Pokud je naše databáze prázdná (čistě vytvořená), neobsahuje žádné tabulky a schema.rb ji popisuje takto:

```
ActiveRecord::Scheme.define() do
end
```

Nyní přistoupíme k tvorbě datového modelu a struktury databáze. Začneme vytvořením modelu

```
$ script/generate model Person
      exists  app/models/
      exists  test/unit/
      exists  test/fixtures/
      create  app/models/person.rb
      create  test/unit/person_test.rb
      create  test/fixtures/people.yml
      create  db/migrate
      create  db/migrate/001_create_people.rb
```

Všiměte si, že mimo vlastního modelu (soubor `app/models/person.rb`) je vytvořena i migrace na verzi databáze 1 v souboru `db/migrate/001_create_people.rb`. Tato obsahuje zatím pouze vytvoření a odstranění tabulky people.

```
class CreatePeople < ActiveRecord::Migration
  def self.up
    create_table :people do |t|
      t.timestamps
    end
  end

  def self.down
    drop_table :people
  end
end
```

Dve mětody této migrace se spouští při upgrade z předchozí verze na verzidefinovanou migrací (`self.up`) a při downgrade o verzi níže (`self.down`.

Při vlastních migracích se spouštějí v daném pořadí metody `self.up` a `self.down` podle toho ke které verzi databáze chceme migrovat. V databázi je tabulka `schema_info` osahující jediný sloupec `version` se jedinou číselnou hodnotou určující na jaké verzi se právě databáze nachází. Voláním

```
$ rake db:migrate
```

Přejdeme k nejvyšší definované verzi a příkazem

```
$ rake db:migrate VERSION=číslo
```

pak ke konkrétní verzi. Číslo 0 znamená návrat před migraci číslo 1 což v připadě že jsme celou strukturu databáze udržovali pouze v migracích znamená návrat k prázdné databázi.

Od verze Ruby on Rails 2.0 má rake ještě jednu úlohu a to:

```
$ rake db:rollback
```

Ale teď zpět k migracím a k definici sloupců v tabulce. Tyto definice jsou tvaru:

t.column :*`název-sloupce`*, :*`typ-dat`* [*`volby`*]

Jsou možné tyto typy dat:

Tabulka 47-1. Typy dat v migracích

název		
:binary		
:boolean		
:date		
:datetime		
:decimal		
:float		
:integer		
:string		
:text		
:time		
:timestamp		

Tabulka 47-2.

název	**typ hodnot**	**význam**
:limit	číslo	délka pole, například maximální délka řetězce
:default	libovolná dle typu	defaultní/implicitní hodnota pokud není zadána
:null	true | false	zdali je přípustná hodnota NULL (nil)
:precision		
:scale		

Od verze Ruby on Rails 2.0 je možno popisovat sloupce způsobem:

t.*`typ-dat`* :*`název-sloupce`* [, *`volby`*]

K dispozici je rovněž speciální hodnota/typ sloupce

```
t.timestamps
```

Tento zápis znamená:

```
t.column :created_at, :datetime
t.column :updated_at, :datetime
```

Tabulka 47-3. Příkazy migrací

tabulky	add_table, drop_table, rename_table

sloupce	add_column, remove_column, rename_column, change_column
indexy	add_index, remove_index

47.10.1. Vytvoření migrace generátorem

S vytvářením migrace nám může pomoci generátor

```
$ script/generate migration CreateLekarny
```

47.10.2. Vytvoření migrace nového modelu

Pokud vytváříme migrací novou tabulku, použijeme k jejímu vytvoření generátor `model`. Jeho použití je následující:

```
script/generate model Název-modelu [sloupec:typ] ...
```

Například novou tabulku uživatelů vytvoříme třeba takto:

```
$ script/generate model --svn User first_name:string last_name:string user_name:string passwor
```

47.10.3. Změna dat v migracích

V migracích můžeme měnit nejen strukturu ale i data v databázi.

```
class ZmenyVDatech < ActiveRecord::Migration
    def self.up
        User.transaction do
            User.find(:all, :conditions => [...]).each do |user|
                ... do something with user
                user.role = 'actor' if some condition
                user.save
            end
        end
    end

    def self.down
        User.transaction do
            User.find(:all, ...).each do |user|
                ... revere changes done in self.up
                user.save
            end
        end
    end
end
```

47.10.3.1. Přidání pole a indexu

Nejdříve přidání a odebrání pole, toto jsou jednořádkové příkazy v `self.up` a `self.down migrace`. Aby bylo jasné co píšu, tak se jedná o přídání nového sloupce s názvem `poradi` do tabulky `produkty`. Sloupec je číselného typu, tedy `integer` (:integer).

```
def self.up
    add_column :produkty, :poradi, :integer
end
def self.down
    remove_column :produkty, :poradi
end
```

Nově přidaný sloupec má sloužit jako pořadové čislo. Určuje tak vzájemné pořadí záznamů a nemá žádný další význam.

```
class AddPoradiToProdukty  < ActiveRecord::Migration
  def self.up
    add_column :produkty, :poradi, :integer
    add_index  :produkty, :poradi, :unique => true
    counter = 0
    Produkt.find(:all).each { |produkt|
      produkt.poradi = counter
      produkt.save
      counter += 1
    }
  end

  def self.down
    remove_index  :produkty, :poradi
    remove_column :produkty, :poradi
  end
end
```

47.10.4. Migration Cheat Sheet

Odkazy:

- Rails Migrations Cheatsheet (http://dizzy.co.uk/ruby_on_rails/cheatsheets/rails-migrations)
- Migrations (http://railstutor.org/projects/1/wiki/Migrations) na Rails Tutor
-
-

FIXME:

Tabulka 47-4. příkazy

příkaz	co dělá
rake db:migrate [VERSION=xxx]	migruje databázi na nejvyšší či udanou verzi
rake db:schema:dump	vytvoří `db/schema.rb` podle databáze
rake db:schema:load	vytvoří databázi podle schematu v `db/schema.rb`

příkaz	co dělá
rake db:structure:dump	vytvoří popis struktury databáze v SQL v `development_structure.sql`
rake db:sessions:create	vytvoří migraci pro ukládání sessions v databázi
rake -T ^db:	seznam rake cílů začínajících `db:`

Standardně pracuje **rake** ve vývojovém prostředí. Tím myslím že všechny příkazy pracující s databází se vykonávají na vývojářskou verzí databáze. Pokud potřebujeme vykonat některé příkazy na produkčním serveru použijeme parametr `RAILS_ENV=production`. Například migrace na produkčním serveru se udělá takto:

```
$ rake db:migrate RAILS_ENV=production
```

Změnit tabulku

```
change_table :table_name do |t|
    t.change :column_name, :new_column_type
    t.remove :column_name
end
```

Vytvořit tabulku

```
create_table :table_name, {table_options} do |t|
    t.string :name, {column_options}
end
```

47.11. Active Record

* *section id="rails.ActiveRecord" xreflabel="ActiveRecord"*

- Class ActiveRecord::Base (http://api.rubyonrails.org/classes/ActiveRecord/Base.html)
- Rails 2.0.2 documentation (http://www.railsbrain.com/api/rails-2.0.2/doc/)

Jak jsem již zmínil dříve, komponenta která je zodpovědná za přístup k datům v RoR se jmenuje Active Record a je reprezentována třídou `ActiveRecord` (gem activerecord). Tato zajišťuje vše od přístupu k databázi až k zpřístupnění dat z této databáze ve formě Ruby objektů.

Active Record klade na strukturu a pojmenování objektů v databázi určité nároky, které si dále popíšeme. Některé z nich lze obejít, a jiné jsou důležité pro správné fungování Active Record.

* ***WORKING: Editovat.***

Pro přístup k datům se používá ActiveRecord. Pokdu tedy definujeme tabulku lidí, vytvoříme model podle vzoru.

```
class Clovek < ActiveRecord::Base
```

Poznámka: Uvedený model předpokládá že máme do inflektoru zavedenou výjimku

```
Inflector.inflections do |inflect|
    inflect.irregular 'clovek', 'lide'
end
```

Jak je vidět, model pro tabulku lidí se jmenuje `Clovek`, ale samotná tabulka pak `lide`. Pokud potřebujeme v modelu určit jiné jméno tabulky, můžeme tak učinit příkazem **set_table_name**

```
class Clovek < ActiveRecord::Base
    set_table_name 'tab015'
end
```

`ActiveRecord` kladou na naše datové tabulky další omezení, která musíme dodržet. Jedním z nejdůležitějších je typ a název klíčového sloupce. `ActiveRecord` předpokládá že tento klíčový sloupec existuje, je typu `Integer` a jmenuje se `id`. Pokude se tento sloupec v naší tabulce jmenuje jinak, oznámíme to `ActiveRecord` příkazem

```
set_primary_key 'myid'
```

FIXME:

`belongs_to`

FIXME:

```
belongs_to :employee, :foreign_key => "pin"
```

`has_one`

FIXME:

`has_many`

FIXME:

```
has_many :offices, :foreign_key => "city_id"
```

`has_and_belongs_to_many`

FIXME:

`validates_presence_of`

FIXME:

`validates_acceptance_of`

FIXME:

`validates_uniqueness_of`

FIXME:

47.11.1. Přístup k SQL serveru

Přístup k SQL serveru je konfigurován v souboru `config/database.yml`. Zde jsou uvedeny přihlašovací informace pro každou databázi, produkční, vývojovou i testovací. V tomto konfiguračním souboru jsou uvedeny parametry tak jak se použijí při sestavení spojení na SQL server. Podívejme se tedy jak se připojíme k SQL serveru přímo bez použití tohoto konfiguračního souboru. Použijeme k tomu metodu `ActiveRecord::Base.establish_connection` které v pojmenovaných argumentech předáme parametry spojení. Například k naší databázi se připojíme takto:

```
$KCODE='u'
require 'jcode'
require 'rubygems'
require 'active_record'
```

```
ActiveRecord::Base.establish_connection(
                                        :adapter => 'postgresql',
                                        :host => '/var/run/postgresql',
                                        :encoding => 'utf8',
                                        :username => 'roxana',
                                        :password => 'cokolada',
                                        :database => 'rorex'
                                        )
```

Co jednotlivé parametry znamenají? Jako první popíši parametr `:adapter`. Tento specifikuje databázový stroj/server v kterém jsou naše data. Nejběžněji používané SQL servery jsou postgresql, mysql a sqlite. Existují ovšem adaptéry pro další databázové servery.

```
:adapter => 'sqlite'   # postgresql/mysql/sqlite/...
```

Další parametry jsou specifické pro použitá databázový adapter. Většina sql serverů má těchto pár parametrů:

- `:host` — adresa počítače na kterém SQL server běží, nebo cesta k socketu beží-li na stejném stroji
- `:port` — port na kterém očekává SQL server spojení, pokud není použit standardní
- `:database` — název databáze
- `:username`, `:password` — přihlašovací jméno a heslo do databáze

Vytvoření databáze v PostgreSQL serveru

```
# sudo -u postgres psql -d template1
template1=# CREATE USER pavel WITH ENCRYPTED PASSWORD 'tomasek' NOCREATEDB NOCREATEUSER;
template1=# CREATE DATABASE gblog WITH OWNER=pavel TEMPLATE=template0 ENCODING='utf-8';
```

K takto vytvořené databázi potřebujeme přístup. To uděláme úpravou konfiguračního souboru

FIXME: Ukázka připojení k postgresql serveru. K takto vytvořené databázi musíme zajistit přístup. To se provede dopsáním následujících řádku do `pg_hba.conf`:

```
local gblog pavel md5
```

```
# /etc/init.d/postgresql/7.4 reload
```

Vytvoření databáze v MySQL serveru

```
# sudo - mysql mysql
mysql> CREATE DATABASE rorex;
mysql> CREATE DATABASE rorexdev;
mysql> CREATE DATABASE rorextest;
mysql> GRANT ALL PRIVILEGES ON rorex.* TO 'roxana'@'localhost' IDENTIFIED BY 'cokolada';
mysql> GRANT ALL PRIVILEGES ON rorexdev.* TO 'roxana'@'localhost' IDENTIFIED BY 'cokolada';
mysql> GRANT ALL PRIVILEGES ON rorextest.* TO 'roxana'@'localhost' IDENTIFIED BY 'cokolada';
```

FIXME: Ukázka připojení k mysql serveru.

FIXME: Ukázka použití sqlite.

47.11.2. Vztahy mezi tabulkami (relace)

V naší databázi máme více tabulek, mezi kterými jsou vztahy. Představme si například databázi jednoduchého blogu, ve které máme tyto dvě tabulky:

```
class Post < ActiveRecord::Base
end
class Person < ActiveRecord::Base
end
```

Mezi těmito tabulkami existuje přirozený vztah daný tím, že každý příspěvek (Post) má jen jednoho autora (Person) a každý autor může mít více příspěvků. Je to vztah 1 ku n, (one to many). Takový vztah popíšeme příkazy **has_many** (má více) a **belongs_to** (patří k). V našem případě příspěvek (Post) patří k (belongs_to) člověku (Person). A obráceně Člověk (Person) má více (has_many) příspěvků (posts). V modelu to vypadá takto:

```
class Person < ActiveRecord
    has_many :posts
end
class Post < ActiveRecord
    belongs_to :person
end
```

To odpovídá databázi:

```
CREATE TABLE person (
    id SERIAL,
    first_name VARCHAR(30),
    last_name VARCHAR(30),
    ⋮
);
CREATE TABLE post (
    id SERIAL,
    person_id INTEGER,
    title VARCHAR(250),
    content TEXT,
    ⋮
);
```

* ***WORKING: Editovat.***

```
belongs_to :author, :class_name => 'Person', :foreign_key => 'author_id'
has_many :posts, :foreign_key => 'author_id'
```

47.11.3. Připojení tabulky z jiné databáze

Odkazy:

- Rails' has_many with models over different databases (http://blog.whitet.net/articles/2008/01/30/rails-has_many-with-models-over-different-databases)
- Multiple database handling with Rails (http://www.railsonwave.com/railsonwave/2006/12/11/multiple-database-handlng-with-rails)
- Rails Interacting with an External Database (http://www.pjhyett.com/posts/186-rails-interacting-with-an-external-database)

- Multiple Database Connection in rails (http://anandmuranal.wordpress.com/2007/08/23/multiple-database-connection-in-rails/)
- Sharing External ActiveRecord Connections (http://pragdave.pragprog.com/pragdave/2006/01/sharing_externa.html)
- HowtoUseMultipleDatabases (http://wiki.rubyonrails.org/rails/pages/HowtoUseMultipleDatabases)
-

Pokud chceme připojit některé tabulky z jiné databáze, začneme od konfigurace připojení k databázi. Do souboru `config/database.yml` vložíme nové sekce pro produkční a vývojovou databázi. Aplikace k jejimž datům se chceme připojit je taktéž v RoR a jmenuje se `bestapp`. Její produkční databáze je `bestapp_production` a vývojářská databáze `bestapp_development`. Do konfiugrace si tedy přidáme dvě sekce:

```
bestapp_production:
  adapter: postgresql
  socket: /var/run/postgresql
  encoding: UTF8
  database: bestapp_production
  username: bestak
  password: jehoheslo

bestapp_development:
  adapter: postgresql
  socket: /var/run/postgresql
  encoding: UTF8
  database: bestapp_development
  username: vyvojar
  password: mojeheslo
```

Důležité je správné pojmenování sekcí, které nám zjednoduší kód. Vlastní databáze se mohou jmenovat jakkoliv. Snažím se ovšem zachovávat konvenci že se databáze jmenuje po aplikaci a má ke jménu doplněno `development` nebo `production`.

Máme tedy připravené konfigurace připojení k databázím. Pokročíme tedy k dalšímu kroku což je vytvoření abstrakní třídy pro připojení k těmto databázím. Já tuto třídu pojmenovávám po aplikaci ke které se připojuji. V našem případě to tedy bude `Bestapp`:

```
class Bestapp < ActiveRecord::Base
    self.abstract_class = true
    establish_connection "bestapp_#{RAILS_ENV}"
end
```

V příkazu pro připojení k databázi **establish_connection** jsme s výhodou využili konvenci v pojmenování konfigurací. Protože proměnná `RAILS_ENV` obsahuje reži ve které server běží (`developlment`/`production`), zajistí nám připojení k té správné databázi jak na vývojářském tak na produkčním serveru..

Nyní si již můžeme připojit vlastní datové tabulky:

```
class Clovek < Bestapp
    has_many :funkce
end
```

Jak jsem ukázal, můžeme navázat mezi tabulkymi v různých databázích i relační vztahy. Ale tady bych byl opatrný, protože není zajištěno že bude vše fungovat tak jako kdyby tabluky byly ve stejné databázi. V projektu který jsem psal mi k mé spokojenosti vztah *has_many* a *belongs_to* mezi databázemi fungoval k mé spokojenosti.

47.11.3.1. Poznámky

```
class Forum < ActiveRecord::Base
    set_table_name 'LUM_User'
    set_primary_key 'UserID'
    self.establish_connection(:adapter => "mysql", :host => "localhost", :database => "vanilla

    def self.new_user(params)
        f = self.new
        f.RoleID = 3
        f.Name = f.FirstName = f.LastName = params['caccount']['login']
        f.Password = MD5.new(params['account']['password']).hexdigest
        f.Email = params['account']['email']
        f.DateFirstVisit = f.DateLastActive = Time.now
        f.save
    end
end

class MyModel < ActiveRecord::Base
    self.connection = "name_in_database_yml"
end

class LegacyBase < ActiveRecord::Base
    establish_connecition "name_in_database_yml"
end

class LegacyOrder < LegacyBase
end

class LegacyLineItem < LegacyBase
end
```

47.11.4. Validace dat

ActiveRecord::Validations::ClassMethods

- validates_acceptance_of
- validates_associated
- validates confirmation_of
- validates_each
- validates_exclusion_of
- validates_format_of
- validates_inclusion_of
- validates_length_of
- validates_numericality_of
- validates_presence_of
- validates_size_of
- validates_uniqueness_of

```
class Person < ActiveRecord::Base
    validates_presence_of :first_name, :last_name, :login, :email
    validates_format_of :email, :with => /\A([^@\s]+)@((?:[-a-z0-9]+\.)+[a-z]{2,})\Z/i
    validates_inclusion_of :funkce, :in => %w[THP VL OZ S P]
    validates_uniqueness_of :login

    private

    def validate
        errors.ass(:person, " can't be root.") if self.login == 'root'
    end
end
```

47.11.4.1. `validates_presence_of`

Metoda ověřuje zdali byla zadána v poli/polích nějaká hodnota.

validate_presence_of :field1, :field2, ...

47.11.4.2. `validates_uniqueness_of`

```
validates_uniqueness_of :ico, :allow_blank => true
```

```
validates_uniqueness_of :ico, :allow_nil => true
```

47.11.4.3. `validates_length_of`

```
validates_length_of :ico, :maximum => 10
```

```
validates_length_of :rodne_cislo, :within => 9..10
```

- `:maximum =>` *n* — minimální velikost
- `:minimum =>` *n* — maximální velikost
- `:within =>` *range* — velikost je v uvedeném rozsahu, například `7..12`
- `:in` — synonymum/alias pro `:within`
- `:is =>` *n* — velikost je přesně *n*
- `:allow_nil => true` — hodnota nemusí být zadána
- `:too_long => "text"` — zpráva v případě že hodnota je delší. Standardní zpráva je `"is too long (maximum is %d characters)"`
- `:too_short => "text"` — zpráva v případě že hodnota je kratší. Standardní zpráva je `"is too short (minimum is %d characters)"`
- `:wrong_length => "text"` — zpráva v případě že nevyhoví :is. Standardní správa je `"is the wrong length (should be %d characters)"`
- `:message => "text"` — chybová hláška v případě neúspěchu podmínky, je aliasem na `:too_long`, `:too_short` nebo `:wrong_length`

- `:on` — říká kdy se provádí kontrola, standardní hodnota je `:save`, alternativní hodnoty jsou `:create` a `:update`
- `:if` — specifikuje proceduru jenž podmiňuje validaci, například `:if => :allow_validation` nebo `:if => Proc.new{ |user| user.signup_step > 2}`

47.11.4.4. Nezapracované poznámky

Ukázky kódu.

```
validates_length_of :title, :maximum => 12, :message => 'Titul je příliš dlouhý - maximálně 12
```

ActiveRecord::Validations (http://rails.rubyonrails.com/classes/ActiveRecord/Validations.html):

```
protected
  def validate
    errors.add_on_empty %w( first_name last_name )
    errors.add("phone_number", "has invalid format") unless phone_number =~ /[0-9]*/
  end

  def validate_on_create # is only run the first time a new object is saved
    unless valid_discount?(membership_discount)
      errors.add("membership_discount", "has expired")
    end
  end

  def validate_on_update
    errors.add_to_base("No changes have occurred") if unchanged_attributes?
  end
```

47.12. Řadič (Controller)

* *section id="rails.controller"*

Odkazy:

- UnderstandingControllers (http://wiki.rubyonrails.com/rails/pages/UnderstandingControllers)
- UnderstandingMVC (http://wiki.rubyonrails.com/rails/pages/UnderstandingMVC)
- **FIXME:** ()

Řadič (Controller) je objekt který zprostředkuje spojení mezi pohledem a datovým modelem / obchodními objekty. V jednom směru přijímá uživatelské požadavky přicházející skrze pohled a zprostředkovává interakci s obchodními objekty (daty). A ve druhém přebírá data z obchodních objektů a umožňuje jejich zobrazení pohledem.

V řadiči je vlastně implementována logika uživatelského rozhraní. Co se stane když zmáčknu tady to tlačítko na stránce, jak se použije hodnota vyplněná do políčka formuláře, která stránka se zobrazí jako další, ...

Z hlediska kódu je řadič specializací třídy `ApplicationController`. Jednotlivé metody objektu této třídy jsou volány v odpovědi na akce uživatele přicházející prostřednictvím pohledu, tedy uživatelského rozhraní. Metody řadiče jsou přímo přístupny přess www rozhraní. V případě užití Apache `http://server/aplikace/řadič/metoda` a v případě serveru WEBrick `http://server:port/řadič/metoda`. U níže uvedeného příkladu řadiče tedy voláme metodu `index` pomocí url `http://localhost:3000/hello/index`.

```
class HelloController < ApplicationController
    def index
        render_text "Hello user!"
    end
end
```

Jak tedy řadič vytvoříme? K tomu nám poslouží generátor **script/generate controller**, jako parametr mu předáme název řadiče, a seznam akcí na které má reagovat. Jméno řadiče by mělo být podstatné jméno v jednotném tvaru v jazyce anglickém. Generátor vytvoří šablony všech souborů jenž s řadičem souvisí.

script/generate controller *`ControllerName`* [*`Actions`*]

```
$ script/generate controller dog
      exists  app/controllers/
      exists  app/helpers/
      create  app/views/dog
      exists  test/functional/
      create  app/controllers/dog_controller.rb
      create  test/functional/dog_controller_test.rb
      create  app/helpers/dog_helper.rb
```

Soubory které se vytvoří jsou tedy:

`app/controller/`*`jméno`*`_controller.rb`

Soubor s řadičem. Každá veřejná metoda třídy řadiče reprezentuje jednu akci jenž je možno z WWW rozhraní volat.

`app/functional/`*`jméno`*`_controller_test.rb`

Soubor s unit testy řadiče. Použijeme pokud aktivně používáme metody extrémního programování v Rails. Vřele doporučuji.

`app/helpers/`*`jméno`*`_helper.rb`

Pomocné metody používané pohled tohoto řadiče. **FIXME:**Zde je to pravé místo pro kód jenž je společný všem pohledům.

`app/views/`*`jméno/`*

Toto není soubor, ale adresář. V tomto adresáři budou pohledy které řadič používá, respektive skrze které zobrazuje data. Běžně odpovídají pohledy názvům metod řadiče.

FIXME:

- Controller stojí mezi databází a zobrazovacím systémem. Implementuje veškeré akce/procesy za pohledem.
- Ačkoliv jsou ve většině případů Controllery svázány s daty v datových tabulkách, můžmeme mít i controller který žádný datový zdroj přímo nepoužívá.

```
$ script/generate controller call
      exists  app/controllers/
      exists  app/helpers/
      create  app/views/call
      exists  test/functional/
      create  app/controllers/call_controller.rb
      create  test/functional/call_controller_test.rb
      create  app/helpers/call_helper.rb
```

Skript vytvoří prázdný controller. Do něj můžeme umístnit na rychlo lešení (`scaffold`).

```
class CallController < ApplicationController
    scaffold :call
end
```

Chceme-li předat nějakou hodnotu do pohledu (view), zapíšeme ji v controlleru do proměnné instance například `@moje_hodnota`. V pohledu se na ni pak odkážeme třeba takto `<%= @moje_hodnota %>`.

47.12.1. Užití controlleru bez podkladové datové tabulky

Nemá-li controller/řadič datovou tabulku, potřebujeme mechanismus kterým budeme uchovávat vybraná data (hodnoty proměnných) mezi zobrazeními stránek (akcemi). Tímto mechanismem může být sezení (Session). Funguje to následovně. V pohledu uvedeme formulář do kterého se zadávají informace. Zde použijeme funkci `text_field` pro vytváření těchto polí.

```
<%= start_form_tag :action => 'jmeno' %>
  Jméno: <%= text_field "frm", "jmeno" %><br/>
  Klíč: <%= text_field "frm", "klic" %><br/>
  <%= submit_tag "OK" %>
<%= end_form_tag %>
```

V akci, která se na odeslání formuláře provede (`:action => 'jmeno'`), vyčteme z parametrů hodnoty zapsané uživatelem do formuláře a uložíme tyto do sezení (session).

```
class PokusController < ApplicationController
...
    def jmeno
        @session['jmeno'] = @params['frm']['jmeno']
        @session['klic']  = @params['frm']['klic']
    end
```

Uložené hodnoty můžeme ihned použít, například v pohledu metody `jmeno`.

```
<p>Z předchozí strany tedy víme že bylo zadáno
jméno:<%= @session['jmeno']-%> a klíč: <%= @session['klic']-%></p>
```

47.12.2. Vytvoření řadiče

```
script/generate Radic metoda1 metoda2 ...
```

```
$ script/generate
```

47.13. Pohled (View)

* *section id="rails.view"*

47.13.1. Použití komboboxu a výběr možností z databáze

Následující ukázka je z jednoho mého projektu. Účelem tothoto kódu ve formuláři `app/views/shots/_form.rhtml`, který slouží pro zadávání nových `Shots` je zobrazit pole `pressman_id` jako box s výberem položek jednotlivých tiskařů.

```
<p><label for="pressman_id">Tiskař</label><br/>
<%= select('shot', 'pressman_id', Employee.find_all.collect{|p| [p.name_pin, p.id]}) %>
<p>
```

Jednotlivé parametry metody `select` jsou: **FIXME:**

47.14. Kostra formuláře (Scaffold)

Tak jak můžeme generovat soubory modelu, řadiče a pohledu. Tak můžeme všechno vygenerovat najednou jedním generátorem `scaffold`.

script/generate scaffold ModelName [ControllerName] [action, ...]

```
$ script/generate Pracovnik
      exists  app/controllers/
      exists  app/helpers/
      create  app/views/pracovnici
      exists  test/functional/
  dependency  model
      exists    app/models/
      exists    test/unit/
      exists    test/fixtures/
        skip    app/models/pracovnik.rb
   identical    test/unit/pracovnik_test.rb
   identical    test/fixtures/pracovnici.yml
      create  app/views/pracovnici/_form.rhtml
      create  app/views/pracovnici/list.rhtml
      create  app/views/pracovnici/show.rhtml
      create  app/views/pracovnici/new.rhtml
      create  app/views/pracovnici/edit.rhtml
      create  app/controllers/pracovnici_controller.rb
      create  test/functional/pracovnici_controller_test.rb
      create  app/helpers/pracovnici_helper.rb
      create  app/views/layouts/pracovnici.rhtml
   identical  public/stylesheets/scaffold.css
```

47.15. Formuláře

V předchozích částech jsme si ukázali jednotlivé komponenty jenž Rails používají. Nyní si ukážeme jak jejich kombinací dosáhneme vytvoření formuláře. Tedy webowské stránky jenž obsahuje editovatelná pole.

Formulář vzniká kombinací řadiče (controller) a pohledu (view). Protože ve formuláři zobrazujeme a editujeme data z databáze, použijeme i datový model (model).

Vzájemná součinnost těchto komponent je následující. Controller připraví datové objekty pomocí modelu a ty předá pohledu (view) k zobrazení. Pohledem vytvořenou html stránku zobrazíme v prohlížeči, editujeme infor-

mace a přes akční tlačítka či odkazy voláme řadič (controller) ke zpracování dat. Řadič po zpracování dat připraví nová a cyklus začíná od začátku.

47.15.1. Vstupní pole bez datového modelu

Odkazy:

- ASDwR: Working with Nonmodel Fields, page 354

Občas potřebujeme na formuláři použít vstupní pole, které není svázáno se žádným datovým modelem. Takové „nezávislé“ pole.

* *Důvodem pro takovýto postup může být potřeba řešit jiným způsobem vstup data do databáze. Tedy když nemůžeme přímo editovat řádky v tabulce ale musíme dodžet postup jenž je garantován vstupním algoritmem. Vstupní algoritmus používá tedy vlastní pole a řízeně zapisuje spočtené hodnoty do databáze.*

Pro vytvoření textového pole použijeme místo funkce `text_field` funkci `text_field_tag`. Ta má parametry

text_field_tag(*`name`*, *`content`* = nil, *`options`* = {})

Parametr *`name`* je název pod kterým hodnotu zadanou výslednou hodnotu najdeme v poli `@params`. Druhý parametr *`content`* udává co se v textovém poli zobrazí jako implicitní hodnota. Nejlépe je zde použít hodnotu `@params[:`*`name`*`]`. Posledním parametrem je hash *`options`* Do něj ukládáme volby/attributy vytvářeného textového pole jako jsou například `:size`, ... **FIXME:**doplnit.

```
náklad: <%= text_field_tag :naklad, @params[:naklad] %>
```

V pohledu vykreslíme pole

```
<%= start_form_tag :action => 'dotisk', :id => ... %>
    K dotisku zadáváme
    <%= text_field_tag :naklad, @params[:naklad] %>
    kusů.
    <%= submit_tag %>
<%= end_form_tag %>
```

V požadované metodě `dotisk` se pak na hodnotu pole `naklad` odkážeme

```
def dotisk
        naklad = params[:naklad]
 ⋮
    end
```

47.15.2. Vstupní formulář

Vstupní formulář předvedu na konkrétním příkladu datového modelu `Person` (pl. people). Jedná se o tabulku lidí kteří jsou například našimi zákazníky nebo zaměstnanci.

Nejdříve definujeme metody `new` a `create` v řadiči `PeopleController` (`app/controller/people_controller.rb`):

```
class PeopleController < ApplicationController
    ...
    def new
```

```
            @person = Person.new()
        end

        def create
            @person = Person.new(params[:person])
            if @person.save
                flash[:notice] = 'Person was succesfully created.'
                redirect_to :action => 'list'
            else
                render :action => 'new'
            end
        end
        ...
    end
```

Nyní si ukážeme jak vypadá pohled new který uvedené metody používají. Pro snadnější použití mám tento pohled strukturován do dvou souborů. V prvním souboru `app/views/people/new.rhtml` je:

```
<h1>Nový člověk</h1>
<% form_for :person, @person, :url => { :action => 'create' } do |f| %>
  <%= render :partial => 'form', :locals => { :f => f } %>
  <%= submit_tag "Zapsat" %>
<% end %>
<%= link_to 'Zpět', :action => 'list' %>
```

Vlastní pole formuláře jsou uložena v samostatném soubru `app/views/people/_form.rhtm`.

```
<%= error_messages_for 'person' %>
<!--[form:person]-->
<table>
  <tr>
    <th><label for="person_firstname">Jméno</label></th>
    <td><%= f.text_field :firstname %></td>
  </tr>
  ...
</table>
<!--[eoform:person]-->
```

Nejdříve starý přístup. Máme datový model Person (pl. people) a chceme vytvořit formulář pro zadávání nového člověka. V řadiči People Controller (`app/controller/people_controller.rb`) vytvoříme dvě metody, `new` a `create`.

K těmto metodám definujeme pohled rozdělený do dvou částí. První část je v souboru

```
<h1>Nový člověk</h1>
<% form_tag :action => 'create' do %>
  <% render :partial => 'form' %>
  <% submit_tag "Vytvořit" %>
<% end %>
<%= link_to 'Zpět', :action => 'list' %gt;
```

Druhá část, tedy samotná vstupní pole jsou v samostatném souboru `app/views/people/_form.rhtml`

```
<%= error_messages_for 'person' %>

<table>
  <tr>
    <th><label for="person_pin">Osobní číslo:</label></th>
```

```
    <td><%= text_field 'person', 'pin' %></td>
  </tr>
...
</table>
```

47.15.2.1. Vstupní a editační formulář

Pohled

```
<% if params[:id].blank? -%>
    <h1>Nový uživatel</h1>

    <% form_tag(:action => 'create') do %>
        <%= render :partial => 'form' %>
        <%= submit_tag 'Create' %>
    <% end %>
< else -%>
    <h1>Změny v uživateli</h1>

    <% form_tag :action => 'update', :id => @person do %>
        <%= render :partial => 'form' %>
        <%= submit_tag 'Edit' %>
    <% end %>
    <%= link_to('Destroy User', { :action => 'manage', :id => @person },
                  :confirm => 'Jste si jist že chcet odstranit trvale tohoto uživatele?',
                  :method => :post) %>
<% end -%>
<%= link_to 'Zpět', :action => 'list' %>
```

47.15.3. Roletky (*Selection Lists*)

select(:variable, :attribute, choices, options, html_options)

`choices` obsahuje položky seznamu. Může to být jakýkoliv *enumerovatelný* typ jako jsou Array, Hash a výsledek databázového dotazu.

```
<%= select(:format, @param[:format], %w{A1 A2 A3 A4 A5}) %>
```

Při zobrazení se nám objeví roletkové menu s možnostmi A1, A2, ... A5.

Jednotlivé položky pole voleb mohou být buď to přímo hodnoty, nebo objekty jenž rozumí zprávám `first` a `last`. Objekty takového druhu jsou například pole.

V prvním případě, tedy jsou-li jednotlivé prvky přímo hodnotami, máme při zpracování v řadiči k dispozici pravě jednu z těchto hodnot, kterou uživatel vybral.

Při užití druhé možnost, tedy že jednotlivé prvky jsou objekty, se výstupní html kód vygeneruje tak, že metodou `first` se získá text jenž se má zobrazit a metodou `last` jeho identifikátor. V řadiči pak máme k dispozici identifikátor volby kterou uživatel zvolil.

47.15.4. Roletky z databáze

Když editujeme tabulku, v jejímž sloupci jsou indexová čísla do slovníku (číselníku), použijeme místo textového pole roletku ktero budeme plnit z navázané databáze. Nejjednodušší postup je vytvořit si pole a a to předat funkci `select`.

```
<%=
    @moznosti = Format.find(:all, order => 'name').map{|e| [e.name, e.id]}
    select :format, @params[:format], @moznosti
%>
```

V prvním řádku si vytvoříme pole s jednotlivými volbami. To získáme tak, že si z tabulky `formaty` vybereme všechny záznamy a poskládáme pole možností tak že každý prvek je dvouprvkévé pole obsahující v prvním části text jenž se zobrazí `e.name` a v druhé části `e.id`. Toto pole voleb pak předáme funkci `select`.

Protože je toto poměrně časté, existuje funkce `collection_select`

collection_select

```
<%=
    @volby = Format.find :all, :order => 'nazev'
    collection_select :format, @params[:format], @volby, :id, :name
%>
```

FIXME:

Pokud položky seznamu k zobrazení *`choices`* rozumí metodám `first` a `last`, použije funkce `select` první hodnotu `first` pro zobrazení a poslední `last` jako klíč.

```
<%=
    @formaty = Format.find(:all, :order => 'nazev').map {|f| [f.nazev, f.id}
    select :format, @params[:format], @formaty
%>

<%= select 'project', 'funding_source_id', FundingSource.find_all.collect {|f| [ f.name, f.id

<% form_for :list, @list, :url => {:action => 'create'} do |f|%>
    ...
    <%= f.select :stredisko_id, Stredisko.find(:all, :order => 'symbol').collect{|s| ["%03d |
    ...
<% end %>
```

47.15.5. Číselník v modelu

Pokud je váber položek pro roletku konstantní, nemusíme je načítat z databáze. Potom je možné mít možné volby popsány v modelu a použit tento. V modelu tedy definuji metodu vracející pole voleb. V prvním sloupci je text k zobrazení v druhém hodnota pro databázi.

```
class Person < ActiveRecord::Base
    def self.volby_typu
        [
            ['Tiskař', 1],
            ['Strojník', 2],
            ['Obchodník', 3]
        ]
    end
end
```

V řadiči aplikace není nic neobvyklého:

```
class PersonController < ApplicationController
    ...
    def new
        @person = Person.new
    end
    ...
end
```

V pohledu pak použijeme metodu definovanou v modelu jenž vrací pole voleb.

```
<% form_for :person, @person, :url => {:action => 'create'} do |f|%>
    ...
    <%= f.select :typ, Person.volby_typu} %>
    ...
<% end %>
```

47.15.6. Zobrazení dat v tabulkové formě

Tabulka je přirozený způsob jak reprezentovat informace které jsou umístněny v pravidelné mřížce tabulky. O tom jak prezentovat data pomocí tabulek si ukážeme v následujících stránkách.

* ***WORKING: Editovat.***

Někdy potřebujeme zobrazit data z databáze ve formě tabulky, protože je to přirozený způsob jejich prezentace. Něž přistoupíme k samotnému vytváření pohledu, popíšeme si jak bude tabulka vypadat v XHTML 1.1..

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1/EN" " http://www.w3.org/TR/xhtml11/DTD/xhtml11.dt
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
    ⋮
    ⋮
</html>
```

Nyní, když víme jaký html kód budeme vytvářet, můžeme přistoupit k psaní pohledu. Pohled rozdělíme do dvou souborů. V prvním což je `list.rhtml` je pospána samotná tabulka a její hlavička. V druhém, vnořeném pohledu jsou popsány řádky tabulky. Nejdříve tabulka a její hlavička v souboru `app/views/`*`tabulka`*`/list.rhtml`:

```
<h1>Název tabulky</h1>
<table class="data" rules="all">
  <caption>Název tabulky</caption>
  <thead>
    <tr class="head">
      <th>#</th>
      <th>sloupec</th>
      ⋮ titulky k jednotlivým sloupcům v tazích <th>
    </tr>
  </thead>
  <tbody>
    <%= render_collection_of_partials 'table_row', @tabulka %>
  </tbody>
</table>
```

V uvedeném vzoru jsem použil jak popis před tabulkou v tagu `h1`, tak i popis v tabulce pomocí tagu `caption`. V značkách `table` a `tr` definuji `class`, jenž používám v css k úpravě designu tabulky. Samotné řádka tabulky, její tělo, nechám na zobrazení vloženému pohledu `table_row`. Tento je uložen v souboru `app/views/`*`tabulka`*`/_table_row.rhtml`

```
<%# table_row_counter začíná počítat řádky od 0.  My je logicky číslujeme od 1.  Proto ten výr
<tr class="<%= table_row_counter.+(1).modulo(2).nonzero? ? 'odd' : 'even' -%>">
  <td><$= table_row.id -></td>
  <td><$= table_row.sloupec -></td>
  ⋮ zobrazení dat ostatních sloupců v tazích <td>
  <%# Následuje definice příkazů na řádku: >
  <td><%= link_to 'Ukaž', :action => 'show', :id => table_row %></td>
  <td><%= link_to 'Odstraň', { :action => 'destroy', :id => table_row }, :confirm => 'Jste si
</tr>
```

Na uvedené předloze je zajímavý způsob definice *`class`* pro rozlišení lichých a sudých řádků. Protože počitadlo `table_row_counter` počítá řádky od nuly, a my chceme aby číslování počínalo jedničkou. Tedy první řádek tabulky je řádek lichý, zvětšíme pro výpočet lichost/sudosti toto počitadlo o jedničku. Zvolil jsem raději tento zápis než kratší a více kryptické prohození řetězců "odd" a "even".

V Rails existuje elegantnější zápis pro proužkování než předchozí. Využívá se při nem pomocná metoda `cycle`. Uvedený řádek s `tr` elementem bude vypadat za použití této pomocné metody následovně:

```
<tr class="<%= cycle('odd', 'even')%>">
```

K uvedeným šablonám ještě patří část css souboru jenž nám tabulku obarví. Efekt kterého chci dosáhnout je ten že liché a sudé řádky mají jinou barvu podkladu a vytvářejí tak proužky. Pro odlišení ještě zvolím jinou barvu podkladu pro první řádek tabulky s nadpisy slpoupců.

```
/*
 * Nastavení vlastností tabulek s daty.
 */
table.data tr.head { background-color: #E0E0F0; } /* Titulek: šedá do modra */
table.data tr.odd  { background-color: #FFFFE0; } /* Lichý: světle žlutý */
table.data tr.even { background-color: #E0FFE0; } /* Sudý: světle zelený */
```

* ***FIXME:****EDIT*

47.15.6.1. HTML tabulky

Tabulka je v html prezentována tagem `<table>` uvnitř kterého jsou taky `<tr>` reprezentující řádky tabulky. Na řádku jsou pak jednotlivé elementy umístněny v tazích `<td>` nebo `<th>`.

```
<table>
  <tr><th>1</th><th>2</th></tr>
  <tr><td>1</td><td>2</td></tr>
  <tr><td>2</td><td>3</td></tr>
</table>
```

V praxi jsem si oblíbil nový zápis tabulky jenž je v XHTML. Tento je podobný jen rozšířený o pár tagů.

table⟶thead|tfoot|tbody⟶tr⟶td|th

A takto vypadá ukázka zobrazení tabulky v XHTML se všemi tagy které používám.

```
<table class="třída" ... ostatní atributy>
    <caption>Titulek tabulky</caption>
```

```
    <thead><!-- Zde uvedujeden či více řádků hlavičky tabulky -->
        <tr>
            <th>A</th>
            <th>B</th>
            <th>C</th>
        </tr>
    </thead>
    <tfoot>
        <tr><!-- optimální místo pro věci zobrazované pod tabulkou jako je navigace,
                 případně boxy pro vyhledání záznamu či zadání nového záznamu --></tr>
    </tfoot>
    <tbody>
        <tr><!-- zobrazení řádků tabulky s daty -->
            <td>1</td>
            <td>2</td>
            <td>3</td>
        </tr>
        ...
    </tbody>
</table>
```

Podle tohoto XHTML vzoru napíši jednotlivé pohledy. Protože budu hojně využívat částečné zobrazení **render :partial**, nebude v mých pohledech nikdy tabulka uvedena takto v celku. Proto ji zde uvádím aby jste si mohli udržet

47.15.6.2. Přehled záznamů s metodou `index`

Na následující ukázce je vidět zobrazení seznamu strojů.

Příklad 47-4. Metoda `index`

```
def
    @stroje = Stroj.find(:all)
end
```

V případě že používáme paginátor, například will paginate, použijeme k vytvoření seznamu strojů příslušnou metodu, která se v mém případě jmenuje `search`

```
def index
    @stroje = Stroj.search(:page => params[:page], :per_page => 5)
end
```

Příklad 47-5. Pohled `app/view/.../index.html.erb`

```
<table class="list">
    <caption>Přehled strojů</caption>
    <thead>
        <tr>#<>
        ⋮
    <thead>
    <tfoot>
        ⋮
    <tfoot>
    <tbody>
        <%= render_collection_of_partials 'table_row', @stroje -%>
```

```
            ⋮
        <tbody>

    </table>
```

47.15.7. Stránkování (*Pagination*)

Odkazy

- paginate_by_sql for Rails: a database independent approach (http://thebogles.com/blog/2006/06/paginate_by_sql-for-rails-a-more-general-approach/)
- HowtoPagination (http://wiki.rubyonrails.org/rails/pages/HowtoPagination)
- PaginationHelper (http://wiki.rubyonrails.org/rails/pages/PaginationHelper)
- Ferret Pagination in Rails (http://www.igvita.com/blog/2007/02/20/ferret-pagination-in-rails/)
- Faster Pagination in Rails (http://www.igvita.com/blog/2006/09/10/faster-pagination-in-rails//)
- Cheap Pagination (http://railsify.com/plugins/18-cheap-pagination)

Stránkování, tak jak jsme jej viděli v 47.15.6 je založene na stránkovači Paginator.

```
class StrediskaController < ApplicationController
    def list

    @celkem = Stredisko.count(:conditions = > vyber)
    @strediska_pages, @strediska = paginate :strediska, :conditions => vyber, :order_by => 'sy
    end
end
```

47.15.7.1. Seznam stránek

Odkazy:

-

* ***FIXME:***

47.15.7.2. Will Paginate

Odkazy:

- Will Paginate (http://rock.errtheblog.com/will_paginate)
- will_paginate (http://railscasts.com/episodes/51)
- AJAX PAGINATION IN LESS THAN 5 MINUTES (http://railsontherun.com/2007/9/27/ajax-pagination-in-less-than-5-minutes)

Instalace:

```
$ script/plugin install will_paginate
```

Když máme plugin `will_paginate` nainstalován, můžeme přistoupit k jeho použití. V následujícím příkladu máme tabulku `listy` s datovým modelem `List`. V metodě `index` řadiče chceme vyhledat a zobrazit paginátorem seznam listů.

Poznámka: Připomínám že vztah mezi list a listy je definován v inflektoru.

Použití stránkovače (*paginator*) v příkladu rovnou protlačím z řadiče do modelu, a zobrazení provedu v tabulce.

Tak tedy nějdřív model a použití stránkovače. V třídě modelu definuji metodu `search` jenž nedělá nic jiného než že volá vlastní paginátor

Popíši rovnou řešení s stránkovačem (*pagniator*) v modelu. Ušetří nám to čas. Takže v modelu definujeme vyhledávací metodu která bude provádět stránkování. Všechny parametry budu do této metody předávat v hashi `options`. Uvnitř přidáme předané parametry k defaultním hodnotám. Tyto definovat nepotřebujeme, ale je vhodné podle dat v modelu a podle plánovaného použití si je připravit.

```
# Vyhledávání a paginátor
def self.search(options = {})
    paginate({ :per_page => 5, :order => 'id'}.merge options)
end
```

Vytvořenou metodu pro vyhledání zázanmů nyní použijeme v řadiči. Místo standardního

```
def index
    listy = List.find(:all, ...)
end
```

Použijeme novou metodu `search`. Jediným rozdílem je vypuštění symbolu `:all`, který nemá v této situaci smysl. Všechny ostatní parametry zachováme. Výsledný kód bude nyní vypadat takto:

```
def index
    listy = List.search({...}, :params => params[:page])
end
```

Zbývá nám ještě napsat pohled. V tom od začátku použiji *partials* a tak bude rozdělen do dvou souborů. Rovněž vyjdu ze zobrazení dat v tabulce. Nejdříve tedy `app/view/lisr/index.html.erb`. V tomto jsou důležité dva řádky. První vloží do pohledu navigační část paginátoru. Jedná se o volání `will_paginate` kterému předáme kolekci objektů získanou v řadiči. Druhý důležitý řádek je přesun zobrazení řádků do `table_row` pomocí `render_collection_of_partials`.

```
<table>
    <thead>
        <tr><th>#</th><th>název</th>...</tr>
    </thead>
    <tfoot>
        <tr>
            <td colspan="..."><%= will_paginate @listy%></td>
        </tr>
    </tfoot>
    <tbody>
        <%= render_collection_of_partials 'table_row', @listy%>
    </tbody>
</table>
```

Zobrazením řádku tabulky vše dokončíme. Toto je popsáno v souboru `app/view/list/_table_row.html.erb`. Zde je jen jedna „zvláštnost“, a to proužkování řádků tabulky definováním css třídy.

```
<tr class="<%= cycle('odd', 'even')%>">
    <td><%= table_row.id %></td>
    <td><%= table_row.name %></td>
    ...další sloupce tabulky
</tr>
```

V příkladu jsou zapracovány i některé další techniky, jako je například proužkování tabulky. To zajišťuje ve spolupráci s nastavením v stylesheeto definování `class` v elementu `tr`

47.15.7.2.1. Vlastní zobrazení paginátoru

- Rendering will_paginate links without previous and next buttons (http://zilkey.com/2008/3/16/rendering-will_paginate-links-without-previous-and-next-buttons)

Zobrazení paginátoru will_paginate můžeme v view řídit přes vlastní renderer. V pohledu přidáme do parametrů `:renderer`:

```
<%= will_paginate @items, :renderer => 'CustomPaginationRenderer' %>
```

Pokud používáme ten samý renderer v celé aplikaci, můžeme jej uvést v souboru `config/initializers/will_paginate.rb`:

```
WillPaginate::ViewHelpers.pagination_options[:renderer] = 'CustomPaginationRenderer'
```

Příklad 47-6. `config/initializers/will_paginate.rb`:

```
class CustomPaginationRenderer < WillPaginate::LinkRenderer
    def to_html
        links = @options[:page_links] ? windowed_links : []
 html = links.join(@options[:separator])
 @options[:container] ? @template.content_tag(:div, html, html_attributes) : html
    end
end
```

47.15.8. Formulář se dvěma modely

Odkazy:

- Creating Two Models in One Form (http://www.railsforum.com/viewtopic.php?id=717)

Příklad 47-7. Metoda new v souboru řadiče `projects_controller`

```
# controller.rb
def new
    @project = Project.new
    @task = Task.new
end
```

Příklad 47-8. projects/new.rhtml

```
<h1>Nový projekt</h1>

<%= error_messages_for :project %>
<%= error_messages_for :task %>

<%= start_form_tag :action => 'create' %>
<p>
    Název projektu:
    <%= text_field :project, :name %>
</p>
<p>
    První úkol:
    <%= text_field :task, :name %>
</p>
<p>
    <= submit_tak 'Vytvořit' %<
</p>
<%= end_form_tag %>
```

Příklad 47-9. Metoda `create` v řadiči `projects_controller.rb`

```
def create
    @project = Project.new/params[:project])
    @task = @project.tasks.build(params[:task])
    if @project.save
        redirect_to :action => 'index'
    else
        render :action => 'new'
    end
end
```

47.15.9. Hlavička a položky

Odkazy:

•

47.15.9.1.

Odkazy:

- Creating Variable number of Models in One Form (http://railsforum.com/viewtopic.php?id=1065)

Příklad 47-10. projects_controller.rb

```
def new
    @project = Project.new
    @project.tasks.build # vytvoří jeden nový úkol
end


def create
```

```
    @project = Project.new(params[:project])
    params[:tasks].each_value { |task| @project.tasks.build(task) }
    if @project.save
        redirect_to :action => 'new'
    else
        render :action => 'new'
    end
end


def add_task
    @task = Task.new
end
```

Příklad 47-11. projects/new.rhtml

```
<% form_for :project, :url => { :action => 'create' } do |f| %>
    <p>Name: <%= f.text_field :name %></p>
    <h2>Tasks</h2>
    <div id="tasks">
        <% @project.tasks.each_with_index do |task, index| %>
            <%= render :partial => 'task_fields', :locals => { :task => task, :index => index
        <% end %>
    </div>
    <%= render :partial => 'add_task_link', :locals => { :index => @project.tasks.size } %>
    <p><%= submit_tag 'Create Project' %></p>
<% end %>
```

Příklad 47-12. projects/_task_fields.rhtml

```
<div id="task_<%= index %>">
    <% fileds_for "tasks[#{index}]", task do |f| %>
        <p><%= f.text_field :name %>
        <%= link_to_remote 'remove', :url => { :action => 'remove_task', :index => index } %>
        </p>
    <% end %>
</div>
```

Příklad 47-13. projects/add_task.rjs

```
page.insert_html :bottom, :tasks, :partial => 'task_fields', :locals => { :task => @task, :inc
```

47.15.10. TableKit

Odkazy:

- TableKit (http://www.millstream.com.au/view/code/tablekit/) od Millstream web software
- Editable table with Javascript, TableKit, AJAX and Rails (http://talkingcode.co.uk/2007/12/10/editable-table-with-javascript-tablekit-ajax-and-rails/)

47.16. Layout

Layout je rozvržení stránky. V tomto je zahrnuto rozložení jednotlivých komponent a hlavního obsahu.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
    <head>
        <title>Simple Web: <%= @page_title ||= 'Staff Area' %></title>
        <meta http-equiv="content-type" content="text/html;charset=utf-8" />
        <meta name="author" content="Radek Hnilica" />
        <meta name="copyright" content="Copyright 2008, Radek Hnilica" />
        <%= stylesheet_link_tag('myweb', 'main', :media => 'all') %>
    </head>
    <body class="public">
        <div id="header">
            <h1><%= link_to('MyWeb', '/', :style => 'text-decoration: none; color: #DDDDDD;')
        </div>
        <div id="pagecontent">
            <h1><%= @page_title %></h1>
            <%= %Q(<div class="notice">#{flash[:notice]}</div>) if flash[:notice] %>
            <%= yield %>
        </div>
        <div id="footer">
            <p>Copyright 2008, Radek Hnilica</p>
        </div>
    </body>
</html>
```

V každém řadiči můžeme specifikovat v jakém layout se budou jeho stránky zobrazovat.

```
class PeopleController < ApplicationController
    layout "staff"

    def index
    end

    def login
    end
    ⋮
end
```

47.17. Routing

Směrování url dotazu na objekty

Konfigurace routingu je uvedena v samostatném konfiguračním souboru `config/routes.rb`. V tomto souboru v definici třídy `ActionController::Routing::Routes.draw` jsou zapsány všechny mapování url na objekty aplikace. Pomocí příkazu **map.connect** se spojí dané url s objektem. Na pořadí příkazů záleží. Při porovnávání url se procházejí mapování po řadě a první které vyhovuje se použije. Tedy pořadí mapování je nejdříve specifická a pak obecná. Poslední mapování jsou proto standardní obecná mapování:

```
# Install the default route as the lowest priority.
map.connect ':controller/:action/:id.:format'
map.connect ':controller/:action/:id'
```

```
end
```

* *Přijít s vlastním a lepším příkladem!*

```
map.connect 'recipes_for/:ingredient', :controller => 'recipes', :action => 'show'
```

První argument příkazu `recipes_for/:ingredient` je url, další parametry vytvářejí spojení na objekt. V našem příkladu se odkaz na url převede na akci 'show' řadiče 'recipes'. Součástí url moho být divoké karty jako je zde `:ingredient` řetězec který bude v url na místě `:ingredient` bude předán do aplikace jako parametr.

Mapování ovšem funguje taky obráceným směrem. Kdykoliv konstruujeme url například pomocí `link_to` projde program jednotlivá mapování a sestaví url. Pro naše mapování třeba

```
<= link_to "Recipe for apples", :controller => 'recipes', :action => 'show', :ingredient => 'a
```

47.17.1. Formát

Standardně jsou poskytovány html stránky. Ale máme možnost specifikovat několika způsoby jiný než html formát, například xml. Pro zadání formátu můžeme použít buď'to url, kdy například požádáme o `http://example.net/items/show/3.xml`. O formát můžeme také požádat v hlavičce

```
$ wget http://example.net/items/show/3 -O - --header="Accept: text/xml"
```

Odpovídající kód v řadiči Item je

```
def show
    @item = Item.find(params[:id])
    respond_to do |format|
        format.html
        format.xml { render :xml => @item.to_xml}
    end
end
```

47.17.2. Prázdné url

Pro svou aplikaci můžeme provést mapování prázdného url. Je třeba ovšem odstranit soubor `public/index.html` a zadat mapování. Například můžeme přesměrovat prázdné url na řadič `wellcome`.

```
map.connect '', :controller => 'wellcome'
```

Další ukázky přesměrování prázdného url.

```
map.connect '', :controller => 'main', :action => 'wellcome'

map.connect '', :controller => 'top', :action => 'login'

map.connect '', :controller => 'main'
```

47.17.3. Pojmenované routy

Pojmenovanou routu vytvoříme jednoduše tak, že místo metody `connect` napíšeme jméno kterým routu označíme.

```
map.help 'help', :controller => 'main', :action => 'show_help'
```

Rails nám vytvoří k takové pojmenované routě několik metod jejichž jména začínají názvem routy.

_path

```
<%= link_to "Pomoc!", help_path %>
```

_url

Metoda vytváří celé url.

47.18. Extrémní programování v Rails (XP)

- HowtosTesting (http://wiki.rubyonrails.com/rails/pages/HowtosTesting)
- 15 TDD steps to create a Rails application (http://andrzejonsoftware.blogspot.com/2007/05/15-tdd-steps-to-create-rails.html) 2007-05-08
- . ()

RoR je již připraven pro použití technik extrémního programování. Při vytváření projektu se automaticky vytvoří potřebné adresáře a soubory.

Jedná se o adresáře:

`test/fixtures`

FIXME:

`test/functional`

FIXME:

`test/mocks`

FIXME:

`test/unit`

FIXME:

V `Rakefile` jsou pak implementovány dva cíle

- `test_units` — spuští testy datových modelů
- `test_functional` — spouští testy funkčnosti

```
# test_helper.rb

# gem install fakeweb
require 'fakeweb'

# gem install test_timer
require 'test_timer'

class Test::Unit::TestCase
```

```
...
```

47.18.1. Testovací metody/funkce

Odkazy:

- 5. Hey Test/Unit. Assert This! (http://manuals.rubyonrails.com/read/chapter/24)
- 35.4.4

47.18.2. Testování datového modelu

Datový model testujeme unit testy. Pro každý datový model máme jeden soubor s testy v adresáři `test/unit/` jenž je pojmenován podle vzoru

*model*_test.rb

Používáme také „fixtures“, datové záznamy definované v textovém souboru. Ten se nachází v adresáři `test/fixtures/` a jemnuje se po datové tabulce do níž se záznamy v něm definované zapisují.

Aby se nám dále snadno pracovalo, uvedeme si konkrétní příklad. Mějme tabulku jenž je definována/vytvářena sql příkazem

```
CREATE TABLE hosts (
        id      SERIAL PRIMARY KEY,
        name    VARCHAR(24) UNIQUE --natural primary key
);
```

Pro tuto tabulku `hosts` máme vytvořen model `Host`. Připravíme si tedy nejdříve „fixtures“ v souboru pojmenovaném po této tabulce, tedy `test/fixtures/hosts.yml`. Původní, vygenerovaný, obsah nahradíme svými dvěma pojmenovanými záznamy.

```
sunrise:
  id: 1
  name: sunrise

yoda:
  id: 2
  name: yoda
```

v našm příkladě mi pěkně vyšlo, že záznamy pojmenovávám stejným jménem které je v sloupečku/poli name. Jméno záznamu nemá jiný význam, než že je pod ním daný záznam přístupný v době testování. Mohl uvedené fixtures napsat také takto

```
prvni_stroj:
  id: 1
  name: sunrise

dalsi_stroj:
  id: 2
  name: yoda
```

Fixtures máme hotové, a můžeme přistoupit k testování. V souboru `test/unit/host_test.rb` máme po vytvoření jednoduchý, prázdný, test `test_truth`. Tento můžeme odstranit, protože uvedeme vlastní testy. Jako první otestujeme naše fixtures, zdali je máme ke dispozici.

```
# Are all fixtures presented?
def test_fixtures_presence
    assert_kind_of Host, hosts(:sunrise)
    assert_kind_of Host, hosts(:yoda)
end
```

Testujeme zdali fixtures existují a jsou daného typu, vyhovují našemu modelu. Po napsání tohoto prvního testu jej již můžeme spustit. Použijeme příkaz

```
$ rake test:units
```

Testy proběhnou bez chyby. V této chvíli jsme se dopustili malého prohřešku na metodologii extreémního programování, přesněji části TDD (Test Driven Development). Měli jsme nejdříve napsat test `test_fixtures_presence` a teprve po neúspěšných testech upravit soubor s fixtures.

Nyní přistoupíme k testování základních operací nad modelem. Jedná se o operace Create, Read, Update a Delete. Někdy jsou dohromady nazývány také CRUD. Náš test se tedy bude jmenovat `test_crud`. Napíšeme si tedy jeho kostru, prázdnou metodu s komentáři na jejichž místo budeme psát kód.

```
def test_crud
    # Create
    # Read
    # Update
    # Delete
end
```

Nejdříve tedy vytvoření záznamu. V prvním příkazu si vytvoříme nový záznam jež má v poli `name` řetězec `joshua` a v druhém řádku jej uložíme do databáze.

```
    # Create
    joshua = Host.new(:name => 'joshua')
    assert joshua.save
```

Po dopsání uvedených dvou řádků můžeme, a taky to doporučuji, opět spustit testy. Měly by proběhnout bez chybičky. Pokud nějaká nastala, máme problém k řešení. V tomto okamžiku mě napadá že by mohl nastat problém zápisu do databáze, například pro nedostatečná oprávnění, případně z tuctu dalších důvodů. Protože ovšem náš test uspěl, přikročíme k napsání dalšího. Budeme testovat čtění. V prvním příkazu načítáme z tabulky náš záznam, identifikovaný číslem `id`. Využijeme toho, že v objektu `jsohua` máme zapsáno jeho `id`. V následujících dvou řádcích srovnáváme obsah pole `name` s hodnotou jenž očekáváme.

```
    # Read
    host = Host.find(joshua.id)
    assert_equal joshua.name, host.name
    assert_equal 'joshua', host.name
```

Spustíme testy. Opět žádná chybička. Můžeme tedy napsat test pro změnu záznamu. V naší tabulce máme jen jeden sloupec s daty, takže skusíme u již nalezeného stroje `joshua` jenž je uložen v proměnné `host` změnit jeho jméno například na `hola`. To provedeme prvním příkazem. Ve druhém ttestujeme že zápis proběhl bez problémů.

```
    # Update
    host.name = 'hola'
    assert host.save
```

Pokud proběhl test bez problémů, můžeme napsat v této metodě poslední test. Test mazání záznamu. Využijeme toho že máme od předchozích testů v proměnné `host` vytvořený záznam a ten smažeme.

```
    # Delete
```

```
        assert host.destroy
```

Pokud i teď proběhnou všechny testy bez problému máme otestovánu a zaručenu základní funkcionalitu datového modelu. Před tím než se budeme věnovat další činnosti, je vhodné zaslat všechny změny do subversion repositáře.

47.18.2.1. fixtures

Fixtures jsou vzorky dat, datových záznamů, které jsou nám k dispozici v průběhu testování. Můžeme je specifikovat několika způsoby.

Jako YAML soubor. V této variantě jsou všechny záznamy zapsány v YAML souboru v `test/fixtures/tabulka.yml`. Každý záznam má formu:

```
jina:
    id: 2
    skupina: ostatni
    nazev: Jiná společnost, s.r.o
    jednatel: Vašek Jednák
    ulice: MyString
    mesto: Brno
    psc: 01234
    ico: 01234568
```

První řádek je symbolický název záznamu pomocí kterého se na záznam můžeme odkazovat. Dále následují jednotlivá pole s hodnotami. Pole `id` nemusíme definovat, bude vyplněnou automaticky.

Jako CSV soubor. Dalším způsobem zápisu je csv soubor. Ten dovoluje „hustější“ zápis kdy každý řádek definuje jeden záznam.

```
id, skupina, nazev, jednatel
1, ostatni, "Jiná společnost, s.r.o.", Vašek Jednák
2, tun, "Tuneláři, a.s.", Tomáš Razič
3, ostatni, "Kamarád & spol", Bořek Stavča
```

Na prvním řádku jsou názvy polí (sloupců) v souboru definovaných. Na dalších řádcích pak jednotlivé záznamy s poli přesně v pořadí podle prvního řádku.

S použitím CSV souborů jsou spojeny všechny ty zvláštnosti CSV zápisu jako jsou:

- pokud obsahuje pole čárku ’,’, musí být ohraničeno uvozovkami ’"’
- pokud obsahuje pole uvozovky ’"’ musí být zdvojeny ’""’
- nesmíte nechat prázdné řádky
- hodnota nil se „zapisuje“ tím, že se na příslušné místo nezapíše nic (v textu se potom nachází dvě čárky po sobě)

Fixtures se automaticky nepoužijí. Pokud je chceme použít můsíme tak učinit sami příkazem **fixtures**. V následující ukázce je tento přikaz použit rovněž s testem který ověří že se načetly opravdu všechy záznamy. V našem případě jsou v souboru `test/fixtures/spolecnosti.yml` dva záznamy.

```
class SpolecnostTest < ActiveSupport::TestCase
    fixtures :spolecnosti
    def test_spolecnost_fixtures
        assert_equal 2, Spolecnost.count
    end
    ...
```

```
end
```

47.18.2.2. Pomocné testovací metody

Protože jsou některé testy rozsáhlé, používám několik metod které mi je umožní zjednodušit. Jsou to v prvé řadě dvě metody jenž vytvářejí dva korektní záznamy daného modelu. Tyto záznamy vyhovují všem stanoveným omezením. Jména metod které jsem jim přidělil jsou `create_valid1` a `create_valid2`. Jako parametr akceptují hash kterým modifikují pole záznamu.

```
private

  def create_valid1(options={})
    Spolecnost.create({
                        :nazev => 'První správná, s.r.o.',
                        :skupina => 'cl' # !!! nesmí být symbol :cl
                      }.merge(options))
  end

  def create_valid2(options={})
    Spolecnost.create({
                        :nazev => 'Druhá správná, a.s.',
                        :skupina => 'ostatni'
                      }.merge(options))
  end
```

A samozřejmě že si ověříme že taky všem omezením vyhovují.

```
  def test_create_valid_1_and_2
    assert create_valid1().save
    assert create_valid2().save
  end
```

Protože když nevyhovují, znamená to, že jsme při nějakém dalším vylepšení modelu zapoměli opravy obě metody.

Protože předpokládám že budu metodu `create_valid1` používat velmi často v případech kdy mi stačí jen jeden záznam, z čistě estetických důvodů si na ni zavedu alias.

```
  alias :create_valid :create_valid1
```

47.18.2.3. Testování validátorů

* *Jak otestovat spravnou funkci jednotlivých validátorů v modelu?*

Odkazy:

- Rails Testing: Not Just for the Paranoid page 2 (http://www.oreillynet.com/pub/a/ruby/2007/06/07/rails-testing-not-just-for-the-paranoid.html?page=2)
- Nice helper for testing model validation (http://yarorb.wordpress.com/2007/12/03/nice-helper-for-testing-models/)
- DB2 and Ruby on Rails, Part 3: Testing with DB2 and Ruby on Rails (http://www.ibm.com/developerworks/db2/library/techarticle/dm-0706chun2/)

FIXME:

47.18.2.3.1. Testování `validates_uniqueness_of`

Odkazy:

- 47.11.4.2

Testování jednoduché formy validátoru

```
class Spolecnost < ActiveRecord::Base
    validates_uniqueness_of :ico
end
```

Příklad 47-14. `test/unit/model_test.rb`:

```
class SpolecnostTest < ActiveSupport::TestCase
    fixtures :spolecnosti

    def_test_uniquness_of_ico
        cl = spolecnosti(:cl)
 spolecnost = create_valid(:ico => cl.ico)   # new record with same ico as cl
        assert spolecnost.errors.invalid?(:ico)
    end

end
```

* ***FIXME:*** *vyřešit* ***assert !****.*

V příkladu jsou využity metody které jsem popsal v 47.18.2.2.

Pokud jsou přípustné prázdné či nil hodnoty, připíšeme další testy.

```
    def test_uniqueness_of_ico_nil
        assert create_valid1(:ico => nil).save
        valid = create_valid2(:ico => nil)
        assert ! valid.errors.invalid?(:ico)
    end

    def test_uniqueness_of_ico_blank
        assert create_valid1(:ico => ' ').save
        valid = create_valid2(:ico => ' ')
        assert ! valid.errors.invalid?(:ico)
    end
```

47.18.3. Testování řadiče

```
def test_should_get_index
    get :index
    assert_response :success

    assert_select "tag", value, "message"
end
```

47.18.4. ZenTest

Odkazy:

- RDoc dokumentace k ZenTest (http://zentest.rubyforge.org/ZenTest/)
- How to Use ZenTest with Ruby (http://www.linuxjournal.com/article/7776) on Linux Journal

```
# gem install ZenTest
# gem install redgreen
.autotest:
require 'autotest/redgreen'
#require 'autotest/pretty'
#require 'autotest/snarl'
#require 'autotest/timestamp'
```

47.18.4.1. Autotest

Odkazy:

- Integrace autotestu a Emacsu (http://www.emacswiki.org/cgi-bin/wiki/RyanDavis)
- Getting started with Autotest - Continuous Testing (http://ph7spot.com/articles/getting_started_with_autotest)

Nástroj Autotest slouží k automatickému spouštění testů. Pracuje se s ním tak, že v dalším terminálu se přepneme do naší aplikace a spustíme autotest. Ten spustí všechny testy a čeká až změníme nějaký soubor. Pak spustí testy znovu.

V adresáři aplikace můžemem mít soubor `.autotest` ze kterého si autotest čte nastavení. Uvádím příklad.

Příklad 47-15. Příklad souboru `.autotest`

```
require 'autotest/redgreen'     obarvuje výstup, vřele doporučuji
# Desktop notification
#require 'autotest/growl'       # OS X
#require 'autotest/snarl'       # Win32, obdoba Growl na OSX
#require 'autotest/kdenotifiy'  # KDE
# Reporting
#require 'autotest/pretty'      # Mac OS X / RubyCocoa
#require 'autotest/html_report'
# Other plugins
#require 'autotest/timestamp'
```

Soubor `.autotest` je regulární ruby soubor a může proto obsahovat i ruby konstrukce, nejen příkaz **require**.

Program **autotest** akceptuje několik přepínačů.

`-h`
`-help`

Vypíše krátkou nápovědu k volbám programu **autotest**

`-v`

Program je více upovídaný.

`-q`

Opak volby `-v` program nevypisuje žádné zbytečnosti. Velmi užitečná volba která usnadní orientaci v tom co program píše na terminál.

`-f`

Rychlý start. Program na začátku nespouští žádné testy.

> **Varování**
>
> Pozor, volba `-help` má na začátku jen jednu pomlčku.

```
$ autotest -q
```

Poznámka: Nejdřive jsem si myslel že je autotest úplná zbytečnost. Taková hračka jen na efekt. Tak jsem ho na ten efekt chtěl zkusit. Nyní zjištuji, že bych chtěl počítač se třema monitory. Na jednom pracuji s editorem na druhém bych chtěl mít v prohlížeči zobrazenou stránku na které pracuji a na tom třetím bych chtěl mít spuštěný autotest.

47.19. CIA — Continuous Integration Automater

* *section id="cia" xreflabel="CIA"*

Odkazy:

- Continuous Integration (http://www.martinfowler.com/articles/continuousIntegration.html) by Martin Fowler and Matthew Foemmel
- Continuous Integration (http://blog.innerewut.de/articles/2005/05/12/continuous-integration) by Jonathan Weiss
- Setting up CIA with Rails and Subversion (http://blog.innerewut.de/articles/2005/09/18/setting-up-cia-with-rails-and-subversion) by Jonathan Weiss
- **FIXME:** ()

CIA je k dispozici přes svn na adrese `http://dev.rubyonrails.com/browser/tools/cia/trunk/`

```
yoda:~# aptitude install sqlite libsqlite-dev
yoda:~# gem install sqlite

# cd /var/www
# svn co http://dev.rubyonrails.org/svn/rails/tools/cia/trunk cia
...
Checked out revision 2357.
# sqlite db/cia.sqlite # creates and loads the database
SQLite version 2.8.16
Enter ".help" for instructions
sqlite> .read db/sqlite.sql
sqlite> .quit
```

Rychlá zkouška

```
yoda:/var/www/cia# ruby script/server -e production
```

```
radek@yoda:/var/www/cia:$ psql -f db/postgresql.sql template1
```

47.20. Správa verzí

47.20.1. Subversion

FIXME:

47.20.2. Git

Odkazy:

- Really Simple Git Deployment with Vlad (http://glu.ttono.us/)
- A Three Finger Salute to Git (http://nubyonrails.com/)

FIXME:

47.21. Použití Subversion

* *section id="rails.subversion" xreflabel="Použití Subversion"*

- Subversion Techniques (http://developer.r-project.org/SVNtips.html)
- Tho Top Ten Subversion Tips for CVS Users (http://www.onlamp.com/pub/a/onlamp/2004/08/19/subversiontips.html)

Ukázka postupu vytvoření projektu `snimkypd`.

```
# mkdir /var/lib/svn/snimkypd
# chown radek:radek /var/lib/svn/snimkypd

$ svnadmin create /var/lib/svn/snimkypd
$ rails /tmp/snimkypd/trunk
$ mkdir /tmp/snimkypd/branches /tmp/snimkypd/tags
$ svn import /tmp/snimkypd file:///var/lib/svn/snimkypd -m "initial import"

...
Committed revision 1.

$ cd ~/src/firma/mpress
$ svn checkout file:///var/lib/svn/snimkypd/trunk snimkypd
...
Checked out revision 1.
```

Tím máme první revizi prázdného projektu v repository a současně k dispozici v pracovní verzi v adresáři `~/src/firma/mpress/snimkypd`. Protože používám CIA, skopíroval a upravil jsem soubor `/var/lib/svn/snimkypd/hooks/post-commit`.

FIXME:

FIXME:doplnit zkontrolovat.

```
This is the top of the Subversion repository.

   trunk/ ......... The latest development sources.  When people say
                    "Get the head of trunk", they mean the latest
                    revision of this directory tree.

   branches/ ...... Various development branches.  Typically a branch
                    contains a complete copy of trunk/, even if the
                    changes are isolated to one subdirectory.  Note
                    that branch copies are generally removed after
                    they've been merged back into trunk, so what's in
                    branches/ now does not reflect all the branches
                    that have ever been created.

   tags/ .......... Snapshots of releases.  As a general policy, we
                    don't change these after they're created; if
                    something needs to change, we move it to
                    branches and work on it there.

   developer-resources/
                    Miscellaneous things that might (or might not) be
                    of interest to developers.
```

47.21.1. Založení nového projektu v Subversion

* *section id="rails.svn.create-project" xreflabel="Založení nového projektu"*

- Checkign a new Rails application into an existing Subversion repository (http://wincent.com/knowledge-base/Checking_a_new_Rails_application_into_an_existing_Subversion_repository)

* ***FIXME:****přepsat.*

Při zakládání nového projektu v Subversion je třeba dodržet několik základních pravidel a doporučených postupů. Pokud se jimi budeme řídit, můžeme pak snadno realizovat jednotlivé větvě.

Nejdříve uvedu několik předpokladů ze kterých vychází následující postup. Svn není realizováno svn serverem ale používám lokální souborový přístup. Adresář v kterém jsou uloženy jednotlivé projekty se jmenuje `/var/lib/svn`. Zkušební projekt se kterým budu pracovat se jmenuje `videoteka`. Můj účet na počítači se jmenuje `radek`.

Takže začneme vytvořením projektu v Subversion

```
# mkdir -p /var/lib/svn/videoteka
# chown radek:radek /var/lib/svn/videoteka
$ svnadmin create /var/lib/svn/videoteka
```

Vytvoříme si někde v pracovním prostoru adresářovou strukturu.

```
$ mkdir -p /tmp/videoteka/trunk
$ mkdir /tmp/videoteka/tags
$ mkdir /tmp/videoteka/branches
```

Vytvořenou prázdnou strukturu můžeme předat svn.

```
$ svn import -m "Vytvoření prázdného projektu videoteka" /tmp/videoteka file:///var/lib/svn/vi
Adding         /tmp/videoteka/trunk
Adding         /tmp/videoteka/branches
Adding         /tmp/videoteka/tags
```

```
Committed revision 1.
```

Nyní si do pracovního prostoru kde pracujeme se zdroji,**FIXME:**, načteme nově vytvořenou prázdnou verzi a vygenerujeme rails aplikaci.

```
$ svn checkout file:///var/lib/svn/videoteka/trunk $HOME/work/videoteka
Checked out revision 1.
$ rails --svn $HOME/work/videoteka
svn: warning: '.' is not a working copy
svn: warning: '.' is not a working copy
      exists
      create  app/controllers
⋮
```

Poznámka: Pokud používáme jinou databázi než standardní sqlite3, můžem již při generování aplikace zadat `-d postgresql`. Vytvořená kostra má pak konfigurační soubor databází připravený pro námi uvedenou databázi.

V takto vytvořeném Rails projektu provedem několik úprav. Určitě nebudeme chtít aby v Subversion byly uloženy deníky.

```
$ cd $HOME/work/videoteka
$ svn remove --force log/*.log
D         log/development.log
D         log/production.log
D         log/server.log
D         log/test.log
$ svn propset svn:ignore "*.log" log/
property 'svn:ignore' set on 'log'
```

Stejným způsobem se zbavíme adresáře tmp v úložišti Subversion, a celý projekt „comitneme“ do repository.

```
$ svn remove --force tmp/*
D         tmp/cache
D         tmp/pids
D         tmp/sessions
D         tmp/sockets
$ svn propset svn:ignore "*" tmp/
property 'svn:ignore' set on 'tmp'
$ svn commit
```

Dalším souborem, který ba neměl být archivován je konfigurace přístupu k databázím. Důvodem jsou přístupová hesla k databázím jenž jsou v tomto souboru uvedena. Pro praktické řešení tedy přjmenujeme původní konfigurační soubor jenž v tuto chvíli neobsahuje hesla, tento poté upravíme aby vyhovoval námi používané konfiguraci databází s tím že v něm neuvedeme přístupová hesla a jména k databázím. To nám dovolí velmi snadno vytvořit „ostrý“ konfigurační soubor jen skopírováním a doplněním hesel a jmen.

```
$ svn move config/database.yml config/database.example
A         config/database.example
D         config/database.yml
$ svn propset svn:ignore 'database.yml' config
```

U starší verze rails nefungoval správně přepínač `--svn` při generování aplikace.

```
$ svn checkout file:///var/lib/svn/addressbook/trunk /tmp/addressbook/rev1
Checked out revision 1.
$ rails /tmp/addressbook/rev1
$ cd /tmp/addressbook/rev1
$ for f in $(svn status|grep ^?|awk '{print $2}'); do svn add $f; done
$ svn remove --force log/*.log
$ svn propset svn:ignore "*.log" log/
$ svn commit -m "Rails skeleton"
```

* ***FIXME:****zrušit.*

Protože uvedený postup nefunguje správně, používám nyní postup jiný. Nejdříve jako administrátor vytvořím adresář pro repository, a učiním sebe jako vývojáře vlastníkem tohoto adresáře. Pracuji na svém notebooku sám a rovněž vyvýjím aplikace sám. Nepotřebuji tedy v této chvíli řešit spolupráci v týmu.

```
$ su
Password:roots password
# mkdir -p /var/lib/svn/project
# chown radek:radek /var/lib/svn/project
# exit
exit
```

47.22. Nasazení (*Deployment*)

47.22.1. Capistrano (dříve SwitchTower)

* ***Attributy:*** *id="rails.Capistrano"*

Odkazy:

- Capistrano (http://www.capify.org/index.php/Capistrano)
- Capistrano: Automating Application Deployment (http://manuals.rubyonrails.com/read/book/17)
- Capistrano 1.1 (http://weblog.rubyonrails.org/2006/03/06/capistrano-1-1/)
- Capistrano (http://wiki.rubyonrails.com/rails/pages/Capistrano)

Předpoklady z nichž Capistrano vychází

- Používáme alespoň jeden vzdálený server.
- K vzdáleným serverům musíme mít přímý přístup.
- Pro přístup se používá ssh.
- Vzdálený server musí mít příkazy definované normou POSIX.
- Hesla na vzdálené server jsou stejná.
- Některé akce chceme vykonávat jen na části serverů.

Standardní předpoklady jsou více specifičtější, ale lze je konfigurací změnit. Uvádím jen některé:

- Vypouštíme webovou aplikaci
- Pro vývoj a běh používáme Ruby on Rails
- Pro udržování kód používáme Subversion
- Aplikace je instalována na každém stroji do stejného adresáře `/u/apps/#{appname}`
- Používáme FastCGI
-

Aplikace musí být v stejných adresářích na všech serverech. Například `/home/rails/#{appname}`. V uvedeném adresáři se nachází samotná struktura aplikace která vypadá nsáledovně.

```
[approot]
  +-- releaded
  |     +-- 20051211101125
  |     +-- 20051217183521
  |     ...
  |     +-- 20060103141058
  |           +-- Rakefile
  |           +-- app
  |           +-- config
  |           +-- db
  |           +-- lib
  |           +-- log --> [approot]/shared/log
  |           +-- public
  |           |     +-- ...
  |           |         system -->[approot]/shared/system
  |           +-- script
  |           +-- test
  |           +-- vendor
  |
  +-- shared
  |     +-- log
  |     +-- system
  +-- current --> [approot]/releases/20060103141058
```

Kde `[approot]` je `/home/rails/#{appname}`.

47.22.1.1. Instalace

Instalujeme pomocí RubyGems příkazem

```
$ gem install capistrano
```

Poznámka: Pokud jsme používali switchtower je ten třeba napřed odinstalovat

```
$ gem uninstall switchtower
```

```
$ gem list capistrano

*** LOCAL GEMS ***

capistrano (2.5.18)
```

47.22.1.2. Konfigurace / nastavení pro vypuštění aplikace

Přepneme se do kořenového adresáře aplikace a vytvoříme konfigurační soubory pro Capostrano příkazem:

```
$ capify .
[add] writing './Capfile'
[add] writing './config/deploy.rb'
[done] capified!
```

Zkontrolujeme si vytvořené soubory a podíváme se co je v nich.

47.22.2. Instalace aplikace na produkční server Debian 4.0 a vyšší

Jednoduchý postup jak vystavuju aplikace. Nejdříve na produkčním serveru vyberu adresář. Já sám preferuji adresáře `/usr/local/share/`*`aplikace`*. Poté do aplikace dopíši rake úlohu `lib/tasks/upload.rake` s tímto obsahem:

```
# -*- mode:ruby; coding:utf-8 -*-

desc "Upload application to a production server"
task :upload do
    rexcl = '--exclude "*~" --exclude tmp/ --exclude log/'
    ropts = '-avPe ssh -C --delete --delete-after'
    puts "Uploading application"
    system "echo $PWD"
    system "rsync #{ropts} #{rexcl} ./ root@server.example.com:/usr/local/share/aplikace"
```

Distribuci aplikace provedu příkazem:

```
$ rake upload
```

Nyní je čas se podívat na server a upravit konfiguraci Apache2. Přidáme do něj další Rails aplikaci. Do souboru `/etc/apache2/sites-enabled/000-default` přidáme mezi Rails aplikace sekci

```
        # Další aplikace v Ruby on Rails.
        Alias /aplikace/     "/usr/local/share/aplikace/public/"
        Alias /aplikace      "/usr/local/share/aplikace/public/"
        <Directory /usr/local/share/aplikace/public>
                Options ExecCGI FollowSymLinks
                #FastCgiConfig -initial-env RAILS_ENV=production
                AllowOverride All
  # Omezení přístupu k aplikaci na úrovni Apache2.
                Order Deny,Allow
                Deny From All
  include /etc/apache2/allow-from-strediska
                include /etc/apache2/allow-special
        </Directory>
```

Po restartu apache vidíme úvodní stránku aplikace, ale nic nefunguje. Je třeba do našeho programu dodat ještě `.htaccess` soubor.

```
# General Apache options
AddHandler fcgid-script .fcgi
#AddHandler fastcgi-script .fcgi
AddHandler cgi-script .cgi
Options +FollowSymLinks +ExecCGI

# If you don't want Rails to look in certain directories,
# use the following rewrite rules so that Apache won't rewrite certain requests
#
# Example:
#   RewriteCond %{REQUEST_URI} ^/notrails.*
#   RewriteRule .* - [L]
```

```
# Redirect all requests not available on the filesystem to Rails
# By default the cgi dispatcher is used which is very slow
#
# For better performance replace the dispatcher with the fastcgi one
#
# Example:
#   RewriteRule ^(.*)$ dispatch.fcgi [QSA,L]
RewriteEngine On

# If your Rails application is accessed via an Alias directive,
# then you MUST also set the RewriteBase in this htaccess file.
#
# Example:
#   Alias /myrailsapp /path/to/myrailsapp/public
#   RewriteBase /myrailsapp

RewriteBase /aplikace
RewriteRule ^$ index.html [QSA]
RewriteRule ^([^.]+)$ $1.html [QSA]
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule ^(.*)$ dispatch.fcgi [E=X-HTTP_AUTHORIZATION:%{HTTP:Authorization},QSA,L]

# In case Rails experiences terminal errors
# Instead of displaying this message you can supply a file here which will be rendered instead
#
# Example:
#   ErrorDocument 500 /500.html

ErrorDocument 500 "<h2>Application error</h2>Rails application failed to start properly"

# Turn off mod_security filtering.
<IfModule mod_security.c>
  SecFilterEngine Off
</IfModule>
```

S tímto souborem jsme se dostali o kousek dál. Rails aplikace již vyhodí chybu při přístup na nějaký controller. Teď vytvoříme SQL databázi a nastavíme k ní přístup. Přihlásíme se tedy jako administrátor do Postgresové databáze a vatvoříme uživatele a databázi.

```
template1=# CREATE USER app_user WITH ENCRYPTED PASSWORD 'tajné-heslo' NOCREATEDB NOCREATEUSER;
CREATE USER
template1=# CREATE DATABASE app_database WITH OWNER app_user TEMPLATE=template0 ENCODING='utf-8'
CREATE DATABASE
```

Nyní musíme povolit přístup do této databáze. Do souboru `pg_hba.conf`, u mě je to `/etc/postgresql/7.4/main/pg_hba.conf` vložíme

```
# Aplikace nova-lekarna, produkční databáze.
local   novalekarna_production  novalekarna     md5
```

Reloadnem konfiguraci Postgresu a vyzkoušíme připojení.

```
# /etc/init.d/postgresql-7.4 reload
# psql -d app_database -U app_user -W
```

* ***FIXME:*** *úpravy v aplikaci*

47.22.3. Vlad

Odkazy:

- Vlad the Deployer (http://rubyhitsquad.com/Vlad_the_Deployer.html)
- Vlad the Deployer and Git (http://scie.nti.st/2007/9/25/vlad-the-deployer-and-git)

47.23. Rozpoznání uživatele a kontrola jeho práv (Authentication and Authorization)

* *section id="rails.authentication"*

Odkazy:

- Authentication (http://wiki.rubyonrails.org/rails/pages/Authentication) on Ruby On Rails Wiki
- User authentication in Ruby on Rails (http://codingbitch.com/p/comboy/User+authentication+in+Ruby+on+Rails)

Rozpoznání uživatele, Autentizace (http://cs.wikipedia.org/wiki/Autentikace) (Authentication (http://en.wikipedia.org/wiki/Authentication)) je nutnou podmínkou k řízení přístupu k naší aplikaci. Pokud nevíme kdo je na druhé straně, u prohlížeče, nevím co vše mu můžeme dovolit.

Postupně projdu od nejjednoduššího řešení k řešením komplikovanějším.

47.23.1. Jednoduchá kontrola přístupu

Začnu tím nejjednodušším řešením. Potřebujeme ochránit přístup k naší aplikaci nebo její části. Začneme třeba u řadiče `Listy` řídicího zobrazení servisních listů. Chceme aby neautorizovaní uživatelé měli jen možnost založit nový servisní list a nikoliv si prohlížet již založené servisní listy.

V řadiči tedy definuji soukromou metodu `login_required`:

```
private

def login_required
    unless session[:user_name]
        flash[:error] = 'Pro přístup k této stráunce musíte být přihlášen(a).'
 redirect_to :action => 'public_index'
    end
end
```

Tato metoda nědělá nic jiného, než že zkontroluje jestli je uživatel přihlášený a v případě že nikoliv, připraví o tom zprávu a přesměruje jej na nějakou veřejnou stránku. Nějaká veřejná stránka se míní taková, která není kontrolována a kterou smí vidět i neutorizovaný uživatel. Může to být například přihlašovací stránka.

Nyní můžeme v řadiči definovat filtr který ochrání stránky. K tomu použijeme `before_filter`. V případě že chceme ochránit přístup jen k několika vybraným stránkám, definujeme filtr takto:

```
before_filter :login_required, :only => [:edit, :destroy, :secret]
```

Pokud naopak chceme uživateli zamezit přístup všude, s výjimkou několika „veřejných“ stránek, napíšeme filtr takto:

```
before_filter :login_required, :except => [:login, :public_index, :new, :show, :edit]
```

V této chvíli nám již aplikace neumožní přístup na neveřejné stránky. Při přístupu na nevěřejnou stránku je připravena zpráva `flash[:error]`, kterou zobrazíme v layoutu. Protože s oblibou používám layout aplikace, uvedu do něj mezi jinými:

Příklad 47-16. `app/views/layouts/application.html.erb`:

```
<div id="pagecontent">
    <%= "<div class=\"error\">#{flash[:error]}</div>" if flash[:error] -%>
    <%= "<div class=\"info\">#{flash[:info]}</div>" if flash[:info] -%>
    <%= "<div class=\"notice\">#{flash[:notice]}</div>" if flash[:notice] -%>
    < yield -%>
</div>
```

Dalším krokem je umožnit uživateli přihlášení, tedy implemetovat metodu `login` s její formulář.

Příklad 47-17. `app/controllers/listy_controller.rb`:

```
    def login
        if request.post?
            if params[:login][:user_name] == 'admin' && params[:login][:password] == 'žížala'
                session[:user_name] == params[:login][:user_name]
  redirect_to :action => 'index'
            else
                @auth_error = 'Špatné jméno nebo heslo'
            end
        end
    end
```

* ***To be done.***

Začnu úpravou datového modelu uživatelů aplikace. Tento model, jenž se často jmenuje `User` s v mém případě jmenuje `Person`. Ale na jménu nezáleží. Tento model rozšíříme o metody umožňující ověření uživatele. V první variantě nebudu ukládat hesla ani jména do tabulky, protože budu mít jen jednoho „administrátora“ jehož přístup k aplikaci nebude omezován.

Pokud chceme ověřovat uživatele podle databáze, provedeme pár úprav. Nejdříve musíme mít v databázi uživatelská jména a hesla.

Příklad 47-18. `db/migrate/`*`999`*`_add_credentials`:

```
class AddCredentials < ActiveRecord::Migration
    def self.up
        add_column      :person, :login, :string, :limit => 16, ...
        add_column      :person, :sha1hash, :string, :limit => 34 ...
    end
    def self.down
        remove_column   :person, :login
        remove_column   :person, :sha1hash
    end
end
```

V modelu pak nejdříve definujeme metodu pro vytváření hashe z hesla. Tu jsem pojmenoval po metodě kterou používám, tedy `sha1_digest`. Metodě předávám jeden zatím nevyužitý parametr a to *`salt`*.

Příklad 47-19. `app/models/person.rb`:

```
require 'digest/sha1'

class Person < ActiveRecord::Base

    def self.sha1_digest(password, salt=nil)
        '{SHA}'+Base64.encode64(Digest::SHA1.digest(password)).chomp
    end
end
```

Metodu si odzkoušíme:

```
$ script/console
Loading development environment (Rails 2.0.2)
>> Person.sha1_digest('heslo')
=> "{SHA}bgF7VGT4IKbBu16fbXEaZnqA2Oo="
```

Nyní napíšeme metodu pro ověření uživatele `authenticate`

Příklad 47-20. `app/models/person.rb`:

```
    def self.authenticate(login, password)
        user = find(:first, :conditions => ['login = ?', login])
        return nil if user.nil?
        return user if (Person.sha1_digest(password) == user.sha1hash)
        nil
    end
```

47.23.2. Jednoduchý začátek

Nejdříve předestřu globální pohled na toto řešení a jeho integraci do aplikace. Přihlašovací informace jsou prezentovány v tabulce, v tomto případě pojmenované `people` ve forme přihlašovacího jména `login` a SHA1 hashe hesla `sha1hash`. Model této tabulky `Person` je rozšířen o několik metod sloužících k ověření hesla daného uživatele. Dále implementujeme

* ***FIXME:*** *work in progress.*

Začnu jednoduchým řešením s uživatli v databázi. Předpokládejme že uživatelé jsou v tabulce 'people' popsané modelem `Person`. Sloupec s přihlašovacím jménem se jmenuje `login`, a hash hesla je uložen v sloupci `sha1hash`.

První věcí kterou potřebujeme udělat je rozšířit model o několik metod které budeme potřebovat.

```
    def valid_password?(password)
        self.sha1hash == self.sha1_digest(password)
    end

    def self.sha1_digest(password, salt=nil)
        '{SHA}'+Base64.encode64(Digest::SHA1.digest(password)).chomp
    end

    def self.authenticate(login, password)
        user = find(:first, :conditions => ["login = ?", login])
        return nil if user.nil?
  return user if (Person.sha1_digest(password) == user.sha1hash)
        nil
```

```
    end

  def login
    if request.post?
      @user = Person.find_by_login(params[:login])
      if @user and @user.login == 'radek'
        session[:uid] = @user.id
        redirect_to :controller => 'listy', :action => 'index'
      else
        @auth_error = 'Wrong username or password'
      end
    end
  end
```

47.23.3. HTTP Basic authentication

- ActionController::HttpAuthentication::Basic (http://api.rubyonrails.org/classes/ActionController/HttpAuthentication/Basic.h

Nejdříve si napíšeme nejjednodušší možnou metodu pro zjištění uživatele a povolení jeho přístupu. Tuto metodu deklarujeme jako privátní a uvedeme ji v řadiči aplikace `app/controllers/application.rb`.

```
class ApplicationController < ActionController::Base
    ...
    private
        def authenticate
            authenticate_or_request_with_http_basic do |username, password|
                username == 'admin' && password == 'heslo'
            end
        end
end
```

Pokud máme uživatelů více a nechceme volit složitější řešení. Tedy pokud nám na penvo zakódované účty a hesla vyhovují, můžeme telo metody napsat nepříklad takto.

```
authenticate_or_request_with_http_basic do |username, password|
    ( username == 'admin' && password == 'heslo' )
 || ( username == 'ja'    && password == 'mojeheslo )
 || ...
end
```

V konfiguraci Apache, pokud jej používáme jako WWW server je třeba upravit jedno přepisovací pravidlo.

```
RewriteRule ^(.*)$ dispatch.fcgi [E=X-HTTP_AUTHORIZATION:%{HTTP:Authorization},QSA,L]
```

Autentikační fukci pak voláme z řadiče aplikace či z konkrétních řadičů. V následujícím příkladu se kontrola uživatele vztahuje na všechna volání řadiče `ProduktyController` s výjimkou volání `nabidka`.

```
class ProduktyController < ApplicationController
    before_filter :authenticate, :except => [:nabidka]
    ...
```

47.23.4. Session authentication

Controller

```
class ListyController < ApplicationController
    def index
        login
        render(:action => 'login')
    end

    def login
        #@user = User.new
    end

    def send_login
 found_user = User.authenticate(params[:username], params[:password])
 if found_user
     session[:user_id] = found_user.id
     flash[:notice] = "You are now logged in."
     redirect_to(:action => 'menu')
 else
     flash.now[:notice] = "Username/password combination incorrect."
     render(:action => 'login')
 end
    end

    def logout
        session[:user_id] = nil
 flash[:notice] = 'You are now logged out."
 redirect_to(:action => 'login')
    end
end
```

app/view/listy/login.html.r

```
<% @page_title = 'Oblast chráněná přihlášením' -%>
<% form_tag(:action => 'send_login') do -%>
    <p>Username: <%= text_field_tag('username', params[:username]) %></p>
    <p>Password: <%= password_field_tag('password') %></p>
    <%= submit_tag("Log in") %>
< end -%>
```

Příklad 47-21. Změny v modelu

```
class User < ActiveRecord::Base
    ...

    attr_accessor :password
    #attr_accessible :first_name, :last_name, :email, ... :username, :password
    attr_protected :hashed_password

    def before_create
        self.hashed_password = User.hash_password(@password)
    end
    def before_update
        if ! @password.blank?
            self.hashed_password = User.hash_password(@password)
```

```
        end
    end

    def after_save
        @password = nil
    end

    def before_destroy
        # Zabránění odstranění prvního uživatele.
        return false if self.id == 1
    end

    # Ověření uživatele podle jména 'username' a hesla 'password'
    def self.authenticate(username, password)
        hashed_password = self.hash_password(password)
        user = self.find(:first, :conditions => ["username = ? AND hashed_password = ?", userr
        return user
    end

    private

    def self.hash_password(password)
        #
        return Digest::SHA1.hexdigest(passwrord)
    end

    ...
end
```

Příklad 47-22. Úpravy v řadiči aplikace

```
class ApplicationController < ActionController::Base
    ...
    private

    def authorize_access
        if !session[:user_id]
            flash[:notice] = "Please log in."
            redirect_to(:controller => 'staff', :acction => 'login')
            return false
        end
    end
end
```

V řadičích pak použijeme

```
class ... < ApplicationController
    before_filter :authorize_access
end

class ... < ApplicationController
    before_filter :authorize_access, :except => [:index, :login, :send_login]
end
```

47.23.5. Declarative Authorization

Do souboru `config/environment.rb` přidáme:

```
Rails::Initializer.run do |config|
    ...
    config.gem "declarative_authorization", :source => "http://gemcutter.org"
    ...
```

Gem nainstalujeme. Tady opět používám uživatelskou instalaci.

```
$ rake gems:install
```

Poté si vytvoříme konfigurační soubor `config/authorization_rules.rb`.

```
authorization do

    role :admin do
        has_permission_on [:articles,:comments], :to => [:index, :show, :new, :create, :edit,
    end
end
```

V modelu pak nastavíme

```
class User < ActiveRecord::Base
    act_as_authentic
    has_many :articles
    has_many :comments
    ...
    has_many :roles, :through => :assignments

    def role_symbols
        roles.map do |role|
            rols.name.underscore.to_sym
        end
    end
end
```

*

* ***To be done.***

47.23.6. Password salt

Před hashováním upravíme heslo tím že k němu přidáme sůl (další text). Bez znalosti této soli nikdo nemůže zopakovat hashování hesel.

Příklad 47-23. Heslo se solí v modelu

```
class User < ActiveRecord::Base
    ...
    private

    def self.hash_with_salt(password)
        retuirn Digest::SHA1.hexdigest("Put salt on the #{password}")
    end
end
```

Náhodná sůl v hesle

```
$ script/generate migration AddPasswordSalt
      exists  db/migrate
      create  db/migrate/006_add_password_salt.rb

class AddPasswordSalt < ActiveRecord::Migration
    def self.up
        add_column :users, :salt, :string, :limit => 40
        User.find(:all).each do |user|
            user.password = user.hashed_password
            user.save
        end
    end

    def self.down
        # Nevratná změna v databázi.
        remove_column :user, :salt
    end
end

$ rake db:migrate

class User < ActiveRecord::Base
    ...
    attr_protected :hashed_password, :salt

    def before_create
        self.salt = User.make_salt(self.login)
        self.hashed_password = User.hash_with_salt(@password,  self.salt)
    end
    def before_update
        unless @password.blank?
     self.salt = User.make_salt(self.username) if self.salt.blank?
            self.hashed_password = User.hash_with_salt(@password, salt.salt)
        end
    end

    def self.authenticate(username = ”, password = ”)
        user = self.find(:first, :conditions => ["username = ?", username])
        return (user && user.authenticated?) ? user : nil
    end

    def authenticated?(password = ”)
        self.hashed_password == User.hash_with_salt(password, self.salt)
    end


    private

    def self.make_salt(string)
        return Digest::SHA1.hexdigest(string.to_s + Time.now.to_s)
    end

    def self.hash_with_salt(password, salt)
        return Digest::SHA1.hexdigest("Put #{salt} on the #{password}")
    end
    ...
```

```
end
```

47.23.7. Poznámky

- ActiveRecords Slides (http://www.slideshare.net/rabble/introduction-to-active-record-silicon-valley-ruby-conference-20007/)

```
class User < ActiveRecord::Base
    def self.authenticate(username, password)
        find(:first, :conditions =>
            {:username => username, :password => password})
    end
end
```

47.24. REST design

REST (*Representational State Transfer*)

RESTfull design je přístup k programování webů který popisuje jakým způsobem se k jednotlivým objektům přistupuje. V podstatě logicky mapuje práci s jednotlivými objekty na url.

Pro přechod k REST budeme tedy měnit nejen řadič, ale také routy jenž k tomuto řadiči povedou.

Tabulka 47-5. RESTFull

standardní metody	sloveso	url	akce
*plural*_path	GET	/notes	index
*singular*_path(id)	GET	/notes/1	show
new_*singular*_path	GET	/notes/new	new
*singular*_path	POST	/notes	create
edit_*singular*_path(id)	GET	/notes/1;edit	edit
*singular*_path(id)	PUT	/notes/1	update
*singular*_path(id)	DELETE	/notes/1	destroy

```
# GET /notes
# GET /notes.xml
# ...
def index
    @notes = Note.find(:all)
    respond_to do |format|
        format.html # index.html.erb
        format.xml  { render :xml => @notes.to_xml }
        format.rss  { render :xml => @notes.to_rss }
    end
end
```

Příklad 47-24. Controller Template

```
class MyController < ApplicationController

    # GET /notes
    # GET /notes.xml
    def index
        @notes = Note.find(:all)
        respond_to do |format|
            format.html # index.html.erb
            format.xml  { render :xml => @notes.to_xml }
        end
    end

    # GET /notes/1
    # GET /notes/1.xml
    def show
        @note = Note.find(params[:id])
        respond_to do |format|
            format.html # index.html.erb
            format.xml  { render :xml => @notes.to_xml }
        end
    end

    # GET /notes/new
    def new
        @mode = 'new'
        @note = Note.new
    end

    def create
        @note = Note.create(params_hash)
        respond_to do |wants|
            wants.html {redirect_to :action=>:index}
            wants.js   {render}
        end
    end

    def edit
        @note = Note.find(params[:id])
    end

    def update
        object = @note, ...
        if Note.update_attributes(params_hash)
            respond_to do |wants|
                wants.html {redirect_to :action=>:index}
                wants.js   {render}
            end
        else
            render
        end
    end

    def destroy
        ...
    end
```

```
end
```

Routes

```
ActionController::Routing::Routes.draw do |map|
    map.resources :notes, :sessions
end
```

V pohledu index.html.erb

```
<h1>Přehled poznámek</h1>
<table>
    <tr></tr>
  <% for note in @notes %>
    <tr>
        <td></td>
        <td><%= link_to 'Show', note_path(note) %></td>
        <td><%= link_to 'Edit', edit_note_path(note) %></td>
        <td><%= link_to 'Destroy', note_path(note),
                        :confirm => 'Are you sure?',
                        :method => :delete %></td>
    </tr>
  <% end %>
</table>
<br/>
<%= link_to 'Nová poznámka'm new_note_path %>
```

Místo odkazů `link_to` můžeme používat tlačítka `button_to`. Je třeba vědět, že tlačítka tako vytvářená sebou "nesou" celý formulář. Pokud chceme použít AJAX, budou uvedené řádky vypadat

```
        <td><%= link_to_remote 'Destroy',
                        :url => note_path(note),
                        :confirm => 'Are you sure?',
                        :method => :delete %></td>
```

Řadič k mazání záznamů (metoda `destroy`)

```
    # DELETE /notes/1
    # DELETE /notes/1.xml
    def destroy
        @note = Note.find(params[:id])
        @note.destroy

        respond_to do |format|
            format.html { redirect_to notes_url }  # go to index
            format.js   # run the destroy.rjs template
            format.xml  { render :nothing = > true }
        end
    end
```

Smazání poznámky z konzole

```
$ curl -X DELETE -H "Accept: text/xml" http://localhost:3000/notes/2

$ curl -X DELETE -H "Accept: application/javascript" http://localhost:3000/notes/2
```

Ošetření chyb. Tedy pokoušíme se smazat něco co již neexistuje

```
    # DELETE /notes/1
    # DELETE /notes/1.xml
    def destroy
        @note = Note.find(params[:id])
        @note.destroy

        respond_to do |format|
            format.html { redirect_to notes_url }  # go to index
            format.js   # run the destroy.rjs template
            format.xml  { render :nothing = > true }
        end
 rescue Exception => e
     respond_to do |format|
                format.html { render :text => "Record not found", status => 404 }
                format.js   { render(:update) {|page| page.alert("There was an error")}}
                format.xml  { render :nothing = > true, :status => 404 }
            end
    end
```

K editaci potřebujeme pohled `edit.html.erb`

```
<h1>Editace poznámky</h1>
<% form_for(:note,
            :url => note_path(@note),
            :html => { :method => :put }) do |f| %>

  <label for="">Title
    <%= f.text_field :title, :class => 'text_field' %>
  </label>
  <label for="">Content
    <%= f.text_area :content %>
  </label>

  # nebo výš uvedená pole formuláře umístníme do samsotatného souboru _form.hmtl.erb
  # a zde jen uvedeme partial
  <%= render :partial => 'form', :locals => {:f => f} %>

  <p><%= submit_tag "Update" %></p>
<% end %>
<% link_to 'Show', note_path(@note) %> |
<% link_to 'Back', notes_path %>
```

Pohled pro vytváření novoho záznamu:

```
<h1>Nová poznámka</h1>
<% form_for(:note, :url => notes_path) do |f| %>
  <%= render :partial => 'form', :locals => {:f => f} %>

  <p><%= submit_tag "Create" %></p>

<% end %>
<% link_to 'Back', notes_path %>

$ script/plugin install http://dev.rubyonrails.org/svn/rails/plugins/simply_helpful/
```

Formáty použité v `respond_to do |format|` můžeme rozšířit o vlastní formát. V souboru `environment.rb` zaregistrujeme mime typ.

```
Mime::Type.register 'application/vnd.blinksale+xml', :api
Mime::Type.register 'application/vnd.visa+xml, :visa
```

Máme-li model s relací one_to_many, potřebujeme naprogramovat aplikaci tak, aby v rámci RESTFull principu "rozuměla" url jako:

/notes/1/keywords/6

Příklad 47-25. routes.rb pro vnořenou tabulku /notes/:note_id/keywords/:id

```
ActionController::Routing::Routes.draw do |map|
    ...
    # /notes/:note_id/keywords/:id
    map.resources :notes do |notes|
        notes.resources :keywords, :name_prefix => 'note_'
    end
    map.resources :keywords
```

47.24.1. Akce řadiče

Tabulka 47-6. RESTfull Named Routes in Interaction with HTTP Request Methods

	GET	POST	PUT	DELETE
note_url(@note)	/notes/1 *show*		/notes/1 *update*	/notes/1 *destroy*
notes_url	/notes *index*	/notes *create*		
edit_note_url(@note)	notes/1;edit *edit*			
new_note_url	/notes/new *new*			

47.24.2. Vnořené routy

/poznamky/3/veci/6

```
map.resources :poznamky do |poznamka|
    poznamka.resources :veci, :name_prefix => 'poznamka_'
end

map.resources :veci
```

Kdyby v uvedené ukázce nebyl použit `:name_prefix` tak by se metody pro veci přepsaly posledním mapem.

Tabulka 47-7. Routy a metody

pomocná metoda	URL
poznamka_url(@poznamka)	/poznamky/3
vec_url(@vec)	/veci/5
poznamka_vec_url(@poznamka,@vec)	/poznamky/3/veci/5

47.25. Rake a Rails

Odkazy:

- Custom Rake Tasks (http://railscasts.com/episodes/66)
- Seed Data in Rails (http://www.quotedprintable.com/2007/11/16/seed-data-in-rails)
- RAKIE PANIKIE (http://errtheblog.com/posts/36-rakie-pankie) on ERR THE BLOG

Rake je v RoR využíváno pro mnoho úloh. Od spouštění testů, migrací, vytváření a práci s databází, generování dokumentace, ... Rovněž je RoR připraveno pro snadné definování clastních úloh v Rake. Tyto se zapisují nejlépe do samostatných souborů do adresáře `lib/tasks/`. Jména souborů musí končit `.rake`.

Příklad 47-26. Jednoduchý task v rake

```
desc "Pokusná úloha"
task :pokus do
    sh %Q(echo "Toto je pokus)  # Voláme shell
    puts "Toto je pokus"        # ruby kód
end
```

Máme-li více úloh, jež spolu logicky souvisejí, dáme je do jednoho jmenného prostoru.

Příklad 47-27. Více seskupených úloh

```
namespace :my do
    desc "První úloha"
    task :first do
        puts "První"
    end

    desc "Druhá úloha"
    task :second do
        puts "Druhá"
    end

    def spolecna_metoda(...)
        ...
    end
end
```

Pokud ve své úloze potřebujeme pracovat v prostředí rails aplikace, musím to oznámit. Učiníme tak tím, že úlohu učiníme "závislou" na rails úloze `:environment`. Úlohu ve které potřebujeme mít definováno prostředí. Pokud tedy naše úloha potřebuje přístup k datům přes ActiveRecord, napíšeme ji podle vzoru:

```
desc "První uživatel"
task :first_user => :environment
    uzivatel = User.find(:first)
    puts "Uživatel: #{uzivatel.full_name}"
end
```

47.25.1. Svn a Rake

Nová úloha / task, umožní rake přidat (add) do svn všechny nově vzniklé (vygenerované, vytvořené) soubory v projektu.

```
###
desc "Add generated files to Subversion"
task :add_to_svn do
    sh %Q{svn status| ruby -nae 'puts $F[1] if $F[0] == "?"'} +
       %Q{| xargs svn add}
end
```

47.25.2. Různé úlohy

47.25.2.1. Nahrání aplikace na server pomocí rsync

Tato rake úloha je náhradou za mnou obvykle používaný shell script `upload` jenž častou používám v jednotlivých projektech pro distribuci na server. Je to velmi jednoduchá úloha, provádějící prosté nahrání aplikace na server. Neřeší věci kolem restartu/reloadu aplikačního serveru, neřeší migrace databází. Prostě jen nahraje aplikaci.

Parametry jsou uvedeny na začátku úlohy. Musíme upravit `ruser`, což je uživatel jenž má právo se přihlásit pomocí **ssh** na server `rserver`. Aplikaci pak nahráváme na serveru do adresáře `rdir`.

```
desc "Upload application to a production server"
task :upload do
    rexcl = '--exclude "*~" --exclude tmp/ --exclude log/'
    ropts = '-avPe ssh -C --delete --delete-after'
    ruser = 'root'
    rserver = 'app.example.com'
    rdir = '/usr/local/share/aplikace'

    puts "Uploading application"
    system "echo $PWD"
    system "rsync #{ropts} #{rexcl} ./ #{ruser}@#{rserver}:#{rdir}"
end
```

47.26. Poznámky

47.26.1. Inflector

Inflector je část Rails zodpovědná za „překlad“ jmen. Z jednotného čísla do množného a naopak, a do různých forem jako je velbloudí forma, podtržítková forma, atd.

`pluralize` / `singularize`

> Jak již název napovídá, tyto metody převádějí podstatné jméno z jendotného do množného čísla a naopak. Upozorňuji, podstatné jména v jazyce anglickém! Inflector nemusí být při tvorbě množného či

jenotného čísla správný, jeho kód nemusí postihovat všechny gramatické zvláštnosti a výjimky. Pokud jsme na pochybách, můžme si ho vyzkoušet.

```
$ script/console
Loading development environment.
>> Inflector.pluralize 'person'
=> "people"
>> Inflector.singularize 'records'
=> "record"
```

Chování inflektoru můžeme ovlivnit předefinováním jeho metod. Pokud ale neznáme všechny důsledky takových úprav, měly bychom býti zdrženliví. Svého dosáhneme ve většině případů úpravou pravidel. Od verze Rails 2.0 se konfigurace inflektoru nachází v souboru `config/initializers/inflections.rb` v dřívějších verzích ji píšeme přímo do souboru `config/environment.rb`. V obou případech je v konfiguračním souboru uvedena ukázka.

```
# Add new inflection rules using the following format
# (all these examples are active by default):
# Inflector.inflections do |inflect|
#   inflect.plural /^(ox)$/i, '\1en'
#   inflect.singular /^(ox)en/i, '\1'
#   inflect.irregular 'person', 'people'
#   inflect.uncountable %w( fish sheep )
# end
```

> **Varování**
>
> Podle tohoto vzoru si na konec konfiguračního souboru provedeme vlastní úpravy. Upozorňuji že to musí být mimo část `Rails::Initializer.run`, jinak nám to nebude fungovat.

Pro všechny tabulky, které budeme používat, si zavedeme vztah mezi názvem v množném a jednotném čísle. K tomu použijeme metodu `irregular`. Například používáme tabulku lidé (`lide`), kde jednotným tvarem je člověk (`clovek`).

```
Inflector.inflections do |inflect|
    inflect.irregular 'clovek', 'lide'
end
```

Protože nevíme co vše inflector umí, je důležité si své změny hned odzkoušet.

```
$ script/console
>> Inflector.pluralize 'clovek'
=> "lide"
>> Inflector.singularize 'lide'
=> "clovek"
```

V našem případě je velmi nepravděpodobné, že by měl inflector naprogramováno slovo clovek mezi `uncountable`. Ale jeden nikdy neví, co se uvnitř děje. Pokud se nám tedy stane, že po zavedení `.irregular` nám toto nefunguje, může to být z důvodu že použité slove je definováno v `.uncountable`. Slova zavedené pomocí metody `.uncountable` totiž mají přednost slovy zavedenými `.irregular`. V takovém případě příslušné slovo z `.uncountable` odebereme.

* *Podle: http://elia.wordpress.com/2007/07/26/rails-inflector-uncountable-troubles-just-remember-whys-dr-cham/*

```
Inflector.inflections do |inflect|
  # následující řádek nestačí, protože uncountables mají přednost
  inflect.irregular 'equipment', 'equipments'
```

```
  def inflect.remove_uncountable word
    @uncountables.delete word
  end

  inflect.remove_uncountable 'equipment'
end
```

FIXME: Pokud potřebujeme zavést jen pár nových pravidel, například pro práci se podstatnými jmény v jiném jazyce, můžeme se inspirovat přímo ukázkou uvedenou v souboru `config/environment.rb`:

Také: 47.5.1

47.26.2. Čeština

Odkazy:

- HowToUseUnicodeStrings (http://wiki.rubyonrails.com/rails/pages/HowToUseUnicodeStrings)
- Secure UTF-8 Input in Rails (http://www.igvita.com/blog/2007/04/11/secure-utf-8-input-in-rails/)
- Unicode in Ruby on Rails (http://ruby.org.ee/wiki/Unicode_in_Ruby/Rails)

Čeština v Rails se dotýká několika věcí. Jednak je to kódování htmlstránek, práce s řetězci a v neposlední řadě práce s databází.

```
# aptitude install libonig-dev
# gem install oniguruma
# gem install unicode
```

47.26.3. Zmrazení Rails

Pokud potřebujeme do Rails aplikace vložit samotné rails, aby se nenačítali z rubygems nebo z systémových adresářů, můžeme použít rake úlohu

```
$ rake rails:freeze:gems
```

V mé starší apliakci v Rails 2.0.2 mi to ovšem nefungovalo. Musel jsem použít ruční postup. Nejdříve jsem si stáhl jednotlivé gemy.

```
$ gem install rails --version 2.0.2 --install-dir /tmp/gem --no-rdoc
```

Potom jsem provedl „ruční“ zmrazení v aplikaci takto:

```
$ cd vendor
$ mkdir rails
$ cp -a /tmp/gem/gems/acti* .
$ cp -a /tmp/gem/gems/rails* .
```

Adresáře je třeba ještě přejmenovat a nebo vytvořit symbolické odkazy. Zvolil jsem druhý postup.

```
$ ln -s actionmailer-2.0.2 actionmailer
$ ln -s actionpack-2.0.2 actionpack
$ ln -s activerercord-2.0.2 activerecord
$ ln -s activeresource-2.0.2 activeresource
```

```
$ ln -s activesupport-2.0.2 activesupport
$ ln -s rails-2.0.2 railties
```

Aplikace niní již nastartuje ale zatím nevím je li vše v pořádku. Musím přenést na druhý počítač s aplikací i databáze.

47.27. Nezatříděné poznámky

* *section id="rails.asorted.notes" status="draft"*

47.27.1. Akce kontroleru

- `show_thing`
- `edit_thing`

```
def show_thing
    @thing = Thing.find(params[:id])    # select 1
end

def edit
    @thing = Thing.find(params[:thing][:id])    # select 2
    @thing.attributes = params[:thing]
    if @thing.save      # update
        ...
    else
        ...
    end
end
```

47.27.2. Prostředí Apache

Proměnné a parametry prostředí (CGI, FCGI) by měly být přístupny metodou `request.env`.

47.27.3. Poznámky k sesion managementu

The ActiveRecordStore backend for CGI::Session allows you to plug in custom session classes. See session/active_record_store.rb in Action Pack. The default session class is a normal Active Record class named Session. An alternative, SqlBypass, is provided which duck-types with the AR class but uses straight SQL over the db connection. You can use it as a starting point for pessimistic locking.

```
class MyPessimisticSession < CGI::Session::ActiveRecordStore::SqlBypass
  # Pick a database connection for straight SQL access.
  self.connection = ActiveRecord::Base.connection

  # Override the finder class method to do a SELECT ... FOR UPDATE.
  def self.find_by_session_id(session_id)
    @@connection.select_one "..."
  end
```

```
end

# Use our custom session class.
CGI::Session::ActiveRecordStore.session_class = MyPessimisticSession

# Use the ActiveRecordStore for CGI sessions.
ActionController::CgiRequest::DEFAULT_SESSION_OPTIONS.update(
  :database_manager => CGI::Session::ActiveRecordStore
)
```

As a footnote, I believe the default session backend (PStore) serializes access since it uses file locking. Perhaps it's worth a second look?

47.27.4. Vypnutí mechanizmu přejmenování objektů

Radek Hnilica wrote:

>Please, pleas, please, is there a way to completly shoot down this
>autorenaming feature?

http://wiki.rubyonrails.com/rails/show/HowtoDisableAutoTableNamePluralisation

In short, add the line:

```
ActiveRecord::Base.pluralize_table_names = false
```

to the bottom of:

```
rails_application_path/config/environment.rb
```

Toto ovšem nepomůže úplně. Pakliže se naše tabulka jmenuj například `pokus`, dojde Rails k názoru že se jedná o množné číslo `poku` a scaffold v controlleru s pokusí použít objekt jména `Poku` který ovšem neexistuje neb byla vytvořena třída `Pokus`. Následující kód přidaný na konec souboru `config/environment.rb` vyřadí mechanismus převodu jmen z jednotného na množné číslo úplně.

```
module Inflector
    def pluralize(word)
        word
    end
    def singularize(word)
        word
    end
end
```

47.27.5. Datové modely / ActiveRecord

Třídu pro nový datový model vygenerujeme skriptem například pro tabulku `calls` vytvoříme model příkazem:

```
$ script/generate model call
      exists  app/models/
      exists  test/unit/
      exists  test/fixtures/
      create  app/models/call.rb
```

```
      create  test/unit/call_test.rb
      create  test/fixtures/calls.yml
```

Všiměte si rozdílu ve jménech. Zatímco datový model se jmenuje `call` (jednotné číslo) datová tabulka se jmenuje `calls` (množné číslo). Tzn. Je třeba znát angličtinu, neb pojmenování datových tabulek je očekáváno v angličtině.

Pokud potřebujeme upravit datový model, otevřeme si v editoru vygenerovaný soubor `app/models/call.rb` a upravíme.

Pokud se datová tabulka jmenuje jinak než ze jména modelu předpokládá `ActiveRecord`, můžeme ji předfinovat

```
class Hovor < ActiveRecord::Base
    # Database Table Maping
    set_table_name 'Hovory'
end

    set_primary_key 'client_id'
    belongs_to :group
    belongs_to :subnet
```

47.27.6. Jednořádkové poznámky

`public/index.html`

`config/routes.rb` — `map.connect 'pressreleases', :controller=>'press_releases', :action=>'index'`

Ruby Friday (http://www.ruby-friday.org/)

47.27.7. Kousky kódu

```
Account.transaction(david, mary) do
    david.withdrawal(100)
    mary.deposit(100)
end

class Account < ActiveRecord::Base
    validates_presence_of     :subdomain, :name, :email_address, :password
    validates_uniqueness_of   :subdomain
    validates_acceptance_of   :terms_of_service, :on => :create
    validates_confirmation_of :password, :email_addressm, :on => :create
end

class Project < ActiveRecord::Base
    belongs_to              :portfolio
    has_one                 :project_manager, :class_name => "Person"
    has_many                :milestones, :dependent => true
    has_and_belongs_to_many :categories, :join_table => "categorizations"
end

class Account < ActiveRecord::Base
    has_many :people do
        def find_or_create_by_name(name)
            first_name, *last_name = name.split
```

```
                last_name = last_name.join " "

                find_or_create_by_first_name_and_last_name(first_name, last_name)
            end
        end
    end

    person = Account.find(:first).people.find_or_create_by_name("Richard Feynman")
    person.first_name # => "Richard"
    person.last_name  # => "Feynman"

    class Post < ActiveRecord::Base
        belongs_to :weblog
        has_many   :comments
        has_one    :author, :class => "Person"
    end

    class Comment < ActiveRecord::Base
        def sel.search(query)
            find(:all, :conditions => [ "body = ?", query ])
        end
    end

    # SELECT * FROM comments WHERE post_id=1 AND body='hi';
    Post.find(1).comments.search "hi"

    class Story < ActiveRecord::Base
        belongs_to   :iteration
        acts_as_list :scope => :iteration
    end

    story.move_higher
    story.move_to_bottom
```

FIXME:

```
    class Author < ActiveRecord::Base
        has_many :authorships
        has_many :books, :through => :authorships
    end

    class Book < ActiveRecord::Base
        has_many :authorships
        has_many :authors, :through => :authorships
    end

    class Authorship < ActiveRecord::Base
        belongs_to :author
        belongs_to :book
    end

    david, awrd = Author.find(1), Book.find(1)
    david.authorships.create(
        :book => awrd, :contribution => "partial", :written_in => "Denmark"
    )

    david.authorships.first.written_in # => "Denamrk"
    david.books.first # => awrd
```

FIXME:

```
class Person < ActiveRecord::Base
    has_many :taggings, :as => :taggable, :dependent => true
    has_many :tags, :through => :taggings
end

class Message < ActiveRecord::Base
    has_many :taggings, :as => :taggable, :dependent => true
    has_many :tags, :through => :taggings
end

class Tagging < ActiveRecord::Base
    belongs_to :tag
    belongs_to :taggable, :polymorfic => true
end

class Tag < ActiveRecord::Base
    has_many :taggings

    def on(*taggables)
        taggables.each {|taggable| taggings.create :taggable => taggable }
    end

    def tagged
        taggings.collect {|tagging| tagging.taggable }
    end
end

david   = Person.find(1)
welcome = Message.find(1)
summer  = Tag.find_or_create_by_name "Summer"

summer.on(david, welcome)

david.tags   # => [ summer ]
welcome.tags # => [ summer ]

summer.tagged # => [ david, welcome ]

class Person < ActiveRecord::Base
    acts_as_tagable
end

class Message < ActiveRecord::Base
    acts_as_tagable
end
```

FIXME:

```
class StoryController < ApplicationController
    session       :off,             :only => :feed
    verify        :method => :post, :only => [ :coment ]
    before_filter :authenticate,    :only => [ :new, :edit ]

    def show
        @story = Story.find(params[:id])
    end
```

```
        def new
            if request.story?
                @story = Story.create(params[:story])
                cookies[:last_story] = {
                    :value   => @story.create_at,
                    :expires => 20.years.from_now
                }

                flash[:notice] = "Created new story"
                redirect_to :action => "show", :id => @story
            else
                @story = Story.new
                render :layout => "creation"
            end
        end
    end

    <h1>Create a new iteration</h1>
    <p>
        You already have <%= pluralize(@iterations.size, "iteration") %>.

        The last one started like this:
        "<%= truncate(@iterations.first.description, 10) %>"
    </p>

    <% form_for :iteration, Iteration.new do |i| %>
        Title:<br/>
        <%= i.text_field :title %>

        Description:<br/>
        <%= i.text_area :title, :size => "20x40" %>

        Completion target:<br/>
        <%= i.date_select :completed_on, :discard_year => true %>

        <% fields_for :story, Story.new do |s| %>
            Headline:<br/>
            <%= s.text_field :headline, :length => 30, :class => "big" %>

            Difficulty:<br/>
            <% s.select :difficulty, %w(easy medium hard) %>
        <% end %>
    <% end %>

    ...or <%= link_to "Cancel", :action => "overvire" %>

    xml.instruct!
    xml.rss("version" => "2.0", "xmlns:dc" => "http://purl.org/dc/elements/1.1/") do
        xml.channel do
            xml.title "Backpack"
            xml.link(url_for(:action => "start", :only_path => false))
            xml.description "Tracking changes for your Backpack pages"
            xml.language "en-us"
            xml.ttl "40"

            for event in @events
                xml.item do
```

```
                xml.title(event.headline)
                xml.description(event.description)
                xml.pubDate(event.created_at.to_s(:rfc822))
                xml.guid(event.id)
                xml.link(page_url(:id => event.page))
            end
        end
    end
end

Rails::Initializer.run do |config|
    config.frameworks -= [ :action_web_service ]
    config.load_paths += [ "#{RAILS_ROOT}/app/services" ]
    config.log_level   = :debug

    config.action_controller.session_store        = :active_record_store
    config.action_controller.fragment_cache_store = :file_store

    config.active_record.default_timezone = :utc
    config.active_record.schema_format    = :ruby
end

ActionController::Routing::Routes.draw do |map|
    map.start   ”, :controller => ’pages’, :action => ’start’
    map.signup  ’signup’, :controller => ’account’, :action => ’new’
    map.page    ’page/:id’, :controller => ’pages’, :action => ’show’
    map.connect ’page/:id/:action’, :controller => ’pages’
    map.connect ’images/icons/:icon_file’, :controller => "assets", :action => "missing_icon"

    map.feed    ’feed/:token’, :controller => "account", :action => "feed"
    map.ical    ’ical/:token’, ::controller => "reminders", :action => "ical"
end

set :application, "backpack"
set :repository,  "svn+ssh://somewhere/nice"

set :gateway, "gateway.37signals.com"

role :app, "app1.37signals.com", :primary => true
role :app, "app2.37signals.com"
role :app, "app3.37signals.com"
role :web, "web1.37signals.com", :primary => true
role :web, "web2.37signals.com"
role :web, "db.37signals.com",   :primary => true
```

47.27.8. Zajímavá konfigurace

Následující ukázka je z konfiguračního souboru `config/environment.rb`

```
86      # Include your app’s configuration here:
87
88  # our primary keys are named <tablename>_id
89  ActiveRecord::Base.set_primary_key do
90    "#{@table_name}_id"
91  end
92
```

```
93  ActiveRecord::Base.pluralize_table_names = false
94  ActiveRecord::Base.set_inheritance_column do
95    'activerecord_stinks'
96  end
```

47.27.9. Magické / speciální sloupce v datové tabulce

FIXME:

- created_at
- `created_on`
- `updated_at`
- `updated_on`
- `lock_version`
- `type`
- `id`
- `#{table_name}_count`
- `position`
- `parent_id`
- `lft`
- `rgt`

47.27.10. Promazávání souborů sezení

FIXME:

```
find /tmp -name "ruby_sess*" -amin +600 -exec rm {} \;
```

47.27.11. Formuláře a vstup dat

FIXME:

47.27.12. Upgrade aplikace v Rails

Nenašel jsem žádný postup jak upgradovat aplikaci na novou verzi Rails. Postupoval jsem tedy ručně. Nejdříve jsem si zazálohoval celou aplikaci. Poté jsem v aplikaci provedl upgrade příkazem

```
$ rails -c .
```

Tento příkaz přepíše řadu souborů které budeme obnovovat ze zálohy. Nejdříve konfigurace. Součásti konfigurace je konfigurace Inflektoru, ta se nyní nachází na jiném místě, a to v souboru `config/initializers/inflections.rb`. Ze zálohy tedy znovu nakonfigurujeme inflector. Dále obnovíme konfiguraci databáze.

```
$ cp ../zaloha/config/database.yml config/
```

Nyní když aplikaci spustíme, vypadá prostředí funkčně, a část aplikace funguje. Nyní budeme dohledávat problémy a ručně řešit. Nesmíme zapomenout na následující:

- obnovit/opravit `public/.htaccess`
- v řadičích a jinde zaměnit `@params[...]` za `params[...]`
-
-

47.27.13. Railscast Episode 189: Embedded Associations

* ***Attributy:*** *id="railscast-189"*
* *Poznámky k vysílání.*

Příklad 47-28. User Model

```
class User < ActiveRecord::Base
    acts_as_authentic
    has_many :articles
    has_many :comments
    has_many :assignments
    has_many :roles, :through => :assignments # many-to-many

    def role_symbols
        roles.map do |role|
            role.name.underscore.to_sym
        end
    end
end
```

Příklad 47-29. `authorization_rules.rb`

```
authorization do
    role :admin do
        has_permission_on [:articles, :comments], :to => [:index, :show, :new, :create, :edit,
    ...
end
```

Protože model/tabula Role je velmi úzce provázána s kódem a jakékoliv změny v tabulce musí být doplněny odpovídajícími změnami v kódu, ztrácí se úplně výhoda použití datové tabulky. Následující postup je o tom jak tuto tabulku odstranit proměnit ve změny kódu. Nejdříve se zruší nepotřebné relace v modelu uživatele `User`.

Příklad 47-30. User Model `user.rb`

```
class User < ActiveRecord::Base
    acts_as_authentic
    has_many :articles
    has_many :comments

    ROLES = %w[admin moderator author]
    def role_symbols
```

```
            [role.to_sym]
        end
    end
```

Pomocí migrací

```
$ script/generate migration add_role_to_users role:string
$ rake db:migrate
```

* *Jak generátor pozná jak vytvořit migraci. Tedy opravdu rozpozná z názvu migrace že ten sloupec role má přidat do tabulky user?*

Opravíme příslušné pohledy kde se zobrazují role.

Příklad 47-31. new_html.erb

```
...
  <p>
    <%= f.label :role %><br />
    <%= f.collection_select :role, User::ROLES, :to_s, :humanize %>
  </p>
...
```

Uvedený postup změní relaci mezi uživatelem a rolí na `one-to-many` (jedna role, více uživatelů). Pokud chceme zachovat vztah `many-to-many`, tedy že uživatel může mít přiřazeno více rolí, musíme to udělat jinak. Nejprve sloupeček role v databázi. Ten je třeba nahradit, protože už nám nestačí jedna role. Můžeme použít například sloupeček `roles` do kterého „serializujeme“ pole/seznam s více rolemi.

Příklad 47-32. `user.rb`

```
...
    serialize :roles
...
```

Tento přístup přináší problémy při práci s uživateli, protože je třeba ošetřit sezam rolí, zakódovaný do sloupečku `roles`. Ukázkou takového problému je, pokud budeme chtít vybrat z uživatelů například všechny kteří jsou administrátory.

Řešením tohoto problému je použití bitové masky. Výhodou je že uložená informace nezabírá mnoho místa, snadno se podle ní vyhledává. Bitovou masku reprezentujeme celým číslem (`integer`)

```
$ script/generate migration add_roles_mask_to_users roles_mask:integer
$ rake db:migrate
```

Upravíme model uživatele

Příklad 47-33. `user.rb`

```
...
    ROLES = %w[admin moderator author]

    def roles=(roles)
        self.roles_mask = (roles & ROLES).map {|r| 2**ROLES.index(r)}.sum
    end

    def roles
        ROLES.reject {|r| ((roles_mask || 0) & 2**ROLES.index(r)).zero? }
```

```
    end

    def role_symbols
        roles.map(&:to_sym)
    end
    ...
```

Upravíme pohledy

Příklad 47-34. new_html.erb

```
...
  <p>
    <%= f.label :roles %><br />
    <% for role in User::ROLES %>
      <%= check_box_tag "user[roles][]", role, @user.roles.include?(role) >
      <%=h role.humanize ><br />
    <% end %>
    <%= hidden_field_tag "user[roles][]", "" %>
  </p>
...
```

Vyhledávání uživatelů s danou rolí upravíme v modelu

Příklad 47-35. `user.rb`

```
...
    named_scope :with_role, lambda {|role| {:conditions => "roles_mask & #{2**ROLES.index(role
...
```

47.28. Řešené problémy

V této části uvádím řešené problémy s komplexním pohledem.

47.28.1. Menu

Odkazy:

- CSS MENU MAKER (http://cssmenumaker.com/)
- Adding default html option to 'link_to' (http://refactormycode.com/codes/242-adding-default-html-option-to-link_to)
- Tip: Overriding link_to to accept a block (http://opensoul.org/2006/08/04/tip-overriding-link_to-to-accept-a-block/)
- CSS Menus (http://www.13styles.com/)
- Using Ruby Blocks for Custom Rails Tags (http://www.railsforum.com/viewtopic.php?id=318)

V aplikaci běžně potřebujeme navigační menu. Seznam odkazů na jednotlivé části aplikace. V dobách HTML se to řešilo různě, neb design menu byl dán html kódem. To je ale již zbytečné, protože můžeme použít CSS. Jak tedy na menu? Nejdříve samotné položky menu. Napíšeme je do tagu `ul`:

```
<ul>
    <li><%= link_to 'Lidé', :controller => 'people'%></li>
    <li><%= link_to 'Firmy', :controller => 'firmy'%></li>
    ...
</ul>
```

První vylepšení které na našem HTML provedem, je změna zobrazení aktuální položky. Chceme aby když jsem na stránce firem, tak aby odkaz na firmy (druhý li element) byl neaktivní. Toho můžeme dosáhnout podmíněným link_to. Například:

```
<% if podmínka %>
    <%= link_to 'Firmy', :controller => 'firmy' %>
<% else %>
    Firmy
<% end %>
```

Tento zápis nám dává plnou kontrolu nad tím, co se bude zobrazovat. Ovšem pro většinu případů je příliš dlouhý. Místo něj můžeme použít metodu `link_to` s podmínkou:

link_to_if(*podmínka*, ...)
link_to_unless(*podmínka*, ...)

Máme ovšem k dispozici ještě silnější metodu a to `link_to_unless_current`. S její pomocí přepíšeme původní kód na:

```
<ul>
    <li><%= link_to_unless_current 'Lidé', :controller => 'people'%></li>
    <li><%= link_to_unless_current 'Firmy', :controller => 'firmy'%></li>
    ...
</ul>
```

To jakým způsobem se bude menu zobrazovat pak řídíme pomocí stylesheetu. Abychom ho mohli napsat, potřebujeme si menu nějak označit. Pokud vyjdeme z toho že na stránce je právě jendo hlavní menu, můžeme je vepsat do tagu `div`:

```
<div id="menu">
    <ul>
        ...
    </ul>
</div>
```

K takto vytvořenému HTML kódu nyní dopíšeme CSS. Můžeme si pomoct a nechat si CSS vytvořit například pomocí CSS Menu Makeru (http://cssmenumaker.com/), či si jej napsat sami například podle Horizontal and Vertical CSS Menu Tutorial (http://www.seoconsultants.com/css/menus/tutorial/).

47.28.2. Authorization

Odkazy:

- Railscasts Episode 188: Declarative Authorization ()
-

47.28.2.1. declarative_authorization

Odkazy:

- declarative_authorization (http://github.com/stffn/declarative_authorization) na github

Příklad 47-36. `environment.rb`

```
Rails::Initailizer.run do |config|
  ...
  config.gem "declarative_authorization", :source => "http://gemcutter.org"
  ...
```

```
$ rake gems:install
```

Vytvoříme si soubor `config/authorization_rules.rb`

```
authorization do
    role :admin do
        has_persmission_on [:articles, :comments], :to => [:index, :show, :new, :create, :edit
    end
end
```

V příslušném datovém modelu pak

```
# File: app/models/user.rb
class User < ActiveRecord::Base
    acts_as_authentic  # Použit gem authentic
    ...
    has_many :roles, :through => :assignemnts

    def role_symbols
        # [:admin] if admin?
        roles.map do |role|
     role.name.underscore.to_sym
        end
    end
end

class ApplicationController < ActionController::Base
    include Authentication
    helper :all
    protect_from_frogery
    before_filter {|c| Authorization.current_user = c.current_user}
end

class ArticlesController < ApplicationController
    filter_resource_access
end

# File: config/authorization_rules.rb
authorization do
    role :admin do
        has_persmission_on [:articles, :comments], :to => [:index, :show, :new, :create, :edit
    end

    role :quest do
        has_persmission_on :articles, :to => [:index, :show]
```

```
     has_persmission_on :comments, :to => [:new, :create]
        end
    end
```

Změny v pohledu

```
    # File: .../show.html.erb
    ...
        <% if permitted_to? :edit, @article %>
          <%= link_to "Edit", edit_article_path(@article) %>
        <% end %>
    ...

    # File: config/authorization_rules.rb
    ...
        role :quest do
            has_persmission_on :articles, :to => [:index, :show]
     has_persmission_on :comments, :to => [:new, :create]
        has_persmission_on :comments, :to => [:edit, :update] do
                if_attribute :user => is { user }
            end
        end
    ...

    # File: .../application_controller.rb
    class ApplicationController < ActionController::Base
        include Authentication
        helper :all
        protect_from_frogery
        before_filter {|c| Authorization.current_user = c.current_user}

        protected
        def permission_denied
            flash[:error] = "Litujeme, ale nemáte oprávnění přístupu k té stránce."
            redirect_ro root_url
        end
    end
```

47.29. Plugins

* ***Attributy:*** *id="rails.plugins"*

47.29.1. ActiveScaffold

* *Umí to hezké věci, ale generuje to nehezký kód s tabulkami vloženými do tabulek. Rovněž stylování vypadá nehezky.*

Odkazy:

- activescaffold / active_scaffold (http://github.com/activescaffold/active_scaffold)
- ACTIVE SCAFFOLD (http://activescaffold.com/)
-

47.29.2. Drag-Drop Sortable for ActiveScaffold

* *Plugin do Pluginu 47.29.1.*

  Odkazy:

  - activescaffold / active_scaffold_sortable (http://github.com/activescaffold/active_scaffold_sortable/) na Github [2010-04-16]
  -

```
$ script/plugin install git://github.com/activescaffold/active_scaffold_sortable.git
```

47.29.3. Acts As List

* ***Attributy:*** *id="act_as_list"*
* *Plugin který do ActiveRecord zavádí pořadí záznamů. Vyžaduje přidání sloupce do tabulky který slouží k uložení pořadí a nabízí řadu metod pro manipulaci s pořadím záznamů.*

  Odkazy:

  - Acts As List Plugin (http://www.railslodge.com/plugins/149-acts-as-list)
  - ryanb / acts-as-list (http://github.com/ryanb/acts-as-list) [2008-07-22 version 0.1.2]
  - rails / acts_as_list (http://github.com/rails/acts_as_list) [2007-10-12]
  - goncalossilva / acts_as_list (http://github.com/goncalossilva/acts_as_list) [2010-04-23]
  - coroutine / acts_as_list_with_sti_support (http://github.com/coroutine/acts_as_list_with_sti_support) [2010-03-25]
  -

47.29.4. calendar_date_select

*

  Odkazy:

  - github timcharper / calendar_date_select (http://github.com/timcharper/calendar_date_select)
  -
  -

47.29.5. event_calendar

*

  Odkazy:

  - github elevation / event_calendar (http://github.com/elevation/event_calendar)
  -

47.29.6. Datepicker

* ***Attributy:***
* *používá jQuery a unobtrusive javascript*

  Odkazy:

  - Datepicker (http://jqueryui.com/demos/datepicker/)
  -

47.29.7. table_builder

*

Odkazy:

- github p8 / table_builder (http://github.com/p8/table_builder)
-
-

```
$ script/plugin install git://github.com/p8/table_builder.git
```

47.30. Recepty

Sbírka příkladů a ukázek

* ***Attributy:*** *id="rails.recipes"*

Odkazy:

-
-

Queue:

- Table Drag and Drop JQuery plugin (http://www.isocra.com/2008/02/table-drag-and-drop-jquery-plugin/)
- shortcut / unobtrusively-sortable (http://github.com/shortcut/unobtrusively-sortable)
-
-

47.30.1. Ajax style drag-n-drop sorting

* ***Attributy:***

Odkazy:

- How to do Ajax style drag-n-drop sorting with Rails (http://harryche2008.wordpress.com/2008/07/19/how-to-do-ajax-style-drag-n-drop-sorting-with-rails/) [2008-07-19]
- rails / acts_as_list (http://github.com/rails/acts_as_list) [2007-10-12]
-
- Rails - Drag and Drop Sorting in a Table (http://craiccomputing.blogspot.com/2008/11/rails-drag-and-drop-sorting-in-table.html) [2008-11-26]
- Rails / Sortable lists (http://whynotwiki.com/Rails_/_Sortable_lists)
-

Tento recept používá plugin 47.29.3.

Ukázkový postup

1. Nejdříve potřebujeme nainstlovat plugin 47.29.3.

   ```
   $ script/plugin install acts_as_list
   ```

2. Přidáme do tabulky, která má mít definováno pořadí, sloupeček který tuto informaci bude udržovat. Pokud ještě tabulku nemáme, do migrace přidáme definici sloupce.

   ```
   create_table :produkty do |t|
     ...
     t.integer :position
     ...
   ```

```
    end
```

Pokud tabulku již máme, napíšeme si migraci která tento sloupec přidá. V této ukázce se tabulka jmenuje `produkty` a pro sloupec s pořadím jsem zvolil název `position`.

```
$ script/generate migration add_position_to_produkty
```

```
class AddPositionToProdukty < ActiveRecord::Migration
  def self.up
    add_column :produkty, position, :integer
    counter = 1
    Produkt.all.each do |produkt|
      produkt.position = counter
      produkt.save
      counter += 1
    end
  end
  def self.down
    remove_column :produkty, :position
  end
end
```

```
$ rake db:migrate
```

3. Do modelu naší tabulky přidáme deklaraci acts_as_list

```
class Produkty < ActiveRecord::Base
  acts_as_list
  named_scope :ordered, :order => :position
end
```

4. A teď zobrazení našich produktů.

```
  <tbody id="produkty" style="cursor:move">
    <% @produkty.each do |produkt| %>
      <% id="item_<%= produkt.id %>"%>
      ...
  </tbody>
</table>
<%= sortable_element 'produkty',
                     :url => { :action => 'sort', :id => @produkty },
                     :complete => visual_effect(:highligth, 'produkty'),
                     :tag => 'tr'
%>
```

47.30.2. Recept

* ***Attributy:***

  Odkazy:

  •

  •

47.30.2.1.

*

47.30.2.2.

*

Kapitola 48. Rails 3

Odkazy:

- 47.1.2
- Ruby on Rails 3.0 Release Notes (http://guides.rails.info/3_0_release_notes.html)
- Rails 3.0: Beta release (http://weblog.rubyonrails.org/2010/2/5/rails-3-0-beta-release/) [2010-02-05]
- Rails 3 Reading Material (http://mediumexposure.com/rails-3-reading-material/) by hakunin [2010-01-21]
-
-

48.1. Active Record Queries in Rails 3

*

Odkazy:

- http://railscasts.com/episodes/202-active-record-queries-in-rails-3
- 215 (http://railscasts.com/episodes/215)
- github / arel ()

```
class Product < ActiveRecord::Base
  belongs_to :category
  scope :discontinued, where(:discontinued => true)
  # scope :cheaper_than, lambda {|price| where("price < ?", price)}

  def self.cheaper_than(price)
    where("products.price < ?", price)
  end

  scope :cheap, cheaper_than(5)
end

class Category < ActiveRecord::Base
  has_many :products
  scope :with_cheap_products, joins(:products) & Product.cheap
end
```

Kapitola 49. Nitro

* *chapter id="nitro"*

Odkazy:

- NITRO (http://www.nitroproject.org/)

S použitím RubyGems je instalce velmi jednoduchá.

```
# gem install nitro
```

Tímto máme nainstalováno.

Vytvoření adresáře se soubory aplikace:

```
$ nitrogen app $(pwd)/adresář
```

Příklad vytvoření aplikace „prvni“:

```
$ nitrogen app $(pwd)/prvni
$ cd prvni
$ chmod u+x run.rb
```

Kapitola 50. Ramaze

Odkazy:

- Ramaze (http://ramaze.net/)
-

Kapitola 51. Web Frameworks

Prostředí pro tvorbu webů

Zde popisuji knihovny a prostředí určená pro web. Od jednoduchýc www serverů přes jednoduché knihovny užitečné při tvorbě stránek až po rozsáhlá komplexní prostředí.

Odkazy na nástroje/knihovny

- Amrita (http://amrita.sourceforge.jp/) — knihovna šablon (template library)
- html-table (http://ruby-miscutils.sourceforge.net/table.html) — An interface for generating HTML Tables with Ruby.
- Arrow (http://www.rubycrafters.com/projects/Arrow/) — Arrow is a framework for building web applications using Apache and mod_ruby.
- Rails (http://www.rubyonrails.org/) — Rails is a web-application framework for the MVC pattern that includes both a template engine, controller framework, and object-relational mapping package. Everything needed to develop web-apps that can run on CGI, FastCGI, and mod_ruby.
- NARF (http://www.narf-lib.org/cgi-bin/wiki.rb) — NARF is a replacement for, and derivative of, the Ruby CGI library.
- htaccess (debian-31r0a-i386-binary-8.iso) — Interface to apache .htaccess and htpasswd files.
- rweb (http://igels.net/ruby/) — CGI Support Library
- Yo (http://yo-lib.rubyforge.org/) — library for easily writing web interfaces to Ruby stuff. It's a bit like CGI
- Ruby/CAPTCHA (http://captcha.rubyforge.org/) — Completely Automated Public Turing Test to Tell Computers and Humans Apart.
- Nemo (http://rubyforge.org/projects/nemo/) — web-application platform that uses object metadata to automatically construct web-interfaces. Runs on top of Michael Neumann's Wee

51.1. `mod_ruby` — spolupráce Ruby a Apache

* *section id="mod_ruby" xreflabel="mod_ruby"*

* *Přesunout tuto sekci do části Programování webových aplikací, případně z ní udělat samostatnou kapitolu v téže části.*

Tato část pojednává o integraci Ruby do Apache. Tedy o tom, jak používat Ruby na straně www serveru Apache.

* *Nezabývat se podrobným popisem eruby. Tento bude/je v samostatné části.*

Zdroje a odkazy:

- mod_ruby the Apache/Ruby integration project (http://modruby.net/)

Chceme-li psát v Ruby www stránky, můžeme používat CGI skripty, tento způsob je historicky první. Nejsou na něj kladeny žádné zvláštní požadavky. Můžeme také použít WWW server napsaný přímo v Ruby (WEBrick, wwwsrv, Nora či jiný)

Ruby lze taky s webovým serverem Appache, a o tom jak na to je tato část. Podstatou integrace je modul `mod_ruby`.

```
LoadModule ruby_module /usr/lib/apache/mod_ruby.so
```

51.1.1. Instalace mod_ruby

Pokud chceme používat eRuby, musím jej instalovat před modulem `mod_ruby` a zohlednit to v konfiguraci modulu. Postup který uvádím s tím počítá.

1. Stáhneme si balíček s mod_ruby a rozbalíme

```
$ cd ~/arch/lang/ruby/mod_ruby
$ get http://www.modruby.net/archive/mod_ruby-1.0.6.tar.gz
$ cd ~/tmpsrc
$ tar xzvf ~/arch/lang/ruby/mod_ruby/mod_ruby-1.0.6.tar.gz
```

2. Nakonfigurujeme pomocí skriptu `./configure.rb` který nám vytvoří `Makefile`.

```
$ cd ~/tmpsrc/mod_ruby-1.0.6
$ ./configure.rb --enable-eruby \
                 --with-apxs=/usr/bin/apxs
                 --with-eruby-includes=$HOME/tmpsrc/eruby-1.0.3 \
                 --with-eruby-libraries=$HOME/tmpsrc/eruby-1.0.3
```

   Budeme-li potřebovat nápovědu k volbám konfiguračního skriptu, získáme ji přepínačem `--help`

```
$ ./configure.rb --help
```

3. Modul přeložíme

```
$ make
```

4. A nainstalujeme

```
$ make install
```

5. Upravíme konfigurační soubor apache `/etc/apache/httpd.conf`. Konfigurace je popsána dále v textu.

6. Restartujeme Apache aby se uplatnily změny v jeho konfiguraci.

```
# /etc/init.d/apache reload
```

51.1.2. Konfigurace mod_ruby

Oproti CGI skriptům, přináší mod_ruby zrychlení, neboť se již pri každém přístupu na stránku nemusí spouštět ruby. Ruby je ve zakompilováno přímo do Apache, nebo ve formě dynamické knihovny zavedeno do paměti.

Příklad 51-1. Konfigurace mod_ruby v `/etc/apache/httpd.conf`

```
### Ruby: mod_ruby
LoadModule ruby_module /home/radek/opt/ruby/lib/mod_ruby.so❶
<IfModule mod_ruby.c>
        ## mod_ruby
        RubyRequire apache/ruby-run
        <FilesMatch ".rbx">                                ❷
                Options +ExecCGI
                SetHandler ruby-object
                RubyHandler Apache::RubyRun.instance
        </FilesMatch>
</IfModule>
```

❶ Zavedení modulu `mod_ruby` do Apache. Je použit modul z aktuální verze Ruby. Toto je zajištěno tak, že adresář `/home/radek/opt/ruby` je symbolickým odkazem na adresář s aktuální verzí, například `/home/radek/opt/ruby-1.8.0-2003-01-23`.

❷ Definice přípony souborů, podle které mod_ruby pozná soubory pro něj určené.

Protože mám ruby zkompilované a nainstalované do vlastního adresáře, musím ještě upravit spouštěcí soubor `/etc/init.d/apache` aby ruby věděl kde má umístněné knihovny. Přidám proto na začátek k exportům řádek

```
export LD_LIBRARY_PATH=/home/radek/opt/ruby/lib
```

51.1.3. FIXME: Použití

A jak se `mod_ruby` používá? V konfiguraci apache jsme uvedli že mod_ruby rozeznává jako své soubory s příponou `.rbx`. Tyto jsou výkonné ruby soubory. Musí být čitelné a spustitelné pro apache

```
$ ls -l /var/www/ruby/modruby/hello.rbx
-rwxr-x---    1 radek    www-data       21 říj 27 20:27 /var/www/ruby/modruby/hello.rbx
```

Ukážeme si jednoduchou stránku

```
  # $Id: handle1.rbx,v 1.1 2003/10/28 16:02:24 radek Exp $
  # $Source: /home/radek/cvs/ruby-book/example/modruby/handle1.rbx,v $
  r = Apache.request
  r.content_type = 'text/html'
5 r.send_http_header
  exit(Apache::OK) if r.header_only?

  puts("The time at the tone is: #{Time.new()}") ;
```

* *Použít informalexample/programlisting/inlinemediaobject/imageobject*

```
  # $Id: headers.rbx,v 1.1 2003/10/28 16:02:24 radek Exp $
  # $Source: /home/radek/cvs/ruby-book/example/modruby/headers.rbx,v $
  r = Apache.request
  r.content_type = 'text/html'
5 r.send_http_header
  exit(Apache::OK) if r.header_only?

  puts '<table border="1">'
  r.headers_in.each_key do |header|
10  puts "<tr><td>#{header}</td><td>#{r.headers_in[header]}</td></tr>"
  end
  puts '</table>';
```

```
  #$Id: request.rbx,v 1.1 2003/10/28 16:02:24 radek Exp $
```

```
r = Apache.request
r.content_type = 'text/html'
r.content_encoding = 'iso-8859-2'
r.send_http_header
exit(Apache::OK) if r.header_only?

r.puts '<h1>Apache.request</h1>'

r.puts <<-EOS
    <h2>request.methods</h2>
    <p>Seznam metod objektu <tt>Apache.request</tt>:<br/>
        #{r.methods.sort.collect{|m| "<tt>#{r.escape_html(m)}</tt>"}.join ", "}
    </p>
    EOS

puts <<-EOS
    <h2>request[]</h2>
    <table border="1">
      <tr><th colspan="2">Hodnoty Apache.request[]</th><tr>
      <tr><th>key</th><th>value</th></tr>
EOS
    r.each_key do |k|
    puts "<tr><td>#{k}</td><td>#{r[k]}</td></tr>"
    end
puts <<-EOS
    </table>
EOS

puts <<EOS
<h2>Hodnoty n<65533>kter<65533>ch atribut<65533> objektu <tt>Apache.request</tt></h2>
<table border="1">
  <tr><th>atribut</th><th>hodnota</th></tr>
  <tr><th>args</th><td><tt>#{r.escape_html(r.args)}</tt></td></tr>
  <tr><th>content_encoding</th><td><tt>#{r.content_encoding.to_s}</tt></td></tr>
</table>
EOS
#  <tr><th>content_encoding</th><td><tt>#{r.escape_html(r.content_encoding)}</tt></td></tr

puts '<h2>request</h2>'
puts '<table border="1">'
r.headers_in.each_key do |header|
  puts "<tr><td>#{header}</td><td>#{r.headers_in[header]}</td></tr>"
end
puts '</table>';
```

51.1.4. Přehled objektů a metod

51.1.4.1. Modul `Apache`

Metody modulu

`add_version_component`

Přidá token do řetězce popisujícího verzi Apache

`chdir_file(str)`

Změní aktuální pracovní adresář serveru na adresář uvedený v cestě k souboru.

`request`

Vrátí aktuální `Apache::Request` objekt.

`server_root`

Vrátí kořenový adresář serveru.

`server_built`

Zjistí řetězec popisující datum sestavení Apache

`server_version`

Zjistí řetězec popisující verzi Apache

`unescape_url(str)`

Dekóduje URL zakódovyný řetězec.

Konstanty

Návratové kódy handleru

`OK`, `DECLINED`, `DONE`

Kódy HTTP odpovědí

`AUTH_REQUIRED`, `BAD_GATEWAY`, `BAD_REQUEST`, `DOCUMENT_FOLLOWS`, `FORBIDDEN`, `HTTP_ACCEPTED`, `HTTP_BAD_GATEWAY`, ...

51.1.4.2. Třída `Apache::Request`

Zapouzdřuje datový typ `request_rec`. Třída je odvozena od třídy `Object` a zahrnuje modul `Enumerable`

`hostname`

vrací jméno jak je zadáno v `URI` nebo `Host:`.

`unparsed_uri`

Vrací surové, nerozdělené `URI`.

`uri`
`uri=` *`str`*

Vrací/Nastavuje cestu vyčtenou z `URI`.

`filename`
`filename=` *`str`*

Vrací/Nastavuje jméno souboru vyčtené z `URI`.

`path_info`
`path_info=` *`str`*

Vrací `PATH_INFO`.

`status`
`status=`

Vrací/Nastavuje číselný kód transakce

`status_line`
`status_line= str`

Vrací/Nastavuje stavový řádek.

`request_time`

Vrací čas kdy byla žádost zadána.

`request_method`

Vrací metodu kterou byla žádost podána `GET`, `HEAD` nebo `POST`.

`method_number`

Vrací metodu dotazu jako celé číslo. Můžeme ji porovnat s konstantami FIXME:.

`header_only?`

Vrací `true` při `HEAD` žádosti.

`allowed`
`allowed=` *`int`*

Vrací/Nastavuje FIXME:

`the_request`

Vrací první řádek dotazu pro potřeby deníku.

`args`

Vrací `QUERY_ARGS`.

`headers_in`

Vrací objekt `Apache::Table`.

`read`([*`len`*])

Přečte *`len`* bajtů od klienta.

```
read([len])
gets([rs])
readline([rs])
readlines([rs])
each([rs]) {|line|...}
each_line([rs]) {|line|...}
each_byte {|ch|...}
getc
readchar
ungetc(ch)
tell
seek(offset, [whence])
rewind
pos
pos= n
eof
eof?
binmode
```

Metody přijímají data od klienty. Fungují podobně jako obdobné metody v `IO`.

`headers_out`

Vrací objekt `Apache::Table`.

`content_type= str`

Vrací objekt `Apache::Table`.

`content_type`

Vrací specifikaci `Content-Type`.

`content_encoding= str`
`content_encoding`

Kódování `Content-Encoding`.

`content_languages= str`
`content_languages`

Specifiakce jazyka `Content-Languages`.

`send_http_header`

Posílá hlavičku odpovědi `HTTP`.

```
write(str)
putc(ch)
print(arg...)
printf(fmt, arg...)
puts(arg...)
<< obj
```

Metody posílají data klientovi. Fungují podobně jako metody v `IO`.

`replace(str)`

Vymění výstupní buffrer s řetězcem *`str`*.

`cancel`

Vyčistí výstupní buffer.

`escape_html(str)`

Zakóduje znaky & " < >.

51.1.4.3. Třída `Apache::Table`

FIXME:

51.2. Používáme WEBrick

* *section id="webrick" xreflabel="WEBrick"*
* *Přesunout tuto sekci do části Programování webových aplikací, případně z ní udělat samostatnou kapitolu v téže části.*

Poznámka: Od verze ruby 1.8.0 je WEBrick jako knihovna součástí ruby.

@query

hash parametrů předaných za otazníkem (...?a=2)

`@query_string`

řetězec parametrů předaných za otazníkem (...?a=2)

Příklad 51-2. Jednoduchý *daytime* server 1

```
#!/usr/bin/env ruby

require 'webrick'

s = WEBrick::GenericServer.new( :Port => 2000 )
trap ("INT") { s.shutdown }
s.start {|sock|
    sock.print(Time.now.to_s + "\r\n")
}
```

Příklad 51-3. Jednoduchý *daytime* server 2

```
#!/usr/bin/env ruby
# $Id: daytime_server2.rb,v 1.1 2002/06/07 07:10:10 radek Exp $
require 'webrick'

class DaytimeServer < WEBrick::GenericServer
    def run(sock)
 sock.print(Time.now.to_s + "\r\n")
    end
end
```

```
s = DaytimeServer.new( :Port => 2000 )
trap ("INT") { s.shutdown }
s.start
```

Příklad 51-4. Jednoduchý *HTTP* server 1

```
#!/usr/bin/env ruby
# $Id: http_server1.rb,v 1.1 2002/06/07 07:10:10 radek Exp $
require 'webrick'
include WEBrick

s = HTTPServer.new( :Port => 2000, :DocumentRoot => Dir::pwd + "/htdocs" )

## mount subdirectories
s.mount("/~gotoyuzo", HTTPServlet::FileHandler, "/home/radek/documents/")
s.mount("/www", HTTPServlet::FileHandler, "/var/www/")

trap ("INT") { s.shutdown }
s.start
```

Příklad 51-5. Jednoduchý *HTTPS* server 1

```
#!/usr/bin/env ruby
# $Id: https_server1.rb,v 1.1 2002/06/07 07:10:11 radek Exp $
require 'webrick'
require 'webrick/https'

s = WEBrick::HTTPServer.new(
    :Port => 2000,
    :DocumentRoot => Dir::pwd + "/htdocs",
    :SSLEnable          => true,
    :SSLVerifyClient    => ::OpenSSL::SSL::VERIFY_NONE,
    :SSLCertName        => [ ["C","JP"],
                             ["O", "WEBrick.Org"],
                             ["CN", "WWW"] ]
)

## mount subdirectories
s.mount("/~gotoyuzo", HTTPServlet::FileHandler, "/home/radek/documents/")
s.mount("/www", HTTPServlet::FileHandler, "/var/www/")

trap ("INT") { s.shutdown }
s.start
```

Příklad 51-6. Jednoduchý *HTTP* server se servletem

```
#!/usr/bin/env ruby
# $Id: http_server2.rb,v 1.1 2002/06/07 07:10:10 radek Exp $
require 'webrick'
include WEBrick
```

```
s = HTTPServer.new( :Port => 2000, :DocumentRoot => Dir::pwd + "/htdocs" )

# HTTPServer#mount(path, servletclass)
#   When a request referring "/hello" is received,
#   then HTTPServer get an instance of servletclass
#   and then call a method named do_"a HTTP method".
class HelloServlet < HTTPServlet::AbstractServlet
    def do_GET(req, res)
 res.body = "<HTML>hello, world.</HTML>"
 res['Content-Type'] = "text/html"
    end
end
s.mount("/hello", HelloServlet)

# HTTPServer#mount_proc(path){|req, res| ...}
#   You can mount also a block by 'mount_proc'.
#   This block is called when GET or POST.
s.mount_proc("/hello/again") {|req, res|
    res.body = "<HTML>hello (again)</HTML>"
    res['Content-Type'] = "text/html"
}

## mount subdirectories
s.mount("/~gotoyuzo", HTTPServlet::FileHandler, "/home/radek/documents/")
s.mount("/www", HTTPServlet::FileHandler, "/var/www/")

trap ("INT") { s.shutdown }
s.start    # Run the server
```

Příklad 51-7. Jednoduchý *HTTP* server spouštěný z `inetd`

```
#!/usr/bin/env ruby
# $Id: httpd.in-1.rb,v 1.1 2002/06/07 07:10:10 radek Exp $

log = open("/var/log/webrick/httpd.log", "a")
STDERR.reopen(log)  # do not send stderr to client.

require 'webrick'
require 'getopts'

getopts nil, 'r:'

sock = TCPSocket.for_fd(0, "w+")  # create TCPSocket from fd.
port = sock.addr[1]

s = WEBrick::HTTPServer.new(
    :DoNotListen    => true,
    :Port           => port,
    :Logger         => WEBrick::Log::new($stderr, WEBrick::Log::DEBUG),
    :DocumentRoot   => $OPT_r || "/var/www"
)
s.run(sock)
```

Příklad 51-8. Jednoduchý *HTTPS* server spouštěný z `inetd`

```
#!/usr/bin/env ruby
# $Id: httpsd.in-1.rb,v 1.1 2002/06/07 07:10:11 radek Exp $
# sample of HTTPS server spawned from inetd

log = open("/var/log/webrick/httpds.log", "a")
STDERR.reopen(log)  # do not send stderr to client.

require 'webrick/https'
require 'getopts'

getopts nil, 'r:'

pkey = cert = cert_name = nil
begin
    data = open(dir + "/conf/sample.key") {|io| io.read}
    pkey = OpenSSL::PKey::RSA.new(data)
    data = open(dir + "/conf/sample.crt") {|io| io.read}
    cert = OpenSSL::X509::Certificate.new(data)
rescue
    $stderr.puts "Switching to use self-signed certificate"
    cert_name = [ ["C","JP"], ["O","WEBrick.Org"], ["CN", "WWW"] ]
end

sock = TCPSocket.for_fd(0, "w+")  # create TCPSocket from fd.
port = sock.addr[1]

s = WEBrick::HTTPServer.new(
    :DoNotListen    => true,
    :Port           => port,
    :Logger         => WEBrick::Log::new($stderr, WEBrick::Log::DEBUG),
    :DocumentRoot   => $OPT_r || "/var/www",
    :SSLEnable      => true,
    :SSLVerifyClient => ::OpenSSL::SSL::VERIFY_NONE,
    :SSLCertificate => cert,
    :SSLPrivateKey  => pkey,
    :SSLCertName    => cert_name
)
s.run(sock)
```

Další příklad je persistentní servlet.

Příklad 51-9. Persistant Servlets

```
#!/usr/local/bin/ruby
    require 'webrick'
    include WEBrick

    s = HTTPServer.new( :Port => 2000 )
    class HelloServlet < HTTPServlet::AbstractServlet

    # Overloads AbstractServlet#get_instance
    # which creates new servant.
    class << self
        def get_instance(*arg)
```

```
            self
        end
    end

    def initialize(server, *options)
        @i = 0
    end

    def do_GET(req, res)
        res.body = "<HTML>Count #{@i}</HTML>"
        res['Content-Type'] = "text/html"
        @i += 1
    end
end
s.mount("/hello", HelloServlet)

s.mount("/hello", HelloServlet.new)

trap("INT"){ s.shutdown }
s.start
```

51.2.1. Div a Tofu

Příklad 51-10. `FirstDiv.rb`

```
#!/usr/bin/env ruby
# $Id: FirstDiv.rb,v 1.2 2002/09/15 07:21:55 radek Exp $
#
# Copyright (C) 2002 Radek Hnilica

$:.unshift('/home/radek/lib/ruby')
require 'tofu/tofulet'
require 'drb/drb'


def setup_bartender(monolithic=true)
    if monolithic
        require 'yourapp'
        Tofu::Bartender.new(YourTofuSession)
    else
        DRbObject.new(nil, 'druby://localhost:7642')
    end
end

def main(monolithic=true)
    DRb.start_service
    logger = WEBrick::Log::new($stderr, WEBrick::Log::DEBUG)

    s = WEBrick::HTTPServer.new(
        :Port       => 2001,
        :Loger      => logger
    )

    bartender = setup_bartender(monolithic)
    s.mount("/div", WEBrick::Tofulet, bartender)
```

```
    trap("INT") { s.shutdown }
    s.start
end

###
main
```

Příklad 51-11. `yourapp.rb`

```
#!/usr/bin/env ruby
# $Id: yourapp.rb,v 1.1 2002/06/07 14:02:21 radek Exp $

require 'div/div'
require 'div/tofusession'

class SumTotal
  def initialize
    @history = []
    @amount = 0
  end
  attr_reader :history, :amount

  def add(value)
    f = value.to_f
    @history.push(f)
    @amount += f
  end

  def undo
    tail = @history.pop
    return unless tail
    @amount -= tail
  end
end

class SumDiv < Div::Div
  set_erb('sum.erb')

  def initialize(session)
    super(session)
    @model = SumTotal.new
  end

  def do_add(context, params)
    value, = params['value']
    @model.add(value)
  end

  def do_reset(context, params)
    @model = SumTotal.new
  end

  def do_undo(context, params)
    @model.undo
  end
```

```
end

class BaseDiv < Div::Div
  set_erb('base.erb')

  def initialize(session)
    super(session)
    @sum = SumDiv.new(session)
  end
end

class YourTofuSession < Div::TofuSession
  def initialize(bartender, hint=nil)
    super(bartender, hint)
    @base = BaseDiv.new(self)
  end

  def do_GET(context)
    update_div(context)
    context.res_header('content-type', 'text/html; charset=euc-jp')
    context.res_body(@base.to_html(context))
  end
end
```

Příklad 51-12. `base.erb`

```
<html>
  <head>
    <title>First App</title>
  </head>
  <body>
    <h1>First app</h1>
    <p><%=h Time.now%></p>
    <p><%= @sum.to_div(context) %></p>
  </body>
</html>
```

Příklad 51-13. `sum.erb`

```
<%=form('add', context)%>
<table>
<% @model.history.each do |v| %>
<tr><td> </td><td align='right'><%=h v%></td><td> </td></tr>
<% end %>
<tr><th>total</th><th align='right'><%=h @model.amount%></th><td> </td></tr>
<tr>
<th align='right'>add</th>
<th><input size="10" type="text" name="value" value="" /></th>
<th><input type="submit" name="Add" value="Add"/></th>
</tr>
<tr><td align="right" colspan="3"><%=a('undo', {}, context)%>undo</a></td></tr>
<tr><td align="right" colspan="3"><%=a('reset', {}, context)%>reset</a></td></tr>
</table>
</form>
```

51.2.2. Jednoduchý Webový aplikační server

Příklad Webového aplikačního server spolupracujícího z databází.

Zadání.

Server je velmi jednoduchý a spravuje jen jednu tabulku uživatelů. O každém uživateli si pamatujeme jeho jméno, identifikační číslo a pár dalších údajů.

- přidání nového záznamu
- vyhledání záznamu podle kritérií
- mazání záznamu
- opravy záznamu

51.2.2.1. Struktura databáze

Databáze obsahuje jen jednu tabulku s názvem `Person`. Tato má následující pole

Id

identifikační číslo člověka, PIN

FirstName

křestní jméno

LastName

příjmení

Email

elektronická poštovní adresa

Příklad 51-14. SQL skript vytvářející tabulku `Person`

```
--*- coding:utf-8; -*-
-- $Id: create-tables.sql,v 1.2 2002/07/09 13:26:00 radek Exp $
-- Vytvo<65533>en<65533> tabulek se v<65533>emi n<65533>le<65533>itostmi.
-- Copyright (C) 2002 Radek Hnilica
-- All rights reserved.


-- Tabulka Person
-- DROP TABLE Person;      -- Odstran<65533>n<65533> tabulky p<65533>ed novou definic<65533>
CREATE TABLE Person (
 Id  INTEGER PRIMARY KEY,
 FirstName VARCHAR(30),
 LastName VARCHAR(30),
 Email  VARCHAR(40)
);
```

Příklad 51-15. SQL skript pro vytvoření databáze `app1`

```
#!/bin/sh
# -*- coding:utf-8; -*-
# $Id: recreate-database,v 1.3 2002/09/23 20:49:08 radek Exp $
# $Source: /home/radek/cvs/ruby-book/example/net/webrick/app1/recreate-database,v $
# Vytvoření (znovuvytvoření) tabulek
# Copyright (C) 2002 Radek Hnilica
# All rights reserved.

DBNAME=sqldb

# Odstranění původní databáze
rm $DBNAME

# Vytvoření databáze a tabulek
sqlite $DBNAME <create-tables.sql

# Import dat
sqlite $DBNAME <import-data.sql
```

Zapouzdření řádku v tabulce do objektu

Příklad 51-16. Třída Person

```
class Person
end
```

Příklad 51-17. Webový server `httpd.rb`

```
#!/usr/bin/env ruby
# -*- coding:utf-8; -*-
# $Id: httpd.rb,v 1.2 2002/09/15 07:21:55 radek Exp $
# Ruční spouštění aplikačního serveru
# Copyright (C) 2002 Radek Hnilica
# All rights reserved.

$:.unshift('/home/radek/lib/ruby')
require 'webrick'
require 'PersonServlet'

# Vytvoření serveru
server =  WEBrick::HTTPServer.new(
    :Port =>    3002
    #:Logger => WEBrick::Log::new($stderr, WEBrick::Log::DEBUG)
)

# Registrace servletů
server.mount("/person", PersonServlet)

# Register shutdown code
trap('INT') do
    server.shutdown
end

server.start   # Let the server serviceing
```

Příklad 51-18. Webový server `httpd-in.rb` spouštěný přes `inetd`

```
#!/usr/bin/env /home/radek/bin/ruby
# -*- coding:utf-8 -*-
# $Id: httpd-in.rb,v 1.1 2002/09/23 20:49:07 radek Exp $
# $Source: /home/radek/cvs/ruby-book/example/net/webrick/app1/httpd-in.rb,v $
# Server spouštěný z inetd
# Copyright (C) 2002 Radek Hnilica

# Insert following line in /etc/inetd.conf
# 3001 stream tcp nowait radek \
# /home/radek/document/book/language/ruby-book/example/\
# net/webrick/app1/httpd-in.rb

APPROOT="/home/radek/document/book/language/ruby-book/example/net/webrick/app1"

$:.unshift('/home/radek/lib/ruby')
$:.unshift(APPROOT)

log = open("#{APPROOT}/http.log", "a")
STDERR.reopen(log)  # do not send stderr to client.

require 'webrick'
require 'getopts'
require 'PersonServlet'

getopts nil, 'r:'

sock = TCPSocket.for_fd(0) # create TCPSocket from fd.
port = sock.addr[1]

server = WEBrick::HTTPServer.new(
    :DoNotListen    => true,
    :Port           => port,
    :Logger         => WEBrick::Log::new($stderr, WEBrick::Log::DEBUG),
    :DocumentRoot   => $OPT_r || "/var/www"
)

# Registrace servletů
server.mount("/person", PersonServlet)

# Spuštění serveru
server.run(sock)
```

Příklad 51-19. Servlet `PersonServlet.rb`

```
#!/usr/bin/env ruby
# -*- coding:utf-8; -*-
# $Id: PersonServlet.rb,v 1.2 2002/09/23 20:49:07 radek Exp $
# Servlet Person
# Copyright (C) 2002 Radek Hnilica
# All rights reserved.

require 'webrick'

# ?show=all       - zobrazi seznam všech osob
# ?show=one&id=9  - zobrazí kartu jedné osoby
```

```
class PersonServlet < WEBrick::HTTPServlet::AbstractServlet
    #require_path_info false

    def page_header
 <<EOT
<HTML>
  <HEAD>
  </HEAD>
  <BODY>
EOT
    end

    def page_footer
 <<EOT
  </BODY>
</HTML>
EOT
    end

    def do_GET(req, res)
        # Společná nastavení
        res['Content-Type'] = "text/html; charset=iso-8859-2"

        # Podle parametru se rozhodneme kterou stranku zobrazime
        if req.query['show'] == "all"
            res.body << "#{show_all_page(req)}"
 elsif req.query['show'] == "one"
     res.body << "#{show_one_page(req)}"
        else
            res.body << <<EOT
<HTML>
  <HEAD>
  </HEAD>
  <BODY>
    ERROR
  </BODY>
</HTML>
EOT
        end
    end

    def show_all_page(req)
 <<EOT
#{page_header()}
<H1>Show All</H1>
#{page_footer}
EOT
    end

    def show_one_page(req)
 <<EOT
#{page_header()}
<H1>Show One</H1>
#{page_footer}
EOT
    end
```

```
end
```

51.2.3. Drobné příklady k serveru WEBrick

51.2.3.1. Přesměrování dotazu

```
require 'webrick'
s = WEBrick::HTTPServer.new(:Port => 2002)
s.mount_proc("/") {|req,res|
    res.body = "http://#{req['host']}#{req.request_uri}";
}
s.start
```

51.3. eRuby

* *section id="eruby" xreflabel="eRuby"*
* *Věci z této sekce byly převedeny do kapitoly **eRuby**.*

FIXME:

51.4. Borges

* *section id="borges" xreflabel="Borges"*
* *V době psaní techto poznámek byl Borges ve verzi 1.0.0-alpha2.*

Sekce pojednávající o webovém aplikačním serveru Borges. Tento je portací Seaside2 do Ruby. Popisována je verze 1.0.0 a vyšší.

Zdroje a odkazy:

- Borges (http://segment7.net/ruby-code/borges/borges.html)
- Webplayer napsán v Borges (http://segment7.net/ruby-code/webplayer/webplayer.html)
- stephensykes.com (http://stephensykes.com/blog_perm.html?40)
- Borges na RubyForge (http://rubyforge.org/projects/borges/)
- . ()

ToDo list

1. Dopsat: Integrace Borges a Apache
2. Dopsat:

Borges je aplikační knihovna pro konstrukci dynamických webových stránek. Vychází z projektu Seaside jenž je napsán jazykem Smalltalk.

Web application framework written in smalltalk

- Web application framework written in Smalltalk
- Linear flow control
- Components call and return from each other
- Uses continuations to support backtracking
- Reusable, embeddable components

51.4.1. Instalace a konfigurace

FIXME: popsat instalaci a konfiguraci. Není příliš odlišná od předchozí verze 0.2.x.

51.4.2. Kostra aplikace

Ukažme si nyní nejednodušší aplikaci. Jedná so o obligátní program „Hello“.

Příklad 51-20. Jednoduchá aplikace

```
#!/usr/bin/env ruby
# $Id: hello1.rb,v 1.2 2003/12/08 18:40:07 radek Exp $
# $Source: /home/radek/cvs/ruby-book/example/net/web/borges/hello1.rb,v $
# Simple Hello1 application
# Copyright (c) 2003 Radek Hnilica
# License: See Ruby license or GPL

require 'Borges'

module Hello
  class Page < Borges::Component
    def render_content_on(r)
      r.heading "Hello, world"
    end
    register_application 'hello1'
  end
end

if $0 == __FILE__ then
  require 'Borges/WEBrick'
  Borges::WEBrickServlet.start
end

### Keep this comment at the end of the file
#Local variables:
#ruby-indent-level:2
#End:
```

Aplikaci spustíme

```
$ ./hello1.rb
[2003-12-08 13:41:13] INFO  WEBrick 1.3.1
[2003-12-08 13:41:13] INFO  ruby 1.8.1 (2003-10-31) [i686-linux]
[2003-12-08 13:41:18] INFO  WEBrick::HTTPServer#start: pid=9678 port=7000
```

A na její stránku podíváme

```
$ galeon -n localhost:7000/borges/hello1
```

Aplikace funguje a nyní si ji upravíme pro Apache. Vytvoříme stránku

Příklad 51-21. Stránka pro Apache

```
#!/usr/bin/env ruby
# $Id: hello1.rbx,v 1.1 2005/10/04 08:52:07 radek Exp $
# $Source: /home/radek/cvs/ruby-book/example/net/web/borges/hello1.rbx,v $
require 'Borges/ApacheDRbClient'
DRb.start_service
Borges::ApacheDRbClient.handle_request('druby://127.0.0.1:7001')
```

Příklad 51-22. Server aplikace hello1

```
#!/usr/bin/env ruby
# $Id: hello1.srv,v 1.1 2005/10/04 08:52:07 radek Exp $
# $Source: /home/radek/cvs/ruby-book/example/net/web/borges/hello1.srv,v $
require 'hello1.rb'
require 'Borges/ApacheDRbServer'

#Borges::register_application("borges/hello.rbx", Hello)
Borges::ApacheDRbServer.start('druby://127.0.0.1:7001')
Borges::ApacheDRbServer.install_INT_handler
puts "Borges DRb Server listening on #{DRb.uri}"
puts "Press Ctrl-c to quit"
DRb.thread.join
```

Metoda `register_application` použitá k registraci aplikace je definována v třídě Třída Borgess::Controller.

FIXME:

51.4.3. Borges zevnitř

Popis jak Borges funguje.

Malý obrázek. **FIXME:**Tento obrázek ještě není hotový.

51.4.3.1. URL

URL kterým se odkazujeme na Borges z webovského prohlížeče je bodem, ukazatelem do běžící aplikace. Ukažme si takové URL

/borges/hello1/@NnbQABleeEWGcOUK/SsLqCRUe

Rozložíme-li si toto URL podle lomítek na části, dostaneme

`/borges`

Kořen aplikací, *mount point*.

`/hello1`

Jméno aplikace.

`/@NnbQABleeEWGcOUK`

Klíč sezení.

`/SsLqCRUe`

Klíč akce.

51.4.3.2. Request Handling

Request Handler je část odpovídající na žádost (*Request*) vracející v odpovědi (*Response*). Soubory jsou soustředěny v adresáři `lib/Borges/RequestHandler/`.

- `Application.rb`
- `Dispatcher.rb` — Rozhoduje o tom, který ovladač (*handler*) pra daný dotaz *request*) má být použit.
- `DocumentHandler.rb`
- `NotFoundHandler.rb` — Obsluhuje žádost pro niž nebyl nalezen žádný *handler*.
- `Registry.rb`
- `RequestHandler.rb`
- `Session.rb`

1. Dispatcher
2. Application
3. session or Handler

51.4.3.2.1. Dispatcher

Aplikace se registruje v dispečerovi. Dispečer pak určuje která aplikace se volá.

51.4.3.2.2. Application

Aplikace udržuje sezení (*Session*) a ovladače (*Handler*) pro aplikaci (*Application*). Vytváří nová sezení a uklízí stará jimž vypršela doba platnosti (*expired Sessions*)

51.4.3.2.3. Session

Udržuje informaci o sezení (seanci) uživatele. Je defiována v souboru `Session/ControllerSession.rb`

Příklad použití. Pro projekt Sbirky potřebuji udržovat informaci o uživatelově jméně. Definuji si tedy ve třídě `Session` atribut `login_name`.

```
class Session < Borges::ControllerSession
    attr_accessor :login_name # Přihlašovací jméno uživatele
end #class Session
```

51.4.3.2.4. TaskFrame

FIXME:

51.4.3.2.5. Request Processing

- Block for action key is retrieved.
- Block called if user clicked anchor.
- Form fields updated if user submitted form.
- Continuation saved for backtracking.
- Page rendered for user.

51.4.4. Přehled tříd, objektů, modulů, metod, ...

51.4.4.1. Třída `Borges::Filter`

Základní předek všech filtrovacích tříd.

FIXME:

Seznam metod

`handle_request_in`(*`req`*, *`session`*)

FIXME:

51.4.4.2. Třída `Borges::Controller`

* *section id="Borges.Controller" xreflabel="Třída Borgess::Controller"*

Je definována v souboru `Controller/Controller.rb`

- Reusable
- Renderable
- Embeddable

Třída `Controller` je stavebním kamenem aplikace.

Z této třídy dědí třídy Třída Borges::Component a Třída Borges::Task

Metody třídy

`register_application`(*`app_name`*, *`session_class`*=`default_session_class`)

Registruje aplikaci.

Metody instance

`active_controller`

FIXME:

`answer(`*`val`*`=self)`

Vrací řízení ven z komponenty volajícímu. Tedy objektu jenž tento Controller volal.

`call(`*`controller`*`=nil)`

Předává řízení. Voláním této metody předáme řízení jinému Controlleru. Není-li specifikován žádný kontroler je použit `self`. Jestli `self` neodpoví, nebude pokračování (call/cc) uloženo. Jinak deleguj na předaný kontroler a zavolej ho.

`clear_delegate`

FIXME:

`confirm(`*`str`*`)`

FIXME:

`delegate`
`delegate=(`*`controller`*`)`

FIXME:

FIXME:

51.4.4.3. Třída `Borges::Component`

* *section id="Borges.Component" xreflabel="Třída Borges::Component"*

Je definována v souboru `Controller/Component.rb`

Je potomkem třídy Třída Borgess::Controller

- Like a widget
- Builds the HTML document

Metody instance

`footer`

FIXME:

`header`

FIXME:

`render_all_on`

FIXME:

`render_content_on`

FIXME:

`render_footer_on`

FIXME:

`render_head_elements_on`

FIXME:

`render_header_on`

FIXME:

51.4.4.4. Třída `Borges::Request` — vnitřní reprezentace HTTP dotazu

Tato třída obsahuje a zapouzdřuje HTTP dotaz a jeho části.

FIXME:

* *FIXME: popsat attributy: `action_key`, `handler_key`, cookies, fields, headers, path, url, username a password.*

Atributy instance

`action_key`

FIXME:

`handler_key`

FIXME:

`cookies`

FIXME:

`fields`

FIXME:

`headers`

FIXME:

`path`

FIXME:

`url`

FIXME:

`username`

FIXME:

`password`

FIXME:

Metody

new(*url*, *headers*, *fields*, *cookies*)

Metoda new slouží pro vytváření objektů reprezentujících HTTP dotaz a současně je jediným způsobem jak změnit atributy dotazu. Všechny atributy jsou totiž publikovány jen pro čtení. Jednotlivé parametry matody znamenají:

- *url* — URL bez jména hostu a čísla portu
- *headers* —
- *fields* —
- *cookies* —

> **Varování**
>
> Prvním parametr *url* obsahuje cestu bez jména hostu.

51.4.4.5. Třída `Borges::Session`

Objekt sezení. Udržuje v sobě všechny informace o sezení uživatele.

respond(&*block*)

FIXME

```
def respond(&block)
  request = callcc do |cc|
    return_response(block.call(action_url_for_continuation(cc)))
  end

  @filters.contents.each do |ea|
    ea.handle_request_in(request, self)
  end

  return request
end
```

51.4.4.6. Třída `Borges::Task`

* *section id="Borges.Task" xreflabel="Třída Borges::Task"*

Je definována v souboru `Controller/Task.rb`

Je potomkem třídy Třída Borgess::Controller

- Guides a user through an operation
- Embeded in a Component
- Typically a TaskFrame

FIXME:

Metody instance

`render_with`

FIXME:

51.4.4.7. Třída `Borges::TaskFrame`

* *section id="Borges.TaskFrame" xreflabel="Třída Borges::TaskFrame"*

Je definována v souboru `Component/TaskFrame.rb`

Je potomkem třídy Třída Borges::Component

Metody instance

`go`

Tato metoda není ve třídě definována ale defunujeme ji v potomcích třídy. Slouží jako spouštěcí metoda.

`call`

FIXME:

`render_content_on`

FIXME:

51.4.4.8. Třída `Borges::ToolFrame`

* *section id="Borges.ToolFrame" xreflabel="Třída Borges::ToolFrame"*

Je definována v souboru `Component/ToolFrame.rb`

Je potomkem třídy Třída Borges::Component

FIXME:

Metody instance

`actions`

FIXME:

51.5. Borges 0.2.x

* *section id="borges02" xreflabel="Borges 0.2.x"*

* *V době psaní techto poznámek byl Borges ve verzi 0.1.0, 0.1.1, 0.2.0.*

Tato sekce pojednává o původní vývojové verzi projektu Borges ve verzi 0.2.x. Řada vlastností Borges se v novějších verzí změnila a přibyla. Tato sekce zde zůstává jen z historických důvod.

Zdroje a odkazy:

- Borges (http://segment7.net/ruby-code/borges/borges.html)
- Webplayer napsán v Borges (http://segment7.net/ruby-code/webplayer/webplayer.html)
- stephensykes.com (http://stephensykes.com/blog_perm.html?40)

ToDo list

1. Dopsat: Integrace Borges a Apache
2. Dopsat:

Borges je aplikační knihovna pro konstrukci dynamických webových stránek. Vychází z projektů IOWA a Seaside.

51.5.1. Instalace s sprovoznění

* *section id="borges02.install" xreflabel="Instalace s sprovoznění"*

Postup instalace

1. Stáhneme si zdroje balíčku Borges.

   ```
   $ wget http://segment7.net/ruby-code/borges/borges-0.2.0.tar.gz
   ```

2. Rozbalíme

   ```
   $ tar ~/arch/lang/ruby/borges/borges-0.2.0.tar.gz
   ```

3. Nainstalujeme

   ```
   $ ruby install.rb
   ```

Pro úspěšné použití budeme ještě potřebovat WWW server. Můžeme použít Apache v kombinaci s mod_ruby a eRuby, nebo WEBrick.

51.5.1.1. Apache

FIXME:

Propojení s Apache funguje prostřednictvím dRuby. Spuštěná aplikace je dRuby server a do Apache je připojena stránkou dRuby klienta. Vzorová stránka uložená v souboru `page.rbx` vypadá následovně

```
require 'borges/ApacheDRbClient'
Borges::ApacheDRbClient.handle_with('druby://127.0.0.1:7001')
```

51.5.1.2. WEBrick

WEBrick proberu jen v té jednodušší variantě. Na konci každého hlavního souboru v podmínce `$0==__FILE__` která je splněna spouštíme-li tento soubor jako program vložím kód jenž startuje WEBrick:

```
if $0 == __FILE__ then
    require 'borges/Webrick'
    Borges::WebrickConnection.start
end
```

Tak vlastně změním každou aplikace v samostatný program, který mohu ladit bez přítomnosti Apache jen spuštěním lokálního ww serveru WEBrick. Běží-li na počítači nějaký jiný www server, musím upravit spouštění WEBricku na jiný port. Takže při startu přidám parametr *`Port`*, například použiji `7000`

```
Borges::WebrickConnection.start({:Port=>7000})
```

51.5.2. První stránka

Nejdříve si ukážeme obligátní „hello“ program.

```
#!/usr/bin/env ruby -w
# $Id: hello.rb,v 1.1 2003/12/08 18:40:07 radek Exp $
# $Source: /home/radek/cvs/ruby-book/example/net/web/borges02/hello.rb,v $
# Just a simple hello application
# Copyright (c) 2003 Radek Hnilica
require 'borges'
require 'borges/Page'

include Borges

class Hello < Page
    def render_on(r)
        r.heading "Hello world!"
    end
end

Borges::register_application("hello", Hello)

if $0 == __FILE__ then
    require 'borges/Webrick'
    Borges::WebrickConnection.start
end;
```

Spustíme jej příkazem

```
$ ruby -w hello.rb
```

A prohlížeč nasměrujeme na `http://localhost:7000/hello`

Od verze 0.1.1 spolupracuje Broges s Apachem přes DRb. Vlastní stránka je pak v Apachi napsaná v mod_ruby

```
#!/usr/bin/env ruby
# $Id: hello.rbx,v 1.1 2003/12/08 18:40:07 radek Exp $
# $Source: /home/radek/cvs/ruby-book/example/net/web/borges02/hello.rbx,v $
require 'borges/ApacheDRbClient'
Borges::ApacheDRbClient.handle_with('druby://127.0.0.1:7001');
```

Aby stránka fungovala, musí být spuštěn server na který se odkazuje. Tento spustíme následujícím skriptem.

```
#!/usr/bin/env ruby -w
# $Id: hello_srv.rb,v 1.1 2003/12/08 18:40:07 radek Exp $
# $Source: /home/radek/cvs/ruby-book/example/net/web/borges02/hello_srv.rb,v $
require 'hello.rb'
require 'borges'
require 'borges/ApacheDRbServer'

Borges::register_application("borges/hello.rbx", Hello)
Borges::ApacheDRbServer.start('druby://127.0.0.1:7001')
Borges::ApacheDRbServer.install_INT_handler
puts "Borges DRb Server listening on #{DRb.uri}"
```

```
puts "Press Ctrl-c to quit"
DRb.thread.join;
```

* *Zatím neodzkoušeno, nemám nainstalován DRb.*

Havaruje se zprávou:

```
[Tue May 20 16:20:33 2003] [error] mod_ruby: error in ruby
/home/radek/opt/ruby-1.8.0-2003.05.20/lib/ruby/site_ruby/1.8/borges/ApacheDRbClient.rb:23:in
‘handle_with’: undefined method ‘contentType’ for #<Borges::Response:0x403dc840> (NoMethodError)
       from /var/www/borges/hello.rbx:5
       from /home/radek/opt/ruby-1.8.0-2003.05.20/lib/ruby/1.8/apache/ruby-run.rb:70:in ‘load’
       from /home/radek/opt/ruby-1.8.0-2003.05.20/lib/ruby/1.8/apache/ruby-run.rb:70:in ‘handler’
```

Na tak jednoduchém příkladu jako je `hello.rb` toho moc neuvidíme. Skusíme něco přímo z příkladů od Broges

```
#!/usr/bin/env ruby
# $Id: counter.rb,v 1.1 2003/12/08 18:40:07 radek Exp $
# $Source: /home/radek/cvs/ruby-book/example/net/web/borges02/counter.rb,v $
require ’borges’
require ’borges/Page’

include Borges

class Counter < Page
    def initialize
        @counter = 0
    end

    def render_on(r)
        r.heading @counter
        r.anchor("++") { inc }
        r.space
        r.anchor("--") { dec }
    end

    def inc(); @counter += 1; end
    def dec(); @counter -= 1; end
end

Borges::register_application("counter", Counter)

if $0 == __FILE__ then
    require ’borges/Webrick’
    Borges::WebrickConnection.start
end
;
```

Pokročíme k složitějšímu příkladu, přihlašovacímu formuláři

```
#!/usr/bin/env ruby
# $Id: login.rb,v 1.1 2003/12/08 18:40:07 radek Exp $
# $Source: /home/radek/cvs/ruby-book/example/net/web/borges02/login.rb,v $
# Copyright (c) 2003 Radek Hnilica
require ’borges’
require ’borges/Page’
```

```
   include Borges

10 class Login < Page
       def initialize
           @username = nil
           @password = nil
       end
15
       def render_on(r)
           r.heading 'Login'
           r.form do
               r.print "Jm<65533>no:"
20             r.text_input(")){|n| @username = n; puts "*> n= #{n}"}
               r.print "<br>Heslo:"
               r.text_input(")){|n| @password = n; puts "*> n= #{n}"}
               r.print "<br>"
               r.submit({'value'=>'P<65533>ihl<65533>sit'}) {login}
25         end
       end

       def login()
           puts "*> Login as #{@username} with #{@password}"
30     end
   end

   Borges::register_application("login", Login)

35 if $0 == __FILE__ then
       require 'borges/Webrick'
       Borges::WebrickConnection.start({:Port=>7000})
   end;
```

Po drobné úpravě v Borges, konkrétne po záplatě:

```
   diff -rC3 borges-0.1.0/lib/borges/Renderer.rb borges-0.1.0.1/lib/borges/Renderer.rb
   *** borges-0.1.0/lib/borges/Renderer.rb Sun Jan 26 02:28:30 2003
   --- borges-0.1.0.radek/lib/borges/Renderer.rb   Mon Feb 10 13:25:03 2003
   ***************
 5 *** 61,66 ****
   --- 61,72 ----
         attrs["name"] = @callbacks.register_callback(update)
         input("text", attrs)
       end
10 +
   +   def password_input(value, attrs={}, &update)
   +     attrs["value"] = value
   +     attrs["name"] = @callbacks.register_callback(update)
   +     input("password", attrs)
15 +   end

       def checkbox(value, attrs={}, &update)
         update_key = @callbacks.register_callback(lambda { |v|
```

který přidává metodu `password_input` a přepsání s použitím tabulky vypadá přihlašovací stránka takto:

```
#!/usr/bin/env ruby
# $Id: login2.rb,v 1.1 2003/12/08 18:40:07 radek Exp $
#<65533>$Source: /home/radek/cvs/ruby-book/example/net/web/borges02/login2.rb,v $
# Copyright (c) 2003 Radek Hnilica
```

```
require 'borges'
require 'borges/Page'

include Borges

class Login < Page
    attr_reader :username

    def initialize
        @username = nil
        @password = nil
    end

    def render_on(r)
        r.tag('html') do
            r.tag('head') do
                r.print "<title>P<65533>ihla<65533>ovac<65533> dialog varianta 2</title>\n"
                r.print %q{<meta http-equiv="Content-Type"}
                r.print %Q{content="text/html; charset=iso-8859-2">\n}
            end
            r.tag('body') do
                r.form do
                    r.table({'frame'=>'border', 'align'=>'center', 'bgcolor'=>'#e0e8ff'}) do
                        r.tr do
                            r.td({'colspan'=>'2', 'align'=>'center', 'bgcolor'=>'#d0c0e0'}){
                                r.print "<b>Pros<65533>m, p<65533>ihlaste se:</b>\n"}
                        end
                        r.tr do
                            r.td{r.print "Jm<65533>no:"}
                            r.td{r.text_input(""){|n| @username = n;}}
                        end
                        r.tr do
                            r.td{r.print "Heslo:"}
                            r.td{r.password_input(""){|n| @password = n;}}
                        end
                        r.tr do
                            r.td({'colspan'=>'2', 'align'=>'center'}){
                                r.submit({'value'=>'P<65533>ihl<65533>sit'}) {login}}
                        end
                    end
                end
            end
        end
    end

    def login()
        # Use user authorization code for login in
        # This is simple fake code.
        if @username == 'radek' and @password == 'pass' then
            puts "#{@username}> P<65533>ihl<65533>enn<65533> bylo <65533>sp<65533>n<65533>."
        else
            puts "#{@username}> ERR: Bad password given!"
            @username = @password = nil
        end
    end
end

Borges::register_application("login-two", Login)

if $0 == __FILE__ then
    require 'borges/Webrick'
    Borges::WebrickConnection.start({:Port=>7000})
end

#Keep this comment at the end of the file
#Local variables:
#indent-tabs-mode: nil
#End:;
```

51.5.3. Komponenty

Další věcí kterou se naučíme je jak vytvářet komponenty a jak je skládat. Například si vytvoříme komponentu pro přihlašování která pak může být součástí libovolné stránky. Všechno kolem přihlašování je zapouzdřeno do komponenty a tu pak jen používáme aniž bychom se museli o něco starat.

Princip komponenty je zapouzdření funkčnosti do samostatného objektu.

Příklad 51-23. Ukázka komponenty

```
class LoginComponent < Page
    def initialize
        @username = nil
    end
    def login(user, pass)
        authorized = authentication(user, pass)
        @username = user if authorized
        authorized
    end
    def authentication(user, pass)
        user == pass            ❶
    end
    def render_on(r)
        name = pass = nil       ❷
        r.form do
            r.print "Name:"
            r.text_input(”){|name|}
            r.print "Password:"
            r.password_input(”){|pass|}
            r.submit({’value’=>’Login’}){login(name, pass)}
        end
    end
end
```

❷ Zde jsou vytvořeny lokální proměnné. Tento postup je nezbytný, neboť hodnotu těchto proměnných měním ve vnořených blocích. Pokud bych zdě proměnné takto nedefinoval, v daných blocích by vznikly a současně s jejich opuštěním zanikly.

❶ Zástupný autentikační kód. V reálné aplikaci je na tomto místě skutečné ověření pravosti hesla v databázi, ldapu či jiným odpovídajícím bezpečným způsobem.

Použití komponenty ve stránce pak vypadá asi takto

Příklad 51-24. Použití dříve vytvořené komponenty

```
class Main < Page
    def initialize()
        @login = LoginComponent.new     ❶
    end
    def render_on(r)
        ...
        @login.render_on(r)     ❷
        ...
    end
end
```

❶ Vytvoření komponenty při inicializaci objektu.

❷ Použití vytvořené komponenty. Na tomto místě se zobrazí (generuje její obsah)

První pokus o komponentu je návrh přihlašovací komponenty do hlavní stránky.

```
#!/usr/bin/env ruby
# $Id: login3.rb,v 1.1 2003/12/08 18:40:07 radek Exp $
# $Source: /home/radek/cvs/ruby-book/example/net/web/borges02/login3.rb,v $
# Component example
# Copyright (c) 2003 Radek Hnilica
require 'borges'
require 'borges/Page'

include Borges

# Component class.  In real word it will be probably in separated file.
class LoginComponent < Page
    def initialize
        @username = nil
    end
    def login(username, password)
        # Use user authorization code for login in
        # This is simple fake code.
        if username == 'radek' and password == 'pass' then
            puts "#{username}> P<65533>ihl<65533>enn<65533> bylo <65533>sp<65533>n<65533>."
            @username = username
        else
            puts "#{username}> ERR: Bad password given!"
            @username = nil
        end
    end
    def render_on(r)
        username = password = nil       # Declaring local variables.
        r.form do
            r.table({'frame'=>'border', 'align'=>'center', 'bgcolor'=>'#e0e8ff'}) do
                r.tr do
                    r.td({'colspan'=>'2', 'align'=>'center', 'bgcolor'=>'#d0c0e0'}){
                        r.print "<b>Pros<65533>m, p<65533>ihlaste se:</b>"}
                end
                r.tr do
                    r.td{r.print "Jm<65533>no:"}
                    r.td{r.text_input(""){|username|}}
                end
                r.tr do
                    r.td{r.print "Heslo:"}
                    r.td{r.password_input(""){|n| password = n;}}
                end
                r.tr do
                    r.td({'colspan'=>'2', 'align'=>'center'}){
                        r.submit({'value'=>'P<65533>ihl<65533>sit'}){
                            login(username, password)}}
                end
            end
        end
    end
end

class Login < Page
    attr_reader :username

    def initialize()
        @login = LoginComponent.new
```

```
            @title = "Hlavn<65533> str<65533>nka s p<65533>ihla<65533>ovac<65533> komponentou"
        end

        def render_on(r)
            r.tag('html') do
                # html/head just to be correct especially with charset.
                r.tag('head') do
                    r.print "<title>P<65533>ihla<65533>ovac<65533> dialog varianta 2</title>"
                    r.print %q{<meta http-equiv="Content-Type" }
                    r.print %Q{content="text/html; charset=iso-8859-2">\n}
                end
                r.tag('body') do
                    # Page main table construction
                    r.table({'rules'=>'all', 'frame'=>'border'}) do
                        r.tr do
                            r.td{r.print "LOGO"}
                            r.td{r.heading @title}
                            r.td do
                                # use the login component
                                @login.render_on(r)
                            end
                        end
                        r.tr do
                            r.td({'rowspan'=>'2'}){r.print "Left<br>Side<br>Menu"}
                            r.td{r.print "<a href="">Restart</a>   <a>Nothing</a>"}
                            r.td({'rowspan'=>'2'}){r.print "Right<br>Side<br>Column"}
                        end
                        r.tr do
                            #Left Side Menu spans here
                            r.td{r.print "CONTENTS"}
                            #Right Side Colume spans here
                        end
                        r.tr do
                            r.td({'colspan'=>'3'}){r.print "BOTTOM"}
                        end
                    end
                end
            end
        end
    end

    Borges::register_application("login-three", Login)

    if $0 == __FILE__ then
        require 'borges/Webrick'
        Borges::WebrickConnection.start({:Port=>7000})
    end
    #Keep this comment at the end of the file
    #Local variables:
    #indent-tabs-mode: nil
    #End: ;
```

51.5.4. Přehled objektů a metod

Přehled objektů a a metod Borges obohacený o poznámky a komentáře.

modul `Borges`

`register_application`(*`name`*, *`entry_point`*=`nil`, *`&block`*)

Zaregistrování nové aplikace. Aplikace se registruje pod jménem *`name`* přes které se k ní přistupuje. Příklad

```
class Hello < Page
    def render_on(r)
        r.heading "Hello world!"
    end
end
Borges::register_application("hello", Hello)
```

`root_application=`(*`name`*)

Accessing `'//'` will redirect to application *`name`*

`respond`

Grab the Session for this thread and ...

`render`(*`&block`*)

FIXME:

třída `Borges::Application`

`Apps` — *hash*

Hash/Asociativní pole/Slovník.

`handle_request(request, response)`

Create a new session for this request if the session_key does not exist. Otherwise pass the request to the session and set the response's session_key to the request's session_key.

`new_session(response)`

Create a new session, and set the response's session_key.

modul `Borges::ApacheDRbClient`

`self.handle_with(server_uri)`

Pass this request off to the DRb Server.

třída `Borges::ApacheDRbServer`

`self.start(drb_uri)`

Start up the DRb server.

`self.install_INT_handler`

Install a SIGINT handler, so DRb will shut down cleanly on ctrl-c

`handle_request(request)`

Pass the request off to Borges for processing, returns the response. Přesměruje/předá *`request`* do Borges voláním

```
Borges::Application.handle_request(request, response)
```

Objekt *`response`* s odpovědí pak vrátí jako návratovou hodnotu.

Třída `Borges::Page`

`go`

FIXME: doplnit

`render_on(r, embeded=false)`

Metoda vykresluje stránku (celou) do *`r`*. Druhý parametr označuje zdali vykreslujeme celou stránku `false` nebo jen její část `true`. Bez uvedení tohoto parametru se předpokládá vykreslení celé stránky.

`on_head(r)`

FIXME: doplnit

`on_body(r)`

Metoda vykresluje tělo stránky na *`r`*. Tato metoda musí být v podtřídě předefinována, jinak vyvolá chybu.

`answer(return_val=`*`nil`*`)`

FIXME: doplnit

Třída `Borges::Session`

`initialize(app)`

Create a new session for app.

`handle_request(request, response)`

Set up the Session's thread for execution, and dispatch.

`respond`

Take the continuation and save it in the cache. ...

`expire`

Kill this thread.

`handle_request_intern(request, response)` — *private method*

Handle a request, dispatching to the request's action_key, if such a key exists in the hash, otherwise start the application from the beginning.

Called by the public handle request.

`unknown_request(request)` — *private method*

Start this request over.

`handle_error(err)` — *private method*

Spit out a backtrace for the Exception err.

Třída `Borges::Renderer`

`render_response`

Render the page and dispatch the callback for the clicked action.

`submit(attrs={}, &action)`

Render tag submit.

Třída `Borges::CallbackStore`

`process_callbacks(request)`

Run the action the user clicked.

51.5.5. Spouštění aplikace na serveru

FIXME: vyřešit problém startování druby serveru a popsat řešení v četně skriptů

Tedy první část úkolu je upravit server aby zapisoval do souboru své číslo procesu. Toto lze ve zkratce zajistit příkazy

```
require 'English'
pid_file = "/var/run/webapp/app.pid"    ❶
...
File.open(pid_file, 'w') do |file|
    file << $PROCESS_ID                 ❷
end
...
File.delete(pid_file)                   ❸
```

❶ Nastavíme si cestu k souboru do nějž budeme ukládat PID.

❷ Tady zapíšeme číslo procesu.

❸ Na konci po sobě uklidíme. Dojde-li k vážné havárii programu tak se tento úklid pravděpodobně nevykoná, na to je třeba mít na paměti.

Jádro spouštěcího skriptu

```
#!/bin/sh
APPDIR=opt/webapp/ldapadm       ❶
SERVER=./ldapadm_srv.rb
PIDFILE=/ver/run/webapp/ldapadm.pid

function start_server {         ❷
    (
        cd $APPDIR
        $SERVER --pidfile $PIDFILE &
    )
}

if [ -r $PIDFILE ]; then        ❸
```

```
    if ! ps -p $(cat $PIDFILE) >/dev/null; then ❹
        start_server            ❺
    fi
else
    start_server                ❻
fi
```

❶ Všechny důležité parametry definuji na začátku jako konstanty. Usnadní to modifikaci skriptu.

❷ Protože server startuju na dvou místech v kódu, udělal jsem si pro jeho start funkci.

❸ Testuji, existuje-li soubor s číslem procesu.

❹ Ověříme si je-li proces s číslem uvedeným v `ldapadm.pid` mezi bežícími proces.

❺ Nastartujeme server.

❻ Nastartujeme server.

51.5.6. Nezpracováno

```
#!/usr/bin/env ruby -w
# $Id: component.rb,v 1.1 2003/12/08 18:40:07 radek Exp $
# $Source: /home/radek/cvs/ruby-book/example/net/web/borges02/component.rb,v $
#
# Copyright (c) 2003 Radek Hnilica

require 'borges'
require 'borges/Page'

include Borges

class MyComponent < Page
    def render_on(r)
        r.heading 'MyComponent'
 r.anchor('Component Action') { action }
    end
    def action
    end
end

class MainPage < Page
    def render_on(r)
        r.heading 'Component example'
 # use MyComponent
 r.print "End Of Page"
    end
end

Borges::register_application("component", MainPage)
if $0 == __FILE__ then
    require 'borges/Webrick'
    Borges::WebrickConnection.start({:Port=>7001})
end
```

Pokus o velmi jednoduchou aplikaci

```
#!/usr/bin/env ruby -w
# $Id: app.rb,v 1.1 2003/12/08 18:40:07 radek Exp $
# $Source: /home/radek/cvs/ruby-book/example/net/web/borges02/app.rb,v $
# Copyright (c) 2003 Radek Hnilica
require 'borges'
require 'borges/Page'
include Borges

class MainPage < Page
    def render_on(r)
 r.heading 'App'
    end
end
Borges::register_application("app", MainPage)
if $0 == __FILE__ then
    require 'borges/Webrick'
    Borges::WebrickConnection.start({:Port=>7002})
end
```

51.5.6.1. Dopis

Webrick provides servlet stubs and everything else, so it would be practical, probably trivial, to port IOWA to it.

Here's what I use for Borges, all the work happens in do_GET/do_POST, and the majority of that work is simply mapping WEBrick Request/Response to a Borges request/response. (With a little work, I may be able to use a webrick request/response inside Borges, I haven't looked into it yet.)

```
require 'webrick'

class BorgesServer < WEBrick::HTTPServlet::AbstractServlet

  attr_accessor :handler

  def initialize(server, options = {})
    super @handler = options[:Handler] || WADispatcher.default
  end

  ##
  # WEBrick HTTP GET handler

  def do_GET(req, res)
    request = WARequest.new(req.path, req.header, req.query, req.cookies)
    response = @handler.handle_request(request)

    res.status = response.status
    res.body = response.contents

    response.headers.each do |k,v|
      res[k] = v
    end
  end
```

```
    ##
    # WEBrick HTTP POST handler (same as GET)

    alias do_POST do_GET

    ##
    # Create a new Borges Server
    def self.create(options)
      options[:BindAddress] ||= '0.0.0.0'
      options[:Listen] ||= [['::', 7000]]
      options[:Port] ||= 7000

      server = WEBrick::HTTPServer.new(options)
      server.mount("/borges", BorgesServer, options)

      return server
    end

    ##
    # Start a new BorgesServer with a SIGINT handler

    def self.start(options = {})
      server = self.create(options)
      trap("INT") do server.shutdown end
      server.start

      return server
    end

  end
```

Eric Hodel - <drbrain@segment7.net> - http://segment7.net

51.6. IOWA - *Interpreted Objects for Web Applications*

* *section id="iowa" xreflabel="IOWA"*
* *Přesunout do části Programování webových aplikací.*

Zdroje:

- IOWA (http://sourceforge.net/projects/iowa/) na SourceForge (http://sourceforge.net/)
- Framsida brukergrensesnitt prototyp (http://v046b.studby.ntnu.no/eit/framsida/)
- IOWA (http://docs.enigo.com/iowa/)

Vývojáři IOWA na SourceForge.NET

- Avi Bryant (http://sourceforge.net/users/avibryant/)
- Eli Green (http://sourceforge.net/users/eligreen/)
- Julian Fitzell (http://sourceforge.net/users/jfitzell/)
- Kirk Haines
- Dave Thomas

Balíček IOWA uvádím jen z historických důvodů. Jedná se o předchůdce Borges a již se dále nevyvíjí. Poslední známá verze je `0.15`. Podle zpráv z října 2003 pracoval na IOWĚ poslední rok a půl Kirk Haines. Slíbil v brzké době publikovat výsledky své práce.

51.6.1. Co je Iowa?

Poznámka: Volně přeloženo z dokumentace.

Iowa je množina/několik tříd uzpůsobených k nesmírnému/obrovskému ulehčení vývoje aplikací s rozhraním html.

Iowa je

Interpretovaná

Všechen kód Iowy je napsán v Ruby, extrémně dynamickém a elegantním programovacím jazyce. Na rozdíl od řešení v Javě a dalších kompilovaných jazycích, Iowa zapadne snadno do cyklu změň-ulož-podívej se (*change-save-reload*). Jako programátoři si snadno zvyknete na to, že můžete měnit text programu za běhu a sledovat změny.

Objektová

Iowa přináší na web model „všechno je objekt“. To není fráze, aplikace jsou sestaveny ze znovupoužitelných zapouzdřených webových komponent a Iowa sama je vysoce modulární systém který se snadno přizpůsobuje a udržuje.

pro webové aplikace

s důrazem na „aplikace“. Iowa vám umožňuje přemýšlet bez starostí o mechaniku generování html, skrývání polí, dotazové řetězce a další nízkoúrovňové fígle se kterými se vývojáři denně setkávají. Je uzpůsobena způsobu myšlení vývojaře.

51.6.2. Kde ji získat?

IOWA je k dispozici na SourceForge (http://sourceforge.net/) a domovskou stránku najdete na http://www.beta4.com/iowa/. K dispozici je i veřejně přístupné cvs.

```
$ export CVSROOT=:pserver:anonymous@cvs.iowa.sourceforge.net:/cvsroot/iowa
$ cvs login
Logging in to :pserver:anonymous@cvs.iowa.sourceforge.net:2401/cvsroot/iowa
CVS password:
$ cvs -z3 co iowa
```

51.6.3. Kompilace a Instalace pod UNIXem

Před kompilací je třeba opravit soubor `makefile`. Ten na začátku deklaruje:

```
APXS=/usr/sbin/apxs
```

který musí být opraven na

```
APXS=/usr/bin/apxs
```

poté můžeme začít překládat:

```
$ make
$ su root make install
```

```
Password:
/usr/bin/apxs -i -a -n 'iowa' mod_iowa.so
[activating module 'iowa' in /etc/apache/httpd.conf]
cp mod_iowa.so /usr/lib/apache/1.3/mod_iowa.so
chmod 755 /usr/lib/apache/1.3/mod_iowa.so
cp /etc/apache/httpd.conf /etc/apache/httpd.conf.bak
cp /etc/apache/httpd.conf.new /etc/apache/httpd.conf
rm /etc/apache/httpd.conf.new
$ ruby instal.rb
```

Dále je třeba nainstalovat `iowa.cgi`

```
$ cp iowa.cgi /usr/lib/cgi-bin/
$ chmod a+x /usr/lib/cgi-bin/iowa.cgi
```

Do souboru `/etc/apache/httpd.conf` jsem přidal

```
LoadModule action_module      /usr/lib/apache/1.3/mod_actions.so
LoadModule iowa_module        /usr/lib/apache/1.3/mod_iowa.so
Action iowa /cgi-bin/iowa.cgi
<Location /iowa>
        SetHandler iowa
        Order   allow,deny
        Allow   from all
</Loction>
```

a do adresáře `/usr/lib/cgi-bin` jsem zkopíroval soubor `iowa.cgi`

* *Vypadá to že CVS verze IOWA a Ruby updatovana 2002-12-17 konečně funguje. Dobrá zpráva.*

51.6.4. Naše první stránky

Po úspěšném sprovoznění balíčku/knihovny IOWA můžeme přistoupit k vytvoření naší první stránky. A jak jinak začneme známým programem `Hello World`. Vytvoříme si tedy adresář pro náš pokus, například `hello` a přepneme se do něj

```
$ mkdir hello
$ cd hello
```

Nyní vytvoříme soubor `Main.html` s následujícím obsahem

Příklad 51-25. IOWA `hello/Main.html`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<!-- $Id: Main.html,v 1.1 2002/11/03 06:55:44 radek Exp $
     $Source: /home/radek/cvs/ruby-book/example/net/iowa/hello/Main.html,v $ -->
<html>
  <body>
    <h1>Hello World!</h1>
  </body>
</html>
```

Náš první „program“ spustíme následujícím příkazem zadaným v adresáři `hello`

```
$ ruby -riowa -e "Iowa.run('hello')"
```

IOWA načte soubor v adresáři a prezentuje jej na *webowském* rozhraní počítače. Vytvořená stránka je přístupná na url (`http://localhost/iowa/hello`) http://localhost/iowa/hello. V případě mého počítače je to http://kvark:8080/iowa/hello

* *FIXME: Podívat se jak vypadají předcházející url na různých vástupech.*

```
$ lynx localhost:8080/iowa/hello
```

Výsledný html kód která získáme ze serveru vypadá takto

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<!-- $Id$
     $Source$ -->
<html>
  <body>
    <h1>Hello World!</h1>
  </body>
</html>
```

Získali jsme tedy velmi „drahý“ www server ;)

Na prvním příkladě jsme se naučili, jak spustit IOWA a kde najdeme jeho výstup. Jednoduchostí příkladu jseme jej však degradovali na obyčejný web server publikující statické stránky. Ukažme si tedy něco z *dynamičnosti* IOWA. Na začátek souboru přidáme část `<% ... %>` a rovněž upravíme tělo dokumentu. Výsledek vypadá takto

Příklad 51-26. IOWA `hello2/Main.html`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<!-- $Id: Main.html,v 1.1 2002/11/03 06:55:45 radek Exp $
     $Source: /home/radek/cvs/ruby-book/example/net/iowa/hello2/Main.html,v $ -->
<%
    class Main < Iowa::Component
        def time
            Time.now
        end
    end
%>
<html>
  <body>
    <h1>Hello World!</h1>
    <p>Aktu<65533>ln<65533> <65533>as je <b>@time</b>.</p>
  </body>
</html>
```

Po uložení souboru dáme v prohlížeči *reload* a hned vidíme změnu. Naše stránka po každém načtení ukazuje aktuální čas v době načtení. Výsledný html vypadá takto:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<!-- $Id$
     $Source$ -->

<html>
  <body>
    <h1>Hello World!</h1>
    <p>Aktu<65533>ln<65533> <65533>as je <b>Fri Nov 01 21:47:38 CET 2002</b>.</p>
  </body>
</html>
```

Nic světoborného, následující příklad, nazvěme si jej `timelog` je již dynamičtější. Opět si vytvoříme adresář `timelog` pro náš „program“ a v něm jeden soubor, `Main.html`

Příklad 51-27. IOWA `timelog/Main.html`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<!-- $Id: Main.html,v 1.1 2002/11/03 06:55:45 radek Exp $
     $Source: /home/radek/cvs/ruby-book/example/net/iowa/timelog/Main.html,v $ -->
<%
    class Main < Iowa::Component
        attr_reader :loggedTimes
        def awake; @loggedTimes = []; end
        def log; @loggedTimes << Time.now; end
        def time; Time.now; end
    end
%>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
  </head>
  <body>
    <h1><65533>asov<65533> zna<65533>ky</h1>
    <p>Aktu<65533>ln<65533> <65533>as je <b>@time</b>.</p>
    <a oid="log">P<65533>idej</a> zna<65533>ku.
    <p>Na tuto str<65533>nku jsi se pod<65533>val ty a nebo n<65533>kdo jin<65533>
       v <65533>asech: @loggedTimes.</p>
  </body>
</html>
```

Jak jste si zajisté všimli, po každém kliknutí na `Přidej` se seznam časových značek prodlouží o aktuální čas kliknutí. Seznam se nezobrazuje příliš pěkně, tak uděláme pár úprav. Předně napíšeme část vazeb (*bindings*) `<? ... ?>` a do html kódu přidáme zobrazení seznamu. Rovněž přidáme řádek do části programu. Výsledný soubor vypadá takto:

Příklad 51-28. IOWA `timelog2/Main.html`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<!-- $Id: Main.html,v 1.1 2002/11/03 06:55:45 radek Exp $
     $Source: /home/radek/cvs/ruby-book/example/net/iowa/timelog2/Main.html,v $ -->
<%
    class Main < Iowa::Component
        attr_reader :loggedTimes
        attr_accessor :logItem
        def awake; @loggedTimes = []; end
        def log; @loggedTimes << Time.now; end
        def time; Time.now; end
    end
%>

<?
    logList {
        list = loggedTimes
        item = logItem
    }
?>
```

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
  </head>
  <body>
    <h1><65533>asov<65533> zna<65533>ky</h1>
    <p>Aktu<65533>ln<65533> <65533>as je <b>@time</b>.</p>
    <a oid="log">P<65533>idej</a> zna<65533>ku.
    <p>Na tuto str<65533>nku jsi se pod<65533>val ty a nebo n<65533>kdo jin<65533>
       v <65533>asech:
      <ul oid="logList">
        <li>@logItem</li>
      </ul>
    </p>
  </body>
</html>
```

51.6.4.1. Spouštěcí soubor

Abychom mohli snadno spouštět náš aplikační server, vytvoříme si proto spouštěcí soubor. Nejjednodušší spouštěcí soubor vypadá následovně:

Příklad 51-29. `iapp_start.rb`

```
require 'iowa'
Iowa.run('název_aplikace')
```

Webová stránka naší aplikace je pak `http://server/iowa/název_aplikace`. Samozřejmě takto to vpadá jen ve standardní konfiguraci. Pokud si s konfiguračními soubory pohrajeme, může to vypadat velmi odlišně.

Takto vytvořený spouštěcí soubor spouštíme přímo v ruby

```
$ ruby iapp_start.rb
```

Program se spustí a na terminál vypisuje zprávy a chybové hlášení. Pokud chceme spustit applikaci na pozadí, přidáme dva řádky:

```
  require 'iowa'
+ exit if fork
+ Process.setsid
  Iowa.run('název_aplikace')
```

51.6.5. Metody objektu `IOWA::Component`

Zdroje a odkazy:

- http://beta4.com/iowa/ref.html

51.6.5.1. `pageNamed(aName)` ⟶ aComponent

Vrací novou instanci stránkového komponentu s jménem *aName*

51.6.5.2. `awake`

Tato metoda je volána po ukončení inicializace komponenty.

51.6.5.3. `session` ⟶ aSession

Vrací objekt aktuálního sezení (*session*).

51.6.5.4. `invokeAction(aMethod, aBlock)`

Tato metoda je volána při kliknutí na odkazu na stránce. Standardně volá `send`.

51.6.5.5. `dup`

Používá se k získání stránek s vyrovnávací paměti (*cache*) při zpracování dotazu. *Override to keep snapshoots of volatile state to facilitate backtracking.*

51.6.5.6. `handleBacktracking`

* ***FIXME:****přeformulovat*

Je volána, pokud přišel dotaz z jiné než právě zobrazené stránky. Standardně nedělá nic. Mužete vyvolat výjimku PageExpired k převední uživatele na nejnovější bod v seanci, nebo ignorovat výjimkou IgnoreRequest, a nevo znovu zobrazit stránku.

51.6.5.7. `back`

Standardní akce zavolá komponentu která tuto stránku vytvořila (předcházející stránku).

51.6.5.8. `reload`

Standardní akce volá tuto komponentu.

51.6.6. Přehled tagů

51.6.6.1. `form`

FIXME:

```
<form action=...>
    <input value=...>
    <textarea value=...>
    <input type="submit" value=... action=...>

</form>
```

51.6.6.2. repeat, ul, table, tr

<repeat>

51.6.6.3. select

```
<select list=... item=...>
```

51.6.6.4. span, if

```
<select list=... item=...>
```

51.6.6.5. string

```
<select value=...>
```

Poznámka: `@name` je zkratka za html tag `<string oid="name"/>`

51.6.7. Nezapracované poznámky

```
$ ruby -riowa -e "Iowa.run('tutorial')"
```

51.6.7.1. Aktivní vývoj

IOWA má jednu zvláštnost. Běžící server rozpozná změny ve zdrojovém kódu stránek a provede aktualizaci z tohoto zdrojového kódu. To nám umožňuje pracovat na stránkách aniž bychom museli restartovat server.

Tato vlastnost je implementována pomocí metody `reloadModified` třídy `Application`

51.6.7.2. Dopis Kirka Hainese z 2003-10-10

I vanished off of the list about a year ago as a result of not having time to keep up with the traffic and keep the family fed. I'm back actively reading, though, and have been skimming through some archived posts for interesting things. I came across this, and thought I'd send out a quick update.

IOWA is not really a dead project. I picked it up for a production application about a year and a half ago, and found it, while rough, a package with a lot of potential. In the last year and a half I have commented it, documented it, refined it, and expanded it quite a bit while at the same time using it on pieces ranging from single dynamic report pages to custom software apps to an entire dynamic site engine for sites with significant dynamic content. I've got about a dozen discrete sites using it at some level.

I've added the ability to receive a faux Apache::Request object into the Iowa app that gives access to most everything Apache::Request does, including the ability to alter the HTTP headers that go out to the client, which lets one use cookies, have access to Query_String parameters, and other useful things. I've created a system to use IOWA to map specific URLs to IOWA, which makes it useful and practical to use IOWA for websites with

dynamic content, but which are not necessarily full fledged web applications themselves. i.e. pages that deliver reports, or pages that need to be bookmarkable, or even sites that map discrete content elements to the same URLs, delivering one or another based on a login cookie (I have a large, complete production site, running hundreds of thousands of hits per day, that does this) or language preference or other criteria.

So, IOWA isn't really dead. It's very much alive. It's just that I've also been keeping it pretty quiet as I used it and worked on it. Unless Avi has any objections (and I haven't asked, yet), I'm working on setting up a new website for IOWA that contains better documentation, live examples, and a current, updated installation package.

Kirk Haines

51.7. CGIKit

* *section id="cgikit" xreflabel="CGIKit" condition="author"*
* *Přesunout tuto sekci do části Programování webových aplikací, případně z ní udělat samostatnou kapitolu v téže části.*

Odkazy a zdroje:

- CGIKit (http://www.spice-of-life.net/download/cgikit/index_en.html)

ToDo

1. První úkol.

CGIKit je aplikační prostředí pro tvorbu webu napsané v Ruby.

51.8. Nora

* *section id="nora" xreflabel="Nora"*

FIXME: Webový aplikační server Nora rwiki

51.9. Ostatní WWW servery

* *section id="ostatni-www-servery" xreflabel="Ostatní WWW servery" condition="author"*

Zdroje a odkazy:

- http://www.freedom.ne.jp/toki/ruby.html#ruby:script:wwwsrv

V této části se krátce zmiňuji o existenci ostatních www a aplikačních webových serverech, jakožto i podpůrných knihovnách pro práci s webem a jeho vytváření.

V Ruby je napsáno, nebo integrováno několik vebserverů

51.9.1. wwwsrv

* *section id="wwwsrv" xreflabel="wwwsrv"*

HTTP server napsaný v Ruby.

Vlastnosti:

- Usable CGI.
- Usable SSI.

- Usable FastCGI http.fastcgi.com/.
- Usable basic authentication.
- Acceptable HTTP/1.1.
- Extensible functions by the modular structure od the contents of HTTP server.

51.9.2. Cerise

* *section id="cerise" xreflabel="Cerise"*

Zdroje a odkazy:

- Cerise (http://cerise.rubyforge.org/)
- J2EE (http://java.sun.com/j2ee)

Cerise (http://cerise.rubyforge.org/) je aplikační/web server v Ruby jenž následuje vzoru J2EE (http://java.sun.com/j2ee) aplikačních serverů.

Kapitola 52. Ostatní nástroje a prostředí pro webové aplikace

Některá prostředí a nástroje jsou stará a už se nevyvýjejí. Případně jsou zmíněny nástroje které jsem dukladněji nezkoumal. Uvádím je zde jen pro doplnění přehledu nástrojů alespoň informativně a s odkazy na zdroje.

52.1. Wee

* *section id="wee" xreflabel="Wee" condition="author" status="draft"*

Odkazy:

- Wee (http://rubyforge.org/projects/wee/) na RubyForge (http://rubyforge.org/)

Wee se řadí k www prostředím, ideově vycházejícím z Seaside (http://www.seaside.st/).

Na RubyForge jsem nalezl poslední verzi Wee 0.10.0 datovanou 2005-July-25.

52.1.1. Kritické a historické informace

Nikoliv jen v uvedeném pořadí.

Podle

Nicméně v komentářích (http://www.cincomsmalltalk.com/userblogs/avi/blogView?showComments=true&title=Wee&entry=327 na blogu HREF Considered Harmful (http://www.cincomsmalltalk.com/userblogs/avi/View.ssp) Mike Nemuan dne 2004-10-29 uvádí:

> I got now the continuations implemented, and it seem that there’s no memory leak anymore (at least, the stress test over the last 5 hours with 25 sessions and around 3 million hits does not show an anormality... it stays below the 20 MB mark).

Kapitola 53. Generování statických stránek

Ačkoliv se nejedná o dynamické weby, nechal jsem kapitolu o geneorvání statických stránek v této části knihy. Zmíním programy na které jsem narazil a podrobněji se rozepíši o několika které jsem zkusil.

Další programy k prozkoumání:

- Webby (http://webby.rubyforge.org/)
-

53.1. nanoc

Odkazy:

- nanoc (http://nanoc.stoneship.org/)

FIXME:

53.2. WebGen

Odkazy:

- webgen (http://webgen.rubyforge.org/installation.html)

FIXME:

Kapitola 54. Nasazení aplikace (deployment)

54.1. Heroku

*

Pokud změníme databázi, použijeme samozřejmě migrace. Poté po nahrání kódu aplikace na Heroku musíme také spusti migrace na Heroku.

```
$ heroku rake db:migrate
```

VI. Teorie a technologie programování

Dořešit

- Fixtures are bad (Jim Weirich)

Kapitola 55. What The Ruby Craftsman Can Learn From The Smalltalk Master

* *Z přednášky "What The Ruby Craftsman Can Learn From The Smalltalk Master"*

  Odkazy:

  - Kent Beck: SMALLTALK BEST PRACTICE PATTERNS
  - Kent Beck: Implementation Patterns
  - PH7 (http://ph7spot.com)

55.1. Naming is Crucial

*

  Naming is Crucial

  Jména přetrvávají dlouhou dobu.

  Jméno ovlivňuje způsob kterým o problému přemýšlíme.

```
class Book
    attr_reader :???

    def initialize(???)
        @??? = ???
    end

    def search(???, ???)
        #...
        ??? =
        #...
    end
end
```

  Name variable after purpose!

```
class User < ActiveRecord::Base
end

class Post < ActiveRecord::Base
    belongs_to :user
end

class User < ActiveRecord::Base
end

class Post < ActiveRecord::Base
    belongs_to :author,
               :class_name => "User"
end
```

55.2. Cognitive Scalability

*

```
class Fixnum
    def to_f
        # ...
    end
end
```

vede k

```
class String
    def to_s
    def to_str
    def to_sym
    def to_i
    def to_d
    def to_f
    def to_a
    def to_enum
    def to_set
    def to_param
    def to_query
    def to_json
    def to_xs
    def to_yaml
    def to_yaml_proprties
    def to_yaml_style
    def to_blob
    def to_date
    def to_time
    # ...
end

class File
    def initialize(file_path)
        # ...
    end
end

File.new "/tmp"

class Date
    def self.from_string(a_string)
        # ...
        Date.new # with the righ args
    end
end
Date.from_string "2009-03-14"
Date.from_julian_string "-4712-03-01"
```

Converter Constructor Method

55.3. Reduce Code to the Essence

*

```
class Point
    attr_reader :x, :y
    def initialize(x,y)
        @x, @y = x, y
    end
end

Point.new 24, -33
# ...
Point.new -75, 64
# ...
Point.new 40, -63
# ...
Point.new 47, -78
# ...
Point.new -91, 65
```

```
class Numeric
    def at(y)
        Point.new self, y
    end
end

24.at(-33)
# ...
-75.at(64)
# ...
40.at(-63)
# ...
47.at(-78)
# ...
-91.at(65)
```

55.4. Shortcut Constructor Method

*

Symmetry

```
def publish_test_report(test_results)
    create_report_directory
    @formatter.generate_report(test_results)
    upload_report
end

class SymmetricalPublisher
    def publish_rest_report(test_results)
        create_report_directory
        generate_report(test_results)
        upload_report
    end
```

```
    def generate_report(test_results)
        @formatter.generate_report(test_results)
    end
```

V Ruby můžeme zajít ještě mnohem dále.

```
class ConcisePublisher
    extend Forwardable
    def_delegators :@formatter, :generate_report

    def publish_test_report(test_results)
        create_report_directory
        generate_report(test_results)
        upload_report
    end
```

Reversing Method

Příklad špatného kódu

```
class SillyPublisher
    extend Forwardable
    def_delegators :@formatter,
                       :create_report_directory,
                       :generate_report,
                       :upload_report

    def publish_test_report(test_results)
        create_report_directory
        generate_report(test_results)
        upload_report
    end
```

Kapitola 56. Principy návrhu (*Design Principes*)

* *Z přednášky Jima Weiricha na MountainWest RubyConf 2009.*

Některé principy

- SOLID — Simple ,... Interface Segregation,
- Law of Demeter
- DRY — Do not Repat Yourself
- Small Methods
- Design by Contract
- Do not use magic numbers.
-

56.1. DRY

* ***Attributy:*** *id="DRY"*

Princip *Don't Repeat Yourself*, česky „neopakuj se“, je tak důležitý, že jej zmiňuji na prvním místě. Prolíná se většinou ostatních pravidel, nebo zněj tato pravidla přímo vycházejí. Nosná myšlenka tohoto principu je, že žádá věc, ať již kód, hodnota, znalost, ..., se nemá v programu opakovat dvakrát. Každá myšlenka je specifikována jen jednou na jednom místě.

Výhody tohoto principu jsou na snadě. Pokud modifukujeme program, provádíme úpravu jen na jednom místě, a nemusíme vzpomínat, kde ještě musíme v kódu provést změnu.

Velmi jednoduchým a snadno pochopitelným příkladem jsou konstanty. Mějme následující kód:

```
avatar = Movie.new('Avatar', 2)
 ironsky = Movie.new('Iron Sky', 1)
 alexander = Movie.new('Alexander the Great', 0)
```

Někde dále v programu s pak podle zadané číselné hodnoty rozhodujeme:

```
 @price = case price_code
           when 0: RegularPrice.new
           when 1: NewReleasePrice.new
           when 2: ChildrensPrice.new
          end
```

Takových rozhodování a použití informace filmu může být v programu více. Nevýhodou takového kódu je prezentace znalosti které číslo jaký typ filmu znamená je roztroušena po celém programu. Pokud „schováme“ znalost o typu filmu do konstant, vyhovíme principu DRY.

```
REGULAR = 0
NEW_RELEASE = 1
CHILDRENS = 2
⋮
  avatar = Movie.new('Avatar', CHILDRENS)
  ironsky = Movie.new('Iron Sky', NEW_RELEASE)
```

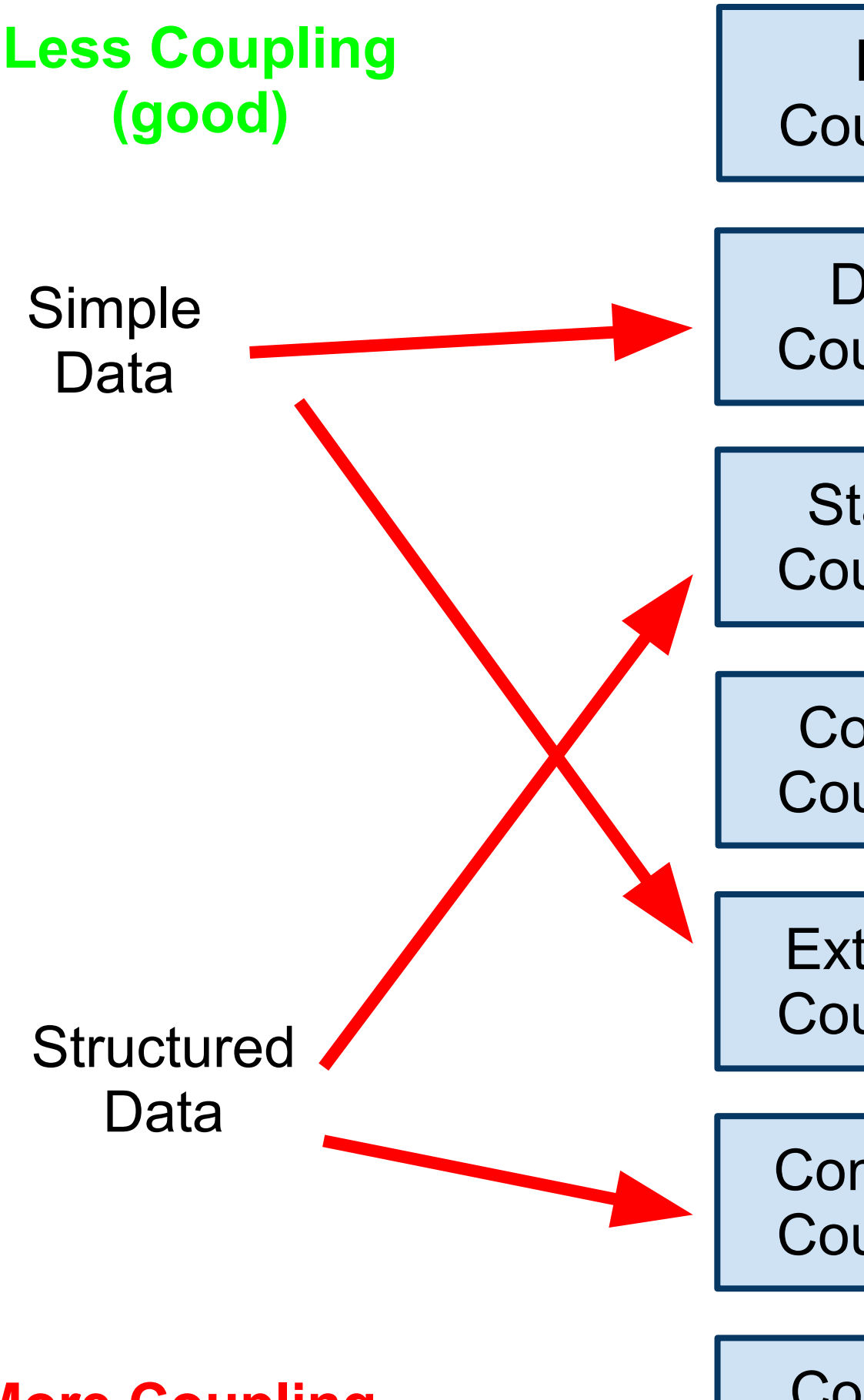
```
  alexander = Movie.new('Alexander the Great', REGULAR)
⋮
  @price = case price_code
             when REGULAR: RegularPrice.new
             when NEW_RELEASE: NewReleasePrice.new
             when CHILDRENS: ChildrensPrice.new
           end
```

56.2. Coupling & Cohesion

*

Z knihy: Composite/Structured Design, od Glenford J.Myers, 1978, z kapitoly Coupling & Cohesion

Obrázek 56-1. Coupling & Cohesion



* *scale="200"*

Control Coupling

```
Array.instance_methods
Array.instance_methods(true)
Array.instance_methods(false)
```

Jiný příklad Control Coupling

```
Customer.find(:first, . . . )
Customer.find(:all, . . . )
```

56.3. Connascence

Myerova metrika byla ve svých letech dobrá, ale nedostatěčně postihuje Objekty a dynamické jazyky.

Kniha: What every programmer should know about Object-Oriented design, Meilir Page-Jones, 1996. Z knihi má největší hodnotu třetí část.

Connascence

1. The common birth of two or more at the same time; production of two or more together.
2. That which is born or producet with another.
3. The act of growing together.

Poznámka: Nepodařil so mi zatím najít překlad slova *connascence* ani z jeho popisu usoudit jak jen popsat česky. Zatím alespoň jak tohle slovo popisuje **dict**.

```
$ dict connascence
1 definition found

From The Collaborative International Dictionary of English v.0.48 [gcide]:

 Connascence \Con*nas"cence\, Connascency \Con*nas"cen*cy\, n.
    [L. con- + nascentia birth, fr. nascens, p. pr. of nasci to
    be born.]
    1. The common birth of two or more at the same tome;
       production of two or more together. --Johnson.
       [1913 Webster]

    2. That which is born or produced with another.
       [1913 Webster]

    3. The act of growing together. [Obs.] --Wiseman.
       [1913 Webster]
```

56.3.1. Connascence of Name

```
class Customer
  def email
    ...
  end
end

def send_mail(customer)
  customer.email
end
```

Nebo.

```
create_table "customers" do |t|
  t.column :email, :string
  ...
end
```

56.3.2. Rule of Locality

- Stronger Connascence
- Weaker Connascence

56.3.3. Connascence of Position

```
:orders => {
  "3" => "1",
  "5" => "2",
}
```

Translate params hash to a List of Pairs.

```
[
  [Order.find(3), true],
  [Order.find(5), false]
]
```

Tento seznam dvojic byl pak zpracováván metodou `procsss_orders`.

```
def process_orders(list_of_pairs)
  list_of_pairs.each do |order, expedite|
    # handle an order
  end
end
```

Na pořadí prvků ve dvojicích záleží!

```
class OrdersController
  def build_order_list(params)
```

```
        [order, flag]
    end
end

class Orders
    def process_orders(pairs)
        pairs.each do |order, flag| ... end
    end
end
```

Jiný příklad. Jednoduchá forma *Connascence*.

```
Customers.find(["last_name = ?", "Weirich"], "age")

def find(conditions, ordered_by)
    ...
end
```

A složitější forma *Connascence*.

```
Customers.find(["last_name = ?", "Weirich"], "age", 12, 24, [’first_name’, ’last_name’])

def find(condition, order_by, limit, offset, slected)
    ...
end
```

Takovou složitou formu CoP je lépe převést na CoN

```
Customers.find(
    :conditions => ["last_name = ?", "Weirich"],
    :order_by => "age",
    :limit = > 12,
    :offset => 24,
    :select => [’first_name’, ’last_name’])

def find(options={})
    ...
end
```

Connascence of Position při testování. Následující příklad při testování vybírá prvníhoo uživatele z databáze.

```
def test_user_can_do_somethin_interesting
    user = User.find(:first)
    ...
end
```

Na různých počítačích/databázích, u různých vývojářů může být tímto prvním uživatelem jiný člověk. Případně pokud nění stanoveno nějaké pořadí, může databázový stroj pokaždé vrátit jiného uživatele. To pak vede k chybám které se projevují podle toho na kterém počítači/databázi byl test spuštěn. Tomuto případu se vyhneme, když zadáme konktrétního uživatele z databáze.

```
def test_user_can_do_somethin_interesting
    user = User.find(:first)
    user = User.find(:first)
    user = User.find(:first)
```

```
        user = User.find_by_name("Jim")
    ...
end
```

56.3.4. Rule of Degree

Convert high degrees od connascence into weaker forms of connascence

56.3.5. Connascence of Meaning

```
<input type="checkbox" value="2" />
<input type="checkbox" value="1" />

if params[:med][id] == "1"
    mark_given(id)
elsif params[:med][id] == "2"
    mark_not_given(id)
end
```

Takováto situace je velmi nešťastná, protože nám uniká souvislost. Kód je křehký, citlivý na změnu hodnoty. Změna hodnoty není na první pohled vidět. Takovou situaci řešíme například použitím konstant. Tedy CoM⟶CoN

```
MED_GIVEN = "1"
MED_NOT_GIVEN = "2"
```

Kód pak vypadá mnohem čitelněji.

```
<input type="checkbox" value="<%= MED_GIVEN %>" />
<input type="checkbox" value="<%= MED_NOT_GIVEN %>" />

if params[:med][id] == MED_GIVEN
    mark_given(id)
elsif params[:med][id] == MED_NOT_GIVEN
    mark_not_given(id)
end
```

Uvedený příklad je příklad pravidla: „Do not use magic numbers“.

56.3.6. Contranascence

```
class Node
    ...
end
```

56.3.7. Connascence of Algorithm

add_check_digit("31415972") ⟶ "314159728"

```
def add_check_digit(digits)
    check_sum = digits.split(//).
        inject(0) {|r,n| r+n.to_i } % 10
    digits + ((10 - check_sum) % 10).to_s
end

def check?(digits)
    check_sum = digits.split(//).
        inject(0) {|r,n| r + n.to_i } % 10
    check_sum == 0
end
```

```
def add_check_digit(digits)
   digits + ((10 - check_sum(digits)) % 10).to_s
end

def check?(digits)
   check_sum(digits) == 0
end

def check_sum(digits)
   digits.split(//).
      inject(0) {|r,n| r + n.to_i } % 10
end
```

56.3.8. Závěr

Obrázek 56-2. Connascence

* Static * Dynamic
  * Connascence of Name * Connascence of Execution
  * Connascence of Type * Connascence of Timing
  * Connascence of Meaning * Connascence of Value
  * Connascence of Agorithm * Connascence of Identity
  * Connascence of Position * Contranascence

Obrázek 56-3. Rules

* Rule of Locality
* Rule of Degree

Kapitola 57. Refaktorizace

* ***Attributy:*** *id="refaktorizace"*

Odkazy:

- Refactoring in Ruby (http://my.safaribooksonline.com/9780321647917)
- Extreme Programming Explained () by Kent Beck

Co to vlastně je „refaktorizace“? Refaktorizace je proces, kdy malými změnami upravujeme kód tak, aby byl čitšlnější, tedy snáze srozumeitelný člověku.

A co není refaktorizace? Refaktorizace zcela jistě není programování nových vlastností a rozšiřování funkcionality.

Důležité: Vždy rozlišujte, jestli právě refaktorizujete, nebo programujete.

- *refactor as we go* — refaktorizujte průběžně
- *keep the system running at all times* — v každém ukamžiku, po každé malé refaktorizaci musí být program funkční
-
-

Obrázek 57-1. Refaktorizační cyklus:

začínáme s funkčním, testovaným kódem
while je možné zjednodušit kód
  vyberte nejhorší problém (*smell*)
  vyberte refaktorizační metodu
  aplikujte ji
  zkontrolujte testy
end

Pradivdla jednoduchého návrhu:

1. Projdou všechny testy.
2. *Communicates every intention important to the programmers.*
3. Neexistují žádné duplicity v kódu, logice nebo znalostech.
4. Neobsahuje nadbytečný kód.

Obrázek 57-2. TDD/BDD mikrporoces

RED: pište nové testy a kontrolujte že neuspějí
GREEN: opravte kód nejjednodušším (naivním) způsobem, aby testy uspěly
REFACTOR: transformujte kód na nejjednodušší možný (odstraňováním zápachu), který uspokojí všechny testy
Opakujte postup v několikaminutových cyklech

Pravidlo tří: Poprvé něco prostě uděláte/napíšete. Podruhé když programujete podobnou věc prostě zkopírujete předešlý kód. Potřetí, když narazíte na stejný případ, refaktorizujete.

57.1. Vůně a zápachy (*Smells*)

* *V anglicky psané literatuře se používá termín smell (vůně, zápach). Zatím se nejsem jist českou terminologií, a mohu ji bez upozornění měnit.*

Zápachy k popsání:

- komentáře
- Divergent Change (Refactoring: Ruby Edition, Pg. 77)
- Shotgun Surgery (Refactoring: Ruby Edition, Pg. 78)
- Feature Envy (Refactoring: Ruby Edition, Pg. 78)
- Data Clumps (Refactoring: Ruby Edition, Pg. 79)
- Primitive Obsession (Refactoring: Ruby Edition, Pg. 79)
- Case Statements (Refactoring: Ruby Edition, Pg. 80)
-
- Temporary Field
- Message Chains
-
-
-

57.1.1. Opakování kódu

* *Duplikování kódu.*

Odkazy:

- DRY
-

Nejhorší vůní, nebo spíše zápachem v programu je opakování kódu. Stejnýu kód na více místech programu způsobuje při změně největší problémy. Opakování kódu je porušním nejdůležitějšího pravidla, pravidla DRY

57.1.2. Dlouhé metody/funkce

*

Pokud je zápis metody příliš dlouhý je to signálem že nemusí být v pořádku. Metody by měly obsahovat tak přibližně do 7 řádků kódu.

Dlouhé metody rozdělíme na více metod.

57.1.3. Příliš mnoho metod ve třídě

*

Má-li třída příliš mnoho metod, je to známkou že je něco v nepořádku. Možná se snažíte ve třídě dělat příliš mnoho věcí. Je na čase se zamyslet, nejde li taková třída rozložit na více tříd. Připadně část funkcionality izolovat do modulů (Mixin).

57.1.4. Příliš mnoho argumentů

FIXME:

57.1.5. Komentáře

FIXME:

57.2. Refaktorizační postupy

*

Různá zatím netříděná pravidla:

- Make the code clean first and then use a profiler to deal with performance issues.
- Thus an important aspect of improving design is to eliminate duplicate code. DRY

57.2.1. Refaktorizace do bloků

Odkazy:

- Refactoring Ruby with Blocks (http://blog.ethanvizitei.com/2008/09/refactoring-ruby-with-blocks.html) [2008-09-05]
-

FIXME:

57.2.2. Šablona

* *Šablona refaktorizačního postupu.*

Odkazy a reference:

-
-

57.2.2.1. Summary

Šablona

57.2.2.2. Motivace

Šablona

57.2.2.3. Mechanizace

Šablona

57.2.2.4. Příklady

Šablona

Kapitola 58. Metaprogramování

* ***Attributy:*** *id="metaprogramming"*

  Odkazy:

  - Metaprogramming (http://en.wikipedia.org/wiki/Metaprogramming) z Wikipedie
  - Metaprogramming in Ruby (http://ruby-metaprogramming.rubylearning.com/)
  - Metaprogramming Ruby: Program Like the Ruby Pros (http://pragprog.com/titles/ppmetr/metaprogramming-ruby)
  - A Ruby Metaprogramming Introduction (http://practicalruby.blogspot.com/2007/02/ruby-metaprogramming-introduction.html)
  - Ruby Metaprogramming techniques (http://ola-bini.blogspot.com/2006/09/ruby-metaprogramming-techniques.html)
  - MetaProgramming - Extending Ruby for Fun and Profit (http://www.infoq.com/presentations/metaprogramming-ruby) v délce 1:00:53 [2007-12-07]
  -

* *Z "Ruby for Fun and Profit" by Dave Thomas*

  - Classes are open
  - Definitions are active
  - All methods calls have a receiver
  - Classes are objects

* *Přidání metody jen a pouze do jednoho objekut.*

```
a = "cat"
def a.encrypt
  tr 'a-z', 'b-za'
end
```

Vytvoří anonymní třídu. Do této třídy vloží metodu `encrypt` a tuto anonymní třídu udělá supertřídou třídy `String`.

Všechny definice jsou aktivní.

```
class Logger
  if ENV['DEBUG']
    def log(msg)
      STDERR.puts "LOG: " + msg
    end
  else
    def log(msg)
    end
  end
end
```

Ruby při načítání programu ze zdrojového suboru vytvoří AST (Abstract Syntax Tree) který reprezentuje daný soubor. Tuto činnost provádí parser. Kompilátory či interprety jiných jazyků dělají totéž. Rozdíl je v tom, že Ruby vykonává tento AST. V kompilovaných jazycích a řadě interpretovaných jazyků interpret pokračuje tím, že z AST vytobí bajtkód. Teprve tento bajtkód je vykonáván. Kompilované jazyky mohou dále pokračovat a přeložit bajtkód do instrukcí pro konkrétní procesor a tyto zapsat do tzv. vykonatelného souboru zvaného často *binárka*.

```
class Demo
  puts self        # => Demo
  puts self.class  # => Class
```

```
end
```

58.1. Anonymní třídy

*

Odkazy:

- Ruby: Creating Anonymous Classes (http://blog.jayfields.com/2008/02/ruby-creating-anonymous-classes.html) [2008-02-24]
-

Bežnou neanonymní třídu definujeme takto.

```
class Person
    ...
end
```

Třídu můžeme také definovat anonymně, voláním `new`. Třídy `Class`.

```
Person = Class.new do
    ...
end
```

Další zajímavé zápisy k prozkoumání a popsání.

```
class Person
  class << self
    def who2
      #
      true
    end
  end
end

class << Person
  def who2
    #
  end
end
```

Můžeme přidat metodu nikoliv do třídy ale do samotného objektu, tedy instance třídy. K tomu můžeme použít dva různé zápisy:

```
class << foo
    def bar
        puts "hello world"
    end
end

def foo.bar
    puts "hello, world"
end
```

Z hlediska Ruby jsou oba zápisy ekvivalentní.

Tři ekvivalentní zápisy definování metody třídy:

```
def String.hello
  puts "hello"
end

class String
  def self.hello
    puts "hello"
  end
end

class String
  class << self
    def hello
      puts "hello"
    end
  end
end
```

58.2. Singleton

Odkazy:

- The Ruby singleton class (http://ola-bini.blogspot.com/2006/09/ruby-singleton-class.html) [2006-09-24]
- 59.2.7

Kapitola 59. Návrhové vzory

Example Design Patterns in Ruby

* *chapter id="design-patterns" xreflabel="Návrhové vzory"*

> *Each pattern describes a problem which occurs over and over again in our environment, and then describes the core od the solution to that problem, in suach a way that you can use this solution a million times over, without ever doing it the same way twice.*
>
> *Christopher Alexander*

Odkazy:

- Example Design Patterns In Ruby (http://www.rubygarden.org/ruby?ExampleDesignPatternsInRuby)

* *Popsat co jsou to návrhové vzory.*

Text kapitoly

Struktura kapitoly návrhových vzorů

1. Creational Design Patterns
    a. Abstract Factory
    b. Abstract Session
    c. Factory
    d. Builder
    e. Prototype
    f. Singleton
    g. Borg
2. Structural Patterns
    a. Adapter
    b. Bridge
    c. Composite
    d. Decorator
    e. Facade
    f. Flyweight
    g. Proxy
3. Behavorial Design Patterns
    a. Prostředník (*Mediator/Colleague*)
    b. Command
    c. Interpreter
    d. Iterator
    e. Chain of Responsibility

f. Memento

g. Observer

59.1. Ukázkový vzor

59.1.1. Klasifikace

TEXT:

59.1.2. Smysl (*Intent*)

TEXT:

59.1.3. Alias (*Also Known As*)

TEXT:

59.1.4. Motiv

TEXT:

59.1.5. Aplikovatelnost

TEXT:

59.1.6. Struktura vzoru a popis vzoru

TEXT:

59.1.7. Účastníci vzoru

TEXT:

59.1.8. Spolupráce

TEXT:

59.1.9. Důsledky

TEXT:

59.1.10. Implementace

TEXT:

59.1.11. Příklad vzoru

TEXT:

59.1.12. Známé použití

TEXT:

59.1.13. Související vzory

TEXT:

59.2. Creational Design Patterns

FIXME: doplnit

59.2.1. Abstract Factory

Odkazy

- AbstractFactory Pattern (http://www.rubycolor.org/db/AbstractFactory.en.html)

Provide an interface for creating families of related or dependent objects without specifying their concrete classes.

Poskytuje/zavádí rozhraní/interface pro tvorbu rodiny souvisejících nebo závislých objektů bez specifikace jejich konkrétních tříd.

Ruby automatically implements

Příklad 59-1. Abstract Factory Pattern

```
class Foo; end
class Bar; end

# Here is the use of the Abstract Factory pattern
def create_something( factory )
    new_object = factory.new
    puts "created a new #{new_object.type} with a factory"
end

# Here we select a factory to use
create_something( Foo )
create_something( Bar )
```

Příklad 59-2. Abstract Factory Pattern

```
def create_something_with_block
    new_object = yield
    puts "created a new #{new_object.type} with a block"
end

def create_something_with_proc( &proc )
    new_object = proc.call
    puts "created a #{new_object.type} with a proc"
end

create_something_with_block { Foo.new }
create_something_with_block { Bar.new }
create_something_with_proc { Foo.new }
create_something_with_proc { Bar.new }
```

59.2.2. Abstract Session Pattern

Object-oriented frameworks are structured in terms of client/server relationships between objects; an objects services are invoked by client objects through the operations of its interface. A common design requirement is for a server object to maintain state for each client that it is serving. Typically this is implemented by returning handles or untyped pointers to the client that are used to identify the per-client data structure holding its state. The lack of strong typing can lead to obscure errors that complicate debigging and maintenance.

The Abstract Session pattern provides a way for an object to store per-client state without sacrificing type-safety or efficiency. A service object, rather than providing a client with a handle to be passed as an argument to the operations of its abstract interface instead creates an intermediate "session" object and returns a pointer to the session object bak to the client. The session object encapsulates the state information for the client which owns the session and is only exposed to the client as an abstract interface through which the client can access the service's functionality with full type-safety. When the client invokes operations of the session, the session co-operates with the service object to complete the operation. When the client has finished using the service, it "releases" the session, after which any pointers to the session object are invalid.

Příklad 59-3. Abstract Session Pattern

```
class Server
    def initialize
 @client_id = 0
    end

    def session
 @client_id += 1
 Session.new( self, @client_id )
    end

    def service_client( session )
 puts "servicing client #{session.client_id}"
    end
end


class Session
```

```
    attr :client_id

    def initialize( server, client_id )
 @server = server
 @client_id = client_id
    end

    def service
 @service.service_client( self )
    end
end

server = Server.new
client1 = server.session
client2 = server.session

client1.service
client2.service

# Returns:
# servicing client 1
# servicing client 2
```

59.2.3. Factory

FIXME: dopsat:

59.2.4. Factory Method

Klasifikace

- Class
- Creational

Odkazy/Zdroje

- Ilja Kraval, Design Patterns v OOP

Alias

- Virtual Constructor (virtuální konstruktor)

59.2.4.1. Motiv

* *Vypsáno z knihy Ilji Kravala. NEPUBLIKOVAT!*

Jedno použití Factory Method jsme již zavedli ve vzoru Abstract Factory, pouze jsme v té chvíli nevědeli, že se jedná právě o využití tohoto vzoru.

59.2.5. Builder

Klasifikace.

Object
Creational

Smysl. Odděluje

59.2.5.1. Smysl

Odděluje konstrukci složeného objektu (tj. postup jeho tvorby) od jeho reprezentace (tj. od jeho konkrétního složení)

59.2.6. Prototype

Klasifikace

- Object
- Creational

59.2.6.1. Smysl

Zavádí skupinu objektů vznikajících klonováním prototypovaných instancí pomocí jednotného rozhraní.

59.2.6.2. Motiv

* *Vypsáno z knihy Ilji Kravala. NEPUBLIKOVAT!*

Statické jazyky mají jednu velkou výhodu -- jsou typově bezpečné. Na druhou stranu práce s typy bývá mnohdy velmi btížná.

59.2.7. Singleton

všichni za jednoho, jeden za všechny

Ensure a class has only one instance, and provide a global point to access it.

V standardní instalaci knihoven je i implementace vzoru Singleton.

Avšak moduly mohou být také použity jako singleton objekty. Modul je v Ruby implementován jako objekt. Funkce modulu jsou implementovány jako metody instance objektu modul, a stav modulu je implementován jako proměnné instance objektu modul. Potom můžeme předa odkaz na modul stejně, jako odkaz na jiný objekt.

Singleton můžeme vytvořit za běhu programu. Můžeme to udělat například takto

```
MyObject = Object.new
def MyObject.foo; ...; end
def MyObject.bar; ...; end
```

nebo

```
MyObject = Object.new
class && MyObject
    def foo; ...; end
    def bar; ...; end
end
```

Definování funkcí modulu je jen komplikovaná cesta definováni metod singletonu. Například

```
module Mod
    def jedna; end
    def dve; end
    module_function :jedna, :dve
end
```

je jen zkratka pro

```
Mod = Module.new
class && Mod
    def jedna; end
    def dve; end
end
module Mod
    private
    def jedna; end
    def dve; end
end
```

```
class SingletonClass

  private_class_method :new

     def SingletonClass.create(*args, &block)
        @@singleton = new(*args, &block) unless @@singleton
        return @@singleton
     end
end

require 'singleton'

class Singltn
  include Singleton

  def initialize
    raise NotImplementedError
  end;
end;

x = Singltn.instance;
```

Sengleton podle C2 (http://c2.com/cgi/wiki?RubySingleton) je vytvořen technologií Mixing. Instance singletonu se nevytvářejí voláním metody `.new` ale voláním `.instance`

```
# $Id: singleton1.ses,v 1.1 2003/11/19 23:54:35 radek Exp $
require 'singleton'
```

```
true
class SClass
    include Singleton
end
SClass
```

59.2.8. Borg

we all are one

Popis tohoto programového vzoru se nachází v ASPN Python Cookbook (http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/66531)

Příklad 59-4. Implemetace programového vzoru Borg

```
class Borg:
    __shared_state = {}
    def __init__(self):
        self.__dict__ = self.__shared_state
```

Příklad 59-5. Použití definované třídy Borg

```
import borg
class Test(borg.Borg):
    def __init(s):
        borg.Borg.__init__(s)
```

59.3. Structural Patterns

59.3.1. Adapter

Klasifikace

- Class nebo Object (možné dvě varianty)
- Structural

Alias

- Wrapper

59.3.1.1. Smysl

Object scope: Zavádí objekt jako spojku mezi klienta, který očekává určitý interface a neznámý interface, takže klient může používat cizí interface.

Class scope: Zavádí třídu spojující neznámý a očekávaný interface, takže klient může používat cizí interface.

59.3.2. Bridge

FIXME: dopsat

Odděluje abstrakci od implementace, takže lze obě dvě měnit nezávisle na sobě.

59.3.3. Composite

FIXME: dopsat

Skládá objekty do stromové struktury, přičemž klient přistupuje k prvkům stromu jednotným způsobem, t.j. klient přistupuje k prvku bez rozlišení, zda se jedná o uzel větvení stromu nebo o koncový prvek.

59.3.4. Decorator

FIXME: dopsat

Přidává další přídavnou funkcionalitu k objektu dynamicky, čímž zavádí flexibilní alternativu k dědění.

59.3.5. Facade

FIXME: dopsat

Zavádí jeden nebo více rozhraní pro řízený přístup k subsystému, čímž zjednodušuje přístup klienta ke složitému systému.

59.3.6. Flyweight

FIXME: dopsat

Řeší problém systémů s velkým počtem objektů pomocí sdílení.

59.3.7. Proxy

FIXME: dopsat

Zavádí zástupce resp. držitele před objekt tak, že kontroluje přístup k objektu.

59.4. Behavorial Design Patterns

59.4.1. Prostředník (Mediator/Colleague, Controller)

* *Podle Linuxjournal č. 98 June 2002 strana 60*

Odkazy

- The Mediator (Behavioral) Design Patern by *Goplan Suresh Raj* (http://www.execpc.com/~gopalan/design/behavioral/mediator/mediator.html)
- Create Interactive Dialog Boxes by *Doug Farrell* (http://www.devx.com/premier/mgznarch/vbpj/2001/04apr01/pc0401/pc040
- Mediator (http://www.dofactory.com/patterns/pattern_mediator.asp)

Příklad 59-6. Třída `Mediator`

Slouží k distribuci zpráv mezi objekty jenž nejsou svázány. Například mezi objekty na formuláři v GUI.

```
class Mediator:
    def __init__(self):
        pass

    def ColleagueChanged(self, control, event):
        self._ColleagueChanges(control, event)

    def _ColleagueChanged(self, control, event):
        pass
```

Příklad 59-7. Třída `Colleague`

```
class Colleague:
    def __init__(self, mediator):
        self.mediator = mediator

    def Changed(self, colleague, event):
        self._Changed(colleague, event)

    def _Changed(self, colleague, event):
        self.mediator.ColleagueChanged(colleague, event)
```

59.4.1.1. The Problem

One of the goals of object-oriented design is to distribute behavior among different objects. This kind of partitioning is good since it encourages reuse.

- Sometimes, the interaction between these objects become so much that every object in the system ends up knowing about every other obejct. Lots of of such interactions prevent an object from working without the support of a lot of other...

59.4.1.2. Solution

Mediator

59.4.1.3. Smysl

Zavádí objekt jako prostředníka mezi jinými objekty, který oddělí a následně zprostředkuje jinak složitou komunikaci mezi mnoha objekty, takže tyto objekty nemusí mít přímé vazby mezi sebou.

59.4.2. Command (Action)

FIXME: dopsat

Zapouzdřuje požadavek do podoby objektu (objektové proměnné), takže lze s požadavkem pracovat jako s každou jinou proměnnou, což vede k možné parametrizaci požadavků, dynamické tvorbě seznamu požadavků, dosazení požadavku za jiný apod.

59.4.3. Interpreter

FIXME: dopsat

Zavádí reprezentaci gramatických pravidel pomocí modelu tříd a k tomu zavádí odpovídající interpret pomocí operací objektů z těchto tříd.

59.4.4. Iterator (Cursor)

FIXME: dopsat

Zavádí pro klienta jednoduchý a přehledný způsob sekvenčního přístupu ke složité struktuře objektů, přičemž tato struktura zůstane klientovi skryta.

> **Poznámka:** Ruby má standardně podporu pro iterátory v sobě.

59.4.5. Chain of Responsibility

FIXME: dopsat

Zavádí flexibilně měnitelný řetěz objektů propojených stejným rozhraním k předání a zpracování nějakého požadavku. Klient může odeslat požadavek libovolnému objektu z tohoto řetězu a tento požadavek bude v řetězu zpracován.

59.4.6. Memento (Token)

Smysl: Aniž by došlo k porušení principu zapouzdření, objekt vydává svůj stav klientovi za účelem možnosti návratu tuhoto objektu do původního stavu.

Klasifikace

- Object
- Behavioral

59.4.6.1. Motiv

* *Vypsáno z knihy Ilji Kravala. NEPUBLIKOVAT!*

V mnoha případech nastává situace, že se u objektu zavolá určitá operace a poté se zjistí, žy by bylo záhodno, aby se objekt vrátil do toho stavu, v kterém byl těsně před zavoláním oné operace. Ukazuje se, že není dobrým řešením pro návrat do původního stavu zavádět u objektů inverzní operace. Mnohdy jsou totiž přechody inverzních operací zpět nepřesné, resp. „nevratné“.

59.4.6.2. Struktura vzoru

* *Vypsáno z knihy Ilji Kravala. NEPUBLIKOVAT!*

```
   Memento
-state
----------
+GetState()
+SetState()
```

59.4.7. Observer (Dependents)

Zavádí možnost sledování změn u objektu tak, že když objekt změní stav, ostatní objekty na tuto změnu zareagují, přičemž nedojde k přímé vazbě od sledovaného objektu k těmto objektům.

Klasifikace

- Object
- Behavioral

59.4.8. State (Object for States)

Provádí změnu stavu objektu výměnou vnitřního objektu reprezentujícího stav.

59.4.9. Strategy

Definuje množinu algoritmů, které jsou kompatibilně vyměnitelné.

59.4.10. Template Method

Zavádí scénář na abstraktnější horní úrovni předka složený z několika polymorfních metod. Dědicové používají scénář tak, že plymorfní operace přepíší (vyplní) a scénář zavolají.

59.4.11. Visitor

Představuje operaci, která může účinkovat na prvky objektové struktury, aniž by se měnil kód ve třídách těchto prvků. Zavádí flexibilní alternativu k přidávání kódu polymorfních operací do tříd.

59.5. Vzory ve vývoji

59.5.1. Přistupovač (Accessor)

59.5.1.1. Problém

Někdy je potřeba zajistit k jednomu objektu přístup vícero objektům.

59.5.1.2. Řešení

```
# o - objekt
o1 = o
o2 = o
```

`o1` a `o2` jsou vlastně ukazatelé na objekt `o`.

59.5.1.3. Problém

.

59.5.2. Zámek (Lock)

Odkazy

- FIXME: CHECKLock pattern (http://www.castle-candenza.demon.co.uk/lock.htm)

59.5.2.1. Problém

Někdy potřebujeme implementovat do objektů mechanismus zamykání.

```
Objekt (atributy, metody)
   Prostředník pro přístup k objektu
```

59.5.2.2. Řešení

FIXME: dopsat

59.5.3. Database code, examples ans patterns

- The Design of a Robust Persistence Layer For Relational Databases (http://www.ambysoft.com/persistenceLayer.html)

Robustní persistentní vrstva

- různé druhy persistentních mechanismů / database backends
- soubory (flat files)
- relační databáze (SQL)
- ...

Metody persistentního objektu

.read() / .retrieve()

načtení objektu z databáze, obnovení sebe sama z databáze

.write() / .save()

uložení objektu, aktuálního stavu, do databáze. Uložení seba sama do databáze.

.delete()

odstranění objektu z databáze

Příklad 59-8. PersistenObject

```
class PersistentObject:   # abstract class
    self.oid            # identifikace objektu v databázové vrstvě
    isProxy
    isPersistent
    timeStamp
    def save(self): pass
    def retrieve(self): pass
    def delete(self): pass
```

59.6. Concurrent programming

*

59.7. Ostatní

* *Jiné nebo zatím neklasifikované návrhové vzory*

59.7.1. Reaktor (*Reactor*)

* ***Attributy:*** *id="pattern.Reactor"*

The reactor design pattern is a concurrent programming pattern for handling service requests delivered concurrently to a service handler by one or more inputs. The service handler then demultiplexes the incoming requests and dispatches them synchronously to the associated request handlers.

Odkazy:

- Reactor pattern (http://en.wikipedia.org/wiki/Reactor_pattern) na Wikipedii
- 28
-
-

*

-

VII. Různé

Směs ruzných inforamcí.

Kapitola 60. Joyau

*

Odkazy:

- Github: Kode / Joyau (http://github.com/Kode/Joyau)
- Joyau v1.0 Released; A Ruby Interpreter Now with Socket Support (http://www.psp-hacks.com/2010/01/13/joyau-v1-0-released-a-ruby-interpreter-w-socket-support/) [2010-01-13]
- Ruby Interpreter (http://www.pspsx.com/psp-apps/ruby-interpreter/)
-

Joyau je Ruby přeložené pro Sony PSP. Součástí je i řada knihoven pro práci s PSP.

A ruby interpreter for PSP, with some extension so that it could do things such as image loading, ...

60.1. Kousky kódu

*

```
require 'joyau/init'
Joyau.init(:intrafont, :audio)
    # Do something
Joyau.stop
```

60.1.1.

Odkazy:

- Vraiment utiliser IRB sur la PSP (http://joyau.scrapping.cc/posts)
-

60.2. Nástroje

*

60.2.1. Remote IRB

*

Odkazy:

- Vraiment utiliser IRB sur la PSP (http://joyau.scrapping.cc/posts/12)
-

```
$ irb
irb(main):001:0> IRB::WorkSpace.remote_ip='272.131.93.395'
```

60.3. Popis modulů a metod

*

60.3.1. Joyau

*

60.3.1.1. metody modulu Joyau

60.3.1.1.1. timestamp()

*

60.3.1.2. Joyau::Timer

*

```
t = Joyau::Timer.new
puts t.time
```

60.3.1.2.1. getTime(), time()

*

60.3.1.2.2. pause()

*

60.3.1.2.3. paused?(), paused()

*

60.3.1.2.4. reset()

*

60.3.1.2.5. resume()

*

60.3.1.3. `Joyau::Wlan`

*

60.3.1.3.1. `button_enabled?`

*

Vrací `true`, je li povolená WiFi (Wlan).

60.3.1.3.2. `configs`

*

Seznam konfigurací WiFi.

```
Joyau::Debug.puts Joyau::Wlan.configs.collect{|ip,name| "#{ip}:#{name}"}.join("\n")
```

60.3.1.3.3. `connect`(access_point, timeout)

*

Připojení k síti. Jako parametr *`access_point`* je zadáno číslo konfigurace ze seznamu konfigurací. Seznam konfigurací je dostupný pomocí metody 60.3.1.3.2. Parametrem *`timeout`* říkáme kolik sekund se má maximálně čekat na připojení k síti.

60.3.1.3.4. connected?

*

60.3.1.3.5. disconnect

*

60.3.1.3.6. `init`

*

Tuto metodu musíme volat před voláním funkce socket. Pokud používáme nový způsob inicializace modulů/subsystémů, nemusíme se tímto zabývat.

60.3.1.3.7. `ip`

*

Pokud jsme připojení, vrátí aktuální ip adresu.

```
Joyau::Debug.puts Joyau::Wlan.ip
```

60.3.1.3.8. `stop`

*

Zastaví modul/subsystém WiFi.

Kapitola 61. Emacs

* ***Attributy:*** *id="emacs"*

  Odkazy:

  - Emacs Code Browser (http://ecb.sourceforge.net/)
  - http://notes.free.railshosting.cz/emacs/ruby-rinari
  -
  -

  Tutoriály a videa:

  - emacs-demo (http://platypope.org/yada/emacs-demo/)
  -

61.1. Ruby a Emacs

Nastavení a speciální nastavení Emacsu.

- Emacs Wiki (http://www.emacswiki.org/cgi-bin/wiki.pl?CategoryOutline)

61.1.1. Mód `hideshow`

Mod ’hideshow’

```
(defun ruby-custom-setup ()
; [other stuff omitted...]
   (add-to-list 'hs-special-modes-alist
                '(ruby-mode
                   "\\(def\\|do\\)"
                   "end"
                   "#"
                   (lambda (arg) (ruby-end-of-block))
                   nil
                   ))
   (hs-minor-mode t)
)

(add-hook 'ruby-mode-hook 'ruby-custom-setup)
```

61.1.2. Skrývání bloků

Nastavení skrývání bloků přes „selective-display“.

```
;; ~/.emacs excerpt
;;   Use function keys f9 through f12 as a 'zoom' control.
(defun zoom-way-out() (interactive)
   (set-selective-display 0))
(defun zoom-way-in() (interactive)
   (set-selective-display 2))
(defun zoom-out() (interactive)
   (set-selective-display
      (if selective-display
```

```
         (if (or (= selective-display 0) (= selective-display 10))
         0
      (+ selective-display 2))
      0)))
(defun zoom-in() (interactive)
   (set-selective-display
      (if selective-display
         (if (= selective-display 0)
            10
            (if (= selective-display 2)
               2
               (- selective-display 2)))
         10)))

(global-set-key [f9] 'zoom-way-out)
(global-set-key [f10] 'zoom-out)
(global-set-key [f11] 'zoom-in)
(global-set-key [f12] 'zoom-way-in)
```

61.1.3. Anotace kódů

* *Podle 1.12 How can I annotate Ruby code with its result (http://www.rubygarden.org/iowa/faquotum)*

```
str = "Billy" + " Bob"          # => "Billy Bob"
str[0,1] + str[2,1] + str[-2,2] # => "Blob"
```

Integrace do Emacsu

```
(defun ruby-xmp-region (reg-start reg-end)
  "Pipe the region through Ruby's xmp utility and replace the \
region with the result."
  (interactive "r")
  (shell-command-on-region reg-start reg-end
      "ruby -r xmp -n -e 'xmp($_, \"%l\t\t# %r\n\")'"
      t))
```

61.2. Ruby on Rails a Emacs

Odkazy:

- HowToUseEmacsWithRails (http://wiki.rubyonrails.org/rails/pages/HowToUseEmacsWithRails)
- Better rhtml-mode fo (http://www.blik.it/2007/03/22/better-rhtml-mode-for-emacs/)
- Emacs and Ruby (http://www.hyperionreactor.net/node/43)
- Rinari (http://rubyforge.org/projects/rinari)
- Emacs on Rails Screencast (http://emacsonrails.drozdov.net/)

61.3. ECB minor mode

* ***Attributy:*** *id="emacs.ecb"*

Odkazy:

-
-

Tabulka 61-1. Některé klávesové skratky

C-c . t	ecb-toggle-layout	Switch between different layouts.

61.4. Formátování Ruby kódu

*

Odkazy:

- change emacs ruby-mode indent to 4 spaces (http://stackoverflow.com/questions/2111041/change-emacs-ruby-mode-indent-to-4-spaces) na stack overflow
-

```
# Local Variables:
# mode: ruby
# ruby-indent-level: 4
# End:
```

Kapitola 62. Jednoduché příklady

Příklady a ukázky

* *Určeno k rozpracování, případně k přeřazení do jiných kapitol.*

Tato kapitola obsahuje jednoduché samostatné příklady. Byly převzaty převážně z mailing listu ruby-talk, ale i z jiných zdrojů.

Zajímavé ukázky kódu. Tyto střípky jsem skládal z různých zdrojů, a většinou pocházejí z příspěvků jež se objevily v mailinglistu ruby-talk.

* *FIXME: 'mailinglistu' - přeložit*

62.1. Slovník necitlivý na velikost písmen *Case Insensitive Hash*

V mailing listu se objevil dotaz jak implementovat slovník který není citlivý na velikost písmen. T.j kde `h['a']` je stejný prvek jako `h['A']`.

Nejjednodušší navržený způsob je předefinovat metodu `has_ikey?` takto

```
class Hash
    def has_ikey?(key)
        self.keys.map {|key| key.upcase }.member? key.upcase
    end
end
```

Velikou nevýhodou tohoto řešení je pomalost, tedy výpočetní naročnost degraduje použití slovníku.

Jiným řešením je změnit chování Ruby nastavením proměnné `$=`

```
$= = true    # case insensitive
```

tento způsob však ovlivní všechny části Ruby.

Asi nejlepším řešením je definovat si vlastní třídu objektů s požadovanou vlastností

```
class CIHash < Hash
    alias get_old []
    alias set_old []
    def [](key)
        return get_old(key.to_s.upcase)
    end
    def []=(key, value)
        return set_old(key.to_s.upcase, value)
    end
end
```

62.2. FIFO

Alan Chen `ruby-talk(44336)`

Well, you use a Array push and shift. If you want this behavior could be coded as:

```
class Simple_FIFO
  def initialize
    @data = Array.new
  end

  def push(val)
    @data.push(val)
  end

  def pop(val)
    @data.shift(val)
  end
end
```

Error handling is left as an excercise to the reader :) - alan

62.3. Jak použít unixovou fifo

Vypsáno z `ruby-talk`. Autor: Nobu Nakada.

Maggioe Xiao se ptá: [Jestli chci vytvořit fifo soubor ruby způsobem, tím myslím bez volání mkfifo, jak to mám udělat.]

Nobu Nakada odpovídá: [Zkus `syscall`.]

```
require 'syscall'
require 'sys/stat'
class File
    def self.mkfifi(path, mode = 0666)
        Syscall.mknod(path, Stat::IFIFO|mode, 0)
    end
end
File.mkfifo("/tmp/foo")
```

Další informace na http://www.ruby-lang.org/en/raa-list.rhtml?name=syscall

62.4. Method ancestors for class Method — reflection

* *condition="author": NEPUBLIKOVAT, RCR*
* *Opsáno/vypsáno z článku na . (http://www.rubygarden.org)*

I have some structures that contain objects that refer to those structures. I wanted to write an inspect method. This RCR would allow me to prevent recursion. If Method#== was true when the method is the same method of the same instance, and Method#eql? was true when it was the same method of an instance of the same class (i.e. broader) I could do this to prevent mutual recursion when doing inspect:

```
class Array
   def inspect
      history = Method.ancestors
```

```
        me = history.shift
        if history.detect {|x| x == me)
           # This is a self reference
           result = "Array #{self.id}n"
        else
           # We're examining some new
           # object
           result = "Array #{self.id} [n"
           self.each { |element|
               result += "   #{element.inspect"}n"
           }
           result += "]n"
        end
        return result
     end
  end
```

for example. This would also help in [ruby-talk:52142] I expect. The reason that caller doesn't satisfy my needs here is that it does not tell you which instance it is referring to, and using its information involves manipulating strings. I believe that this suggestion fits in with the philosophy of "eveything being an object".

62.5. Idiomy

* *Rozpustit do jednotlivých konstrukcí, vytvořit šablonu XSL* `idiom title/ para/* /idiom`

62.5.1. If

Normální konstrukci if

```
if myvar
    return myvar
else
    return another_value
end
```

můžeme s pomocí operátoru `||` napsat takto

```
return myvar || another_value
```

a při přiřazení můžeme nahradit operátor `||=` tedy místo

```
myvar = another_value unles myvar
```

nebo

```
if myvar.nil?
    myvar = another_value
end
```

můžeme psát

```
myvar ||= another_value
```

62.5.2. Zavolej metodu objektu, pokud máš objekt

Pokud máme v proměnné objekt a chceme zavolat joho metodu, pak pokud může nastat případ že v proměnné objekt není použijeme

```
if myvar
    return myvar.size
else
    return nil
end
```

toto můžeme pomocí operátoru && přepsat

```
return myvar && myvar.size
```

v případě že potřebujeme vrátit jinou hodnotu než `nil` napíšeme

```
return myvar && myvar.size || 0
```

62.5.3. Spusť kód jen když je spuštěn tento soubor

Někdy je dobré přidat do souboru vykonatelný kód který se vykoná jen je li soubor spuštěn jako hlavní program.

```
if $0 == __FILE__
    do_stuff
end
```

62.5.4. Chci aby byl objekt destruován (zničen) když se dostane z rozsahu.

```
Resource.use( identifier ) do |resource|
    process( resource )
end
# resource is now closed
```

Implementace metody třídy musí použít příkaz (konstrukci) begin...ensure aby jsme si byli jisti že zdroj je vždy uvolněn.

```
def Resource.open( identifier )
    resource = Resource.new( identifier )
    begin
        yield resource
    ensure
        resource.close
    end
end
```

62.5.5. Analýza argumentů programu

```
        opts = GetoptLong.new(
                [ "--referer",  "-f",   GetoptLong::REQUIRED_ARGUMENT ],
                [ "--verbose",  "-v",   GetoptLong::NO_ARGUMENT ],
                [ "--images",   "-i",   GetoptLong::REQUIRED_ARGUMENT ],
                [ "--help",             "-h",   GetoptLong::NO_ARGUMENT ]
        )

        opts.each do |opt, arg|
                print "1 Help\n" if #{opt} =~ "help"
                if opt =~ "help"
                        print "2 Help\n"
                end
                puts "Option: #{opt}, arg #{arg}"
        end

        opts.each do |opt, arg|
                if (if print "2 Help\n"
                   end)
                   print "1 Help\n"
                end
                puts "Option: #{opt}, arg #{arg}"
        end



        puts "Remaining args: #{ARGV.join(', ')}"
```

```
% ruby test.rb  -h
```

62.5.6. Fronta

```
class EmptyQueue
  def initialize
    @waiting = Queue.new
    @que = Queue.new
  end
  def push(obj)
    @waiting.pop
    @que.push obj
  end
  def pop
    @waiting.push true
    @que.shift
  end
end
```

62.5.7. Ladicí tisky

```
You may have the interpolation occurring in a block that is evaluated
only in debug mode:

def debug_print( &a_block )
  if $DEBUG
    print a_block.call()
  end
end
# Then use:
#   debug_print{"Hello"}
# instead of
#   debug_print("Hello")
# .
```

62.5.8. Čtení souboru

```
lines.readlines("somefile").each do |line|
        puts line if (/start/../end/) === line
end

myFile.each do |line|
    ... = line.chomp.split(/regexp/)
    ...
end
```

62.5.9. Mutex

```
  def method_missing(method, *args, &block)
    @mutex.synchronize do
      @internalArray.send(method, *args, &block)
    end
  end
```

62.5.10. Thread Safe Array

```
I wrote this ThreadSafeArray class. Seems to work, but I thought I'd throw
it out here for review. Any holes in this approach?

class ThreadSafeArray
  def initialize
    @mutex = Mutex.new
    @internalArray = []
  end

  def ary
    @internalArray
  end

  def method_missing(method, *args, &block)
```

```
    @mutex.lock
    begin
      @internalArray.send(method, *args, &block)
    ensure
      @mutex.unlock
    end
  end
end

Chris Morris
```

62.5.11. Delegování iterátoru

Delegating each

```
class C
    include Enumerable
    ...
    def each(&proc)
        @myarray.each do |e|
            proc.call(e)
        end
    end
end
```

Nebo lépe

```
def each(&block)
    @myarray.each(&+block)
end
```

S použitím modulu `Forwardable` pak

```
require 'forwardable'

class C
    extend Forwardable
    def_delegators(:@myarray, :each)
end
```

62.6. Technologie

62.6.1. Mutex (Mutual Exclusion)

FIXME:

Mechanizmy

- `thread`
- `monitor`

- sync

```
#!/usr/bin/env ruby
# $Id: mutex2.rb,v 1.1 2005/10/04 08:52:07 radek Exp $

require 'thread'
mutex = Mutex.new

count1 = count2 = 0
difference = 0
counter = Thread.new do
    loop do
        mutex.synchronize do
            count1 += 1
            count2 += 1
        end
    end
end
spy = Thread.new do
    loop do
        mutex.synchronize do
            difference += (count1 - count2).abs
        end
    end
end
sleep 1
mutex.lock
p count1
p count2
p difference
```

```
#!/usr/bin/env ruby
# $Id: mutex3.rb,v 1.1 2005/10/04 08:52:07 radek Exp $

require 'thread'
mutex = Mutex.new
cv = ConditionVariable.new

a = Thread.new {
    mutex.synchronize {
 puts "A: I have critical section, but will wait for cv"
 cv.wait(mutex)
 puts "A: I have critical section again! I rule!"
    }
}

puts "(Later, back at the ranch...)"

b = Thread.new {
    mutex.synchronize {
 puts "B: Now I am critical, but am done with cv"
 cv.signal
 puts "B: I am still critical, finishing up"
    }
}

a.join
```

```
b.join
```

62.7. Unikód

* *Poznámky k používání unikódu v Ruby*

62.7.1. Nezapracované podklady

62.7.1.1. Email „Re: multi-language support in Ruby“ od Nobu Nakady

```
At Mon, 16 Dec 2002 04:07:42 +0900,
Shannon Fang wrote:
> I used
>
>     text.gsub!(/&#(\d+);/) do $1.to_i.chr end
>
> to process it, and chr reported that 8217 is too large. I don’t know if
> it is related to double-byte support? The entire article is in English,
> IE display is "don’t" for the &#8217;... although the apostrophe looks
> a little weird...

Integer#chr makes only sinle byte char.  If you want UTF8
string,

    text.gsub!(/&#(\d+);/) {[$1.to_i].pack("U")}

--
Nobu Nakada
```

62.8. Funkcionální styl programování

* *Poznámky k používání funkcionálního programování v Ruby*

62.8.1. Nezapracované podklady

62.8.1.1. Email „Re: functional programming "style"“ od <jcb@iteris.com> (MetalOne)

```
>
> one of my love examples - is a "quick" sort one-liner:
>
> qs() = ()
> qs(x:xs) = qs( a←xs | a<x ) || x || qs( a←xs | a >= x )
>
```

```
Here is quick sort in Ruby
def qsort(arr)
    return [] if arr.length <= 0
    x, *xs = arr
    qsort(xs.select{|y| y<=x}) + [x] + qsort(xs.select{|y| y>x})
end
```

62.9. Ruby a Forth

- ratlast (http://www3.sympatico.ca/mark.probert/download/files/ratlast_0_1.tar.gz)

ATLAST je jednoduchý Forth od John Walker

ratlast je embeded forth (ATLAST). Ratlast je od Mark Probert <probertm@acm.org>

62.9.1. Zpráva „Re: Ruby and Forth“ z comp.lang.forth od Mark Probert <probertm@acm.org>

```
Ruby is an excellent scripting language that I personally
prefer to the likes of perl and python.

ATLAST is a light-weight embeded Forth, after F83.

Putting ATLAST into Ruby exposes the Ruby community to Forth,
and it makes it fun to do the things that Forth does well
in Forth rather than Ruby.

For example, I can write (in Ruby)

    require 'Atlast'            # include the ATLAST extension

    t = Atlast.new              # create a Forth interpreter

    t.expr(": fact (n -- n!)  dup 1 do i * loop ; ")
    n = 1
    until n > 10
        fact = t.run("#{n} fact .")     # (1..10)! in a loop
        puts "#{n}! = #{fact}"          # print n! = fact \n
    end

or
    t.expr(": linear ( a b -- ) create swap , , does> dup r> @ * r> 4 + @ + ;")
    t.expr("3 17 linear aline")
    y = t.run("10 aline .")
```

62.10. Lisp

V ruby-talk proběhlo několik dotazů na Lisp napsaný v Ruby. Jednou z odpovědí, od Yukihiro Matsumoto byl odkaz do RAA na balíček rogue (http://www.ruby-lang.org/raa/list.rhtml?name=rogue)

Hal E. Fulton měl vtipnou připomínku popisující způsob jak získat interpret Lispu v Ruby: [For a Lisp interpreter in Ruby, here's a recipe. First of all, it's much easier to write a Ruby interpreter in Lisp; do that, and then turn the universe inside-out.]

Akira Tanaka pak poslal krátké řešení, „Interpreter Lispu (nebo lambda kalkulu) v Ruby za 10 minut“

```
#!/usr/bin/env ruby
# $Id: attr_list_accessor.rb,v 1.1 2005/10/04 08:52:07 radek Exp $
# $Source: /home/radek/cvs/ruby-book/example/attr_list_accessor.rb,v $
#- Copyright (C) 2003 Radek Hnilica

class Class
    def attr_list_accessor (*symbols)
 symbols.each do |s|
     class_eval <<-EOS
         def add_#{s}(elem)
      (@#{s} ||= []) << elem
         end
         def each_#{s}(&block)
      (@#{s} ||= []).each(&block)
  end
     EOS
 end
    end
end


class Test
    attr_list_accessor :foo, :bar
end
```

62.11. Jednořádkové skripty

62.11.1. Quine

```
_=["print'_=',_.inspect,';',_\n"];print'_=',_.inspect,';',_
```

62.11.2. Password sniffer

```
tcpdump -w- | perl -le 'while (sysread STDIN, $b, 80)
     { print map /[[:graph:]]/ ? $_ : "." => split // => $b; }'
```

62.11.3. Výpočet Hammingovy vzdálenosti mezi dvěma IP adresami

Například mezi adresami `224.0.0.1` a 224.2.63.254

```
perl -le '@a = split /\./ => shift; @b = split /\./ => shift;
    for $i (0 .. 3) {
        for $j (0 .. 7) {
            (($a[$i] >> $j ^ $b[$i] >> $j)) & 1 and $h++;
        }
    }
    END { print $h }' 224.0.0.1 224.2.63.254
```

62.12. Nezpracované ukázky z IRC

62.12.1. IRCNet: ruby, 2008-04-20 19:56

ion: There was a small discussion #elsewhere about ways to map an IPv4 address to something that is easy to remember and e.g. pass along in a phone discussion. This is what i came up with:
ion: >> [83,145,237,222].merge(256).split(2048).map {|i| RFC2289::WORDS[i] }.join(' ')
ion: => "MOW ABET MIND"
ion: >> %w{MOW ABET MIND}.map {|w| RFC2289::INDEX[w] }.merge(2048).split(256)
ion: => [83, 145, 237, 222]
ion: ...or just use DNS. ;-)
ytti: nice
ytti: where is Array#merge defined?
ion: Hm, that would work for phone numbers as well. My Finnish phone number would be DAM HEFT TREE, and with the 358 p
fix added, AX MERT DAVY DARE.
ion: I defined Array#merge and Numeric#split. I can put the code somewhere if you're curious.
ytti: nah, nevermind

VIII. Reference

zkouška referencí

Referenční přehled některých tříd a jejich metod.

I. File

Bleee

Jméno

bb — účel bb

UDPSocket

SYNOPSIS

```
uaaa
```

DESCRIPTION

*

II. Třídy

File

Jméno

`File` — Standardní knohovna pro práci se soubory a adresáři.

File

SYNOPSIS

```
uaaa
```

DESCRIPTION

*

Bleee

Jméno

`bb` — účel bb

UDPSocket

SYNOPSIS

```
uaaa
```

DESCRIPTION

*

IX. Přílohy

V této části jsou přílohy. Je to část Sprovozňujeme ruby o získání a sprovoznění interpretu jazyka Ruby. Dále pak přehled jazyka a jeho konstrukcí, popis některých tříd a modulů, a několik dalších příloh.

Příloha A. Sprovozňujeme ruby

Kompilace, instalace a spouštění ruby

* ***Attributy:*** *id="booting-ruby" xreflabel="Sprovozňujeme ruby"*
* *Tuto kapitolu by bylo lépe přesunout mezi dodatky.*

A.1. Jak získat ruby

* *Jak se dostaneme k Ruby.*

Ruby je možno nahrát z ftp://ftp.ruby-lang.org/pub/ruby/. Jsou zde zdrojové kódy různých verzí je tu i archív Mailing listu ruby-talk.

A.2. Instalace z binární distribuce

A.2.1. Instalace na Debian GNU/Linux 5.0 (Lenny)

*

A.2.2. Instalace na Debian GNU/Linux 4.0 (Etch)

* ***Attributy:*** *id="ruby-on.debian.etch"*

V Debian Etch je ruby ve verzi 1.8.5 a je rovněž k dispozici vývojová verze 1.9.0.

```
# aptitude install ruby
# aptitude install rubygems
```

Upgrade gems na nejnovější verzi:

```
# gem install rubygems-update
```

A.2.3. Instalace z balíčků na Debian GNU/Linux 3.1 (Sarge)

* ***Attributy:*** *id="ruby-on.debian.sarge"*

Budeme instalovat novější verzi 1.8.2.

```
# aptitude install ruby
```

Nainstalují se: `libruby1.8` `ruby ruby1.8`. Tím je hotová holá instalace. Můžeme se přesvědčit.

```
$ ruby --version
ruby 1.8.2 (2005-04-11) [i386-linux]
```

Nyní přiinstlaujeme další balíčky dle potřeby. Budeme-li na tomto počítači taky psát/vyvíjet programy, bude se nám hodit `ruby-elisp` do Emacsu, `irb` na interaktivní zkoušení, `ri` jenž je interaktivní referencí k Ruby, `libdbd-pg-ruby`/`libdbd-sqlite-ruby`/`libdbd-mysql-ruby` pro připojení k databázovým serverům, a řada dalších balíčků.

A.2.4. Instalace na Debian GNU/Linux 3.0 (Woody)

* ***Attributy:*** *id="ruby-on.debian.woody"*

```
# apt-get install ruby
```

A.2.5. Instalace na MS Windows

FIXME:

A.3. Překlad ze zdrojových kódů

FIXME:

A.3.1. Kde získat zdrojové kódy

FIXME:popsat kde se nacházejí zdrojové kódy a jak získat aktuální verzi z CVS

Zdrojové kódy všech verzí ruby od verze 1.0 až po současnost jsou ke stažení na ftp://ftp.ruby-lang.org/pub/ruby/

A.3.2. Překlad a instalace

FIXME:

Kompilace z balíčků v Debian/GNU Linux je jednoduchá. Pokud máme v `/etc/apt/sources.list`odkazy na zdroje, já jsem použil vnitrofiremní cache

```
deb-src http://ferit:9999/main woody main contrib non-free
deb-src http://ferit:9999/non-US woody/non-US main contrib non-free
deb-src http://ferit:9999/main testing main contrib non-free
deb-src http://ferit:9999/non-US testing/non-US main contrib non-free
deb-src http://ferit:9999/main unstable main contrib non-free
deb-src http://ferit:9999/non-US unstable/non-US main contrib non-free
```

zdroje si stáhneme

```
$ apt-get -t unstable source ruby1.7
```

a skompilujeme

```
$ cd ruby-beta-1.7.2.0cvs2002.07.13
$ dpkg-buildpackage -b -uc -rfakeroot
```

Jediný zádrhel by mohl být v chybějících balíčcích pro kompilaci. Mě konkrétně chyběl `tk8.3-dev`.

Trochu odlišná je kompilace z aktuálních zdrojů z CVS. Nejdříve musíme získat tyto zdroje Poté provedem konfiguraci

```
$ autoconf
$ ./configure --prefix=$HOME
```

a můžeme kompilovat

```
$ make
```

přeložený interpret vyzkoušíme

```
$ make test
```

a nejsou-li žádné problémy, aspoň u mě proběhl test bez problémů, můžeme **ruby** nainstalovat

```
$ make install
```

* *condition="author"*

Postup použitý na cvs verzi dne 2002-12-08

```
$ make clean
$ autoconf
$ ./configure --prefix=$HOME
$ make
$ make test
$ make install
```

* *condition="author"*

Postup použitý na cvs verzi dne 2002-12-17. Za příkazy je uvedena doba jejich trvání na počítači `kvark`.

```
$ make clean             0:08:01
$ autoconf               0:00:13
$ ./configure --prefix=$HOME/opt/ruby-2002.12.17  0:04:51
$ make                   0:19:41
$ make test              0:01:06
$ make install           0:01:21
```

* *FIXME: Aktualizovat následující odstavec.*

Abych mohl snadno spouštět a testovat různé verze Ruby a jiných programů, používám spouštěcí skripty v adresáři `$HOME/bin`. Například pro verzi 1.7.3 z cvs získanou dne 2002-12-17 mám vytvořen skript `$HOME/bin/ruby-2002.12.17` Spouštěcí skript `$HOME/bin/ruby` je pak symbolickým odkazem na některou verzi jenž bez větších problémů funguje.

```
$ cd $HOME/bin
$ ln -s ruby-2002.12.17 ruby
```

Poznámka: Tento způsob již nepoužívám.

A.3.2.1. Překlad aktuální verze z CVS

Po aktualizaci zdrojů z CVS viz FIXME:Získání aktuálních zdrojů z CVS si tyto zkopíruji do adresáře

```
$ cd $HOME/source
$ mkdir ruby-1.8.0-2003.01.07
$ cd ruby-1.8.0-2003.01.07
$ cp -a $HOME/mirror/cvs/ruby/ruby/* .
```

a přeložím

```
$ autoconf                                          0:00:34
$ mkdir $HOME/opt/ruby-1.8.0-2003.01.07
$ ./configure --prefix=$HOME/opt/ruby-1.8.0-2003.01.07          0:05:19
$ make clean
$ make                                              0:20:38
$ make test
$ make install
```

A.3.2.2. Překlad ze zdrojů Debian Testing/Unstable

Při hledání nečeho úplně jiného jsem narazil na tento krátký postup (http://lists.rubyonrails.org/pipermail/rails/2006-May/040813.html)

Poznáma k sestavování Ruby-1.8.4 na Debian Sarge.

* *Následující postup osobně odzkoušet a upřesnit.*

1. Přidejte do `sources.list` odkaz na zdroje debian testing nebo unstable
2. Aktualizovat lokální databázi

   ```
   # aptitude update
          # apt-get source ruby1.8
   ```

3.

   ```
   # apt-get build-dep ruby1.8
   ```

4. Install devscripts
5.

   ```
   $ cd ruby1.8; debuild -us -uc
   ```

6. Dostanete následující balíčky:
7. Nainstalujte balíčky

A.3.2.3. Překlad 1.9.1-p378 na SuSE 9.0

Ukázka konfigurace a překladu Ruby verze 1.9.1-p378 na SuSE Linux verze 9.0.

```
# cd /usr/local/download
# wget ftp://ftp.ruby-lang.org/pub/ruby/1.9/ruby-1.9.1-p378.tar.bz2
# cd /usr/local/src
# tar xjvf ../download/ruby-1.9.1-p378.tar.bz2
# cd ruby-1.9.1-p378
# ./configure --program-suffix=1.9 --prefix=/usr/local/ruby-1.9.1
# make
# make test
# make install
# make distclean
```

A.3.2.4. Překlad 1.9.1-p378 na Debian Etch

*

Odkazy:

- compiling ruby 1.8.5 w/ openssl on Debian (Etch/testing) and FreeBSD in $HOME (http://plasti.cx/2007/02/03/compiling-ruby-1-8-5-w-openssl-on-debian-etch-testing-and-freebsd-in-home)
- CentOS - Ruby on Rails (http://articles.slicehost.com/2009/4/7/centos-ruby-on-rails)
- Installing Ruby on Rails on Debian/Ubuntu (http://wiki.rubyonrails.org/getting-started/installation/linux-ubuntu)
-

Nejedná se o překlad programu tak jak jsme zvyklí. V tomto případě překládám ruby tak, abych jej mohl distribuovat jako součást jiného systému. Nebude se tedy instalovat do systémových adresářů ale do adresářů "aplikace" která se distribuuje na jiné servery pomocí kopírování.

Překlad provádím na čistě nainstalovaném virtuálním stroji, abych měl jistotu že mi nikde nic nepřebývá a nikde nic nechybí. Potřebuji vědět přesně co vše potřebuji. Pro jistotu aktualizujeme systém.

```
# aptitude update
# aptitude upgrade
```

Poté nainstalujeme potřebné balíčky.

```
# aptitude install bzip2 g++ make
Následující NOVÉ balíky budou nainstalovány automaticky:
  binutils cpp cpp-4.1 g++-4.1 gcc gcc-4.1 libc6-dev libmudflap0
  libmudflap0-dev libssp0 libstdc++6-4.1-dev linux-kernel-headers
# aptitude install zlib1g-dev libssl-dev libpth-dev libsqlite3-dev libreadline-dev
Následující NOVÉ balíky budou nainstalovány automaticky:
  libncurses5-dev libpth20 libsqlite3-0
```

Stáhl jsem si poslední v danou chvíli existující verzi z ftp.ruby-lang.org (ftp://ftp.ruby-lang.org/pub/ruby/1.9/).

```
# mkdir /usr/local/download
# cd /usr/local/download
# wget ftp://ftp.ruby-lang.org/pub/ruby/1.9/ruby-1.9.1-p378.tar.bz2
```

Nyní rozbalení, konfigurace a překlad a instalace.

```
# cd /usr/local/src
# tar xjvf ../download/ruby-1.9.1-p378.tar.bz2
# cd ruby-1.9.1-p378
# mkdir -p /usr/local/cl/lib/etch/bin
# export PATH=/usr/local/cl/lib/etch/bin:$PATH
# ./configure --program-suffix=1.9 --prefix=/usr/local/cl/lib/etch --enable-pthread
# make
# make test
# make install
# make distclean
```

Instalace některých gemů:

```
# cd /usr/src/cl/lib/etch/bin
# export PATH=/usr/local/cl/lib/etch/bin:$PATH
# export GEM_HOME=/usr/local/cl/lib/etch/lib/ruby1.9/gems/1.9.1
# gem1.9 update --system
# gem1.9 install sqlite3
```

```
# gem1.9 install eventmachine
```

Pro ruby 1.9 se používá gem sqlite3 na rozdíl od ruby 1.8 kde se používá sqlite3-ruby.

* *sudo apt-get install libsqlite3-dev sqlite3 sqlite3-doc sudo gem install sqlite3-ruby*
* *?? libiconv-1.11*
* *$ ldd /usr/local/cl/lib/etch/bin/ruby1.9*
* *./configure –enable-pthread*
* *libreadline5-dev*
* *libthread*
* *Pokud přenáším skompilované ruby a nainstalované gemy, je potřeba na cílových počítačích doinstalovat případné závislosti. Například gem sqlite3 potřebuje mít nainstalovaný deb balíček sqlite3.*

A.3.2.5. Překlad 1.9.1-p378 na Debian Lenny

*

Odkazy:

•

•

Podobně jako v předchozím případě, kdy jsem překládal ruby 1.9.1-p378 na Debian Etch, jedná se o podobný způsob překladu tentokrát na Debian Lenny.

Překlad provádím na čistě nainstalovaném virtuálním stroji, abych měl jistotu že mi nikde nic nepřebývá a nikde nic nechybí. Potřebuji vědět přesně co vše potřebuji. Pro jistotu aktualizujeme systém.

```
# aptitude update
# aptitude upgrade
```

Poté nainstalujeme potřebné balíčky.

```
# aptitude install bzip2 g++ make
Následující NOVÉ balíky budou nainstalovány automaticky:
  binutils cpp cpp-4.1 g++-4.1 gcc gcc-4.1 libc6-dev libmudflap0
  libmudflap0-dev libssp0 libstdc++6-4.1-dev linux-kernel-headers
# aptitude install zlib1g-dev libssl-dev libpth-dev libsqlite3-dev libreadline-dev
Následující NOVÉ balíky budou nainstalovány automaticky:
  libncurses5-dev libpth20 libsqlite3-0
```

Stáhl jsem si poslední v danou chvíli existující verzi z ftp.ruby-lang.org (ftp://ftp.ruby-lang.org/pub/ruby/1.9/).

```
# mkdir /usr/local/download
# cd /usr/local/download
# wget ftp://ftp.ruby-lang.org/pub/ruby/1.9/ruby-1.9.1-p378.tar.bz2
```

Nyní rozbalení, konfigurace a překlad a instalace.

```
# cd /usr/local/src
# tar xjvf ../download/ruby-1.9.1-p378.tar.bz2
# cd ruby-1.9.1-p378
# mkdir -p /usr/local/cl/lib/lenny/bin
# export PATH=/usr/local/cl/lib/lenny/bin:$PATH
# ./configure --program-suffix=1.9 --prefix=/usr/local/cl/lib/lenny --enable-pthread
# make
# make test
# make install
# make distclean
```

Instalace některých gemů:

```
# cd /usr/local/cl/lib/lenny/bin
# export PATH=/usr/local/cl/lib/lenny/bin:$PATH
# export GEM_HOME=/usr/local/cl/lib/lenny/lib/ruby1.9/gems/1.9.1
# gem1.9 update --system
# gem1.9 install sqlite3
# gem1.9 install eventmachine
```

* *sudo apt-get install libsqlite3-dev sqlite3 sqlite3-doc sudo gem install sqlite3-ruby*
* *?? libiconv-1.11*
* *$ ldd /usr/local/cl/lib/etch/bin/ruby1.9*
* *./configure –enable-pthread*
* *libreadline5-dev*
* *libthread*

A.4. Jak získat ruby

Ruby je možno získat z archívu ftp://ftp.netlab.co.jp/pub/lang/ruby, nebo z jeho zrcadel

* *FIXME:napsat seznam zrcadel do dodatku a odkázat se na něj*

Najdete tam poslední stabilní verzi ruby [1] stejně jako několik posledních vývojových verzí [2] . Také je k dispozici několik zkompilovaných binárních verzí pro různé platformy. Například binární distribuce Ruby pro Windows na http://www.pragmaticprogrammer.com/ruby/downloads/ruby-install.html.

Uživatelé rozličných distribucí linuxu si mohou prohlédnout archivy balíčků na svých instalačních médiích či archívech v síti.

Debian GNU/Linux. Verze Woody obsahuje balíčky

FIXME: doplnit

Verze Sarge obsahuje v době psaní této knihy balíčky

FIXME: doplnit

RedHat Linux. FIXME: doplnit

* *Doplnit informace o ostatních běžných distribucích. Suse, Mandrake, ...*

Další možností je získat zdrojové kódy ruby přímo z cvs archívu. Jak kódy poslední stabilní, tak také poslední vývojové verze, či jiné kterou se rozhodneme získat.

Postup je následující:

* *FIXME: ověřit, odzkoušet*

```
$ cd ~/work/cvs                                                    ❶
$ export CVSROOT=:pserver:anonymous@cvs.ruby-lang.org:/src
$ cvs login
(Logging in to anonymous@cvs.ruby-lang.org)
CVS password: anonymous                                            ❷
$ cvs -z4 checkout ruby
cvs server: Updating ruby
U ruby/.cvsignore
U ruby/COPYING
```

```
U ruby/COPYING.ja
...
U ruby/x68/fconvert.c
U ruby/x68/select.c
$
```

❶ Přepneme se do adresáře kam budeme zdrojový kód načítat. Já mám pro tyto účely vyhražen adresář `~/work/cvs`, v němž je každý projekt jako podadresář.

❷ Heslo se při vypisování nezobrazuje.

Takto získaný zdrojový kód můžeme kdykoliv aktualizovat příkazem

```
$ cvs update
```

A.5. Jak nainstalovat Ruby

Pod Debian/GNU Linuxem verze Woody je to jednoduché. Jako root zadám:

```
# apt-get install ruby
```

a vše potřebné je připraveno. Potřebuji-li novější verzi ruby, mohu ji nainstalovat z unstable

```
# apt-get -t unstable install ruby1.7
```

A.5.1. Zdroje odkud je možno instalovat

A.5.1.1. Debian archiv Akiry Yamady

Na stránce [4] je mimo krátkou noticku o knize [2] jíž je Akiro spoluautorem též odkaz na archiv Debianovských balíčků.

Do `/etc/apt/sources.list` vložíme řádky odkazující na Akirův archiv

```
deb http://deb.ruby-lang.org/debian unstable main contrib non-free
deb http://deb.ruby-lang.org/debian potato main contrib non-free
deb http://deb.ruby-lang.org/debian project/experimental/
```

A.6. Verze Ruby

*FIXME:

A.6.1. 1.6.x

Starší stabilní verze.

A.6.2. 1.7.x

Vývojová větev.

A.6.3. 1.8.0

Nejaktuálnější stabilní verze v době psaní této knihy.

A.6.4. Rubinius: git evanphx / rubinius 2010-06-04

*

```
$ git clone git://github.com/evanphx/rubinius.git
$ cd rubinius
$ ./configure --enable-llvm --skip-system
Configuring LLVM...
  Checking for existing LLVM tree: not found.
  Checking for prebuilt LLVM build...
    [ 100% (8259176 of 8259176) ]: done!
   No MD5 checksum for llvm-x86_64-unknown-linux-gnu.tar.bz2 available on server.
   Using LLVM library without checksum validation.
    Prebuilt packages updated.
  Unpacking prebuilt LLVM for x86_64-unknown-linux-gnu: done!

Checking for function 'backtrace': found!

Configured. Run 'rake' to build and run VM tests and rubyspecs
$ rake build
...
/bin/sh ./libtool --tag=CC   --mode=compile gcc -DHAVE_CONFIG_H -I. -I. -I/usr/local/include    -g -O2 -
libtool: Version mismatch error.  This is libtool 2.2.4, but the
libtool: definition of this LT_INIT comes from libtool 2.2.6.
libtool: You should recreate aclocal.m4 with macros from libtool 2.2.4
libtool: and run autoconf again.
make[2]: *** [regerror.lo] Error 63
make[2]: Leaving directory '/home/radek/src/rubinius/rubinius/vm/external_libs/onig'
make[1]: *** [all-recursive] Error 1
make[1]: Leaving directory '/home/radek/src/rubinius/rubinius/vm/external_libs/onig'
make: *** [all] Error 2
rake aborted!
Command failed with status (2): [cd vm/external_libs/onig; ./configure && m...]
```

A.7. Programy (Software)

A.7.1. Balíčky v Debian/GNU Linuxu

Balíčky dostupné v Debian/GNU Linuxu

ruby

: Interpret jazyka Ruby

ruby-examples

: Příklady, velmi vhodné pro seznamování se s jazykem.

rubybook

: Kniha o jazyku Ruby

ruby-elisp

: Mód pro zvýrazňování syntaxe pro editor emacs.

rubyunit

: Jednoduché testovací prostředí pro Ruby.

rdtool

: RD is Ruby's POD. RDtool is formater for RD.

libtk-ruby

: Tk interface for Ruby. This module is built on top of ruby-tcltklib-module.

  Doinstaluje si libtcltk-ruby.

libtcltk-ruby

: Tk interface for Ruby. Tcl/Tk library interface for Ruby. This modules provides low level interface to the Tcl/Tk library.

libcurses-ruby

: Curses interface for Ruby.

irb

: The Interactive Ruby

libmutexm-ruby

: yet another mutex library for Ruby

webrick

: Simple HTTP Server Toolkit for Ruby

libgtk-ruby

: Gtk+ interface for scripting language Ruby

libgnome-ruby

Gnome interface for scripting language Ruby.

Doinstaluje si `libgdk-imlib-ruby`.

libssl-ruby

OpenSSL interface for scripting language Ruby

libopenssl-ruby

OpenSSL interface for scripting language Ruby

libpgsql-ruby

PostgreSQL interface for scripting language Ruby

libmysql-ruby

mysql module for Ruby

libldap-ruby

Ruby/LDAP module providing an interface to OpenLDAP library

drb

distributed ruby

Doinstaluje si `libmutexm-ruby`

libreadline-ruby

Readline interface for Ruby

`rdoc`

version: 0.0.beta.1-1, dist: unstable Sid

Vytváří HTML a XML dokumentaci ze zdrojových souborů

A.7.2. Ostatní

ruby-gsl (http://www.ruby-lang.org/en/raa-list.rhtml?name=ruby-gsl)

GSL is the GNU Scientific Library, a collection of routines for numerical computing. ruby-gsl is a wrapper to be able to use those functions in Ruby.

A.8. Spuštíme ruby

```
$ ruby [options] [--] [programfile] [arguments]
```

Po spuštění `ruby` se „nic neděje“. Interpret nevypisuje žádnou výzvu (prompt), jen „tupě“ čeká na náš vstup. Chcete-li mít interpret s výzvou, který má obdobné chování jako u jiných jazyků (tcl, python, scheme, ...) musíte spustit program `irb`.

* *Popis spouštění ruby, výkonné soubory, kde v adresářích se nacházejí důležité soubory, irb*

A.8.1. Soubory a adresáře

Poznámka: Popisuji na Debian GNU/Linu 3.0 (Woody)

Vlastní interpret je v souboru `/usr/bin/ruby`. Knihovny jsou v adresáři `/usr/lib/ruby/1.6` a jeho podadresářích. V jednom z nich `/usr/lib/ruby/1.6/i386-linux` jsou i `.so` soubory dynamických knihoven.

A.8.2. Spouštění ruby

Program spustíme jednoduše

```
$ ruby
```

Program nevypisuje žádný prompt. Očekává na standardním vstupu text programu. Zkusíme jednoduchý program.

```
puts "Ahoj"
```

Po napsání programu ukončím vstup stiskem **Ctrl**+**d**. Ruby program analyzuje a vykoná

```
Ahoj
```

Pokud je program malý, můžeme jej předat Ruby na příkazové řádce.

```
# $Id: spoustime-ruby.ses,v 1.1 2003/11/19 23:54:35 radek Exp $
ruby -e '(1..7).each{|n| print "#{n} "}'
1 2 3 4 5 6 7

$ ruby -e '(1..7).each{|n| print "#{n} "}'
1 2 3 4 5 6 7
$
```

Tento způsob je obvyklý u celé řady programů jako jsou **sed**, **awk**, **perl**, **python** a spousty dalších. Používá se hlavně k snadnému začlenění do skriptů.

A.8.3. Interaktivní Ruby (`irb`)

* *Změnit název sekce. Například „Interaktivní práce s Ruby“.*

Jak jste si zajisté všimli, ruby je pro interaktivní práci nevhodný. Potřebujeme-li si vyzkoušet nápad, ověřit si jak se některé programové konstrukce chovají, či jen si s Ruby hrát, použijeme program irb (*Interactive Ruby*). Ten je přesně pro takové případy určen.

```
# $Id: interaktivni-ruby.ses,v 1.1 2003/11/19 23:54:35 radek Exp $
1+2
3
3*4
12
exit
```

Jak je z ukázky vidět. `irb` můžeme použít také jako kalkulátor. Jako programovatelný kalkulátor.

```
# $Id: fact.ses,v 1.1 2002/12/16 20:34:12 radek Exp $
```

```
def f(n)
    return 1 if n == 0
    n * f(n-1)
end
nil
f(0)
1
f(5)
120
f(50)
30414093201713378043612608166064768844377641568960512000000000000
```

* *Napsat skript který následující výstup převede do tvaru předchozí ukázky. T.j. na všech řádcích jenž začínají `irb(main):číslo:číslo[\>\*]` vše co je tza tímto začátkem obalíme tagem `userinput`*

* *condition="author"*
  Zkoušení různých způsobů „importování“ záznamu sezení.

```
# $Id: var.ses,v 1.1 2002/12/16 20:34:13 radek Exp $
irb(main):001:0> vyska = 3
3
irb(main):002:0> delka = 5
5
irb(main):003:0> sirka = 2
2
irb(main):004:0> vyska * delka * sirka
30
irb(main):005:0> def objem(a,b,c); a*b*c; end
nil
irb(main):006:0> objem(1, 2, 3)
6
irb(main):007:0>
```

A.8.3.1. Spuštění/vyvolání irb z kódu

Irb je možno vyvolat i z běžícího kódu. Na začátek skriptu vložíme

```
require 'irb'
```

na místě v kódu, kde chceme vyvolat interaktivní sezení s irb vložímě příkaz

```
IRB::start $STDIN
```

chceme-li, aby po ukončení interaktivního sezení pokračovalo vykonávání programu ukončujeme irb příkazem **quit**

Dalším způsobem zaručení návratu do programu je ošetřit událost `SystemExit`, například takto

```
require 'irb'
puts "Before IRB"
begin
    IRB::start $STDIN
rescue SystemExit
end

puts "Okie donkey"
```

Takto spuštěný **irb** (irb) má ovšem omezené možnosti. Chceme-li zkoumat a měnit obsahy proměnných, musíme to udělat trochu jinak. Především budeme muset zasáhnout do modulu `irb` a dodefinovat si metodu `start2` výsledný ukázkový příklad vypadá takto:

Příklad A-1. Start `irb` z programu

```
# $Id: invoke-irb.ses,v 1.1 2002/12/16 20:34:12 radek Exp $
sh: line 3: ./invoke-irb.rb: není souborem ani adresářem
sh: line 4: p: command not found
sh: line 5: a: command not found
sh: line 6: quit: command not found
```

Jednoduchá interakce s tímto programem pak může být třeba:

```
# $Id: invoke-irb.ses,v 1.1 2002/12/16 20:34:12 radek Exp $
sh: line 3: ./invoke-irb.rb: není souborem ani adresářem
sh: line 4: p: command not found
sh: line 5: a: command not found
sh: line 6: quit: command not found
```

Nobu Nakada mi poslal tento kousek kódu

```
require 'irb'
IRB.setup(nil)
irb = IRB::Irb.new(IRB::WorkSpace.new(binding))
IRB.conf[:MAIN_CONTEXT] = irb.context
a = 10
begin
    raise "#{a} left" if (a -= 1) > 0
rescue StandardError, ScriptError
    puts $!.to_s
    trap("SIGINT") {irb.signal_handle}
    catch(:IRB_EXIT) {irb.eval_input} or retry
end
```

A.8.3.2. Nezpracované texty

```
#!/usr/bin/ruby
require 'irb'
class << STDOUT
    def write(s)
        $stderr.puts "HERE #{s}"
    end
end
IRB.start
```

```
pigeon% b.rb
irb(main):001:0>
HERE NameError
HERE :
```

```
HERE undefined local variable or method 'aaa' for #<Object:0x401c6ce0>
HERE
HERE    from (irb):1
HERE
irb(main):002:0> HERE
pigeon%
```

Příklad A-2. Z dopisu Hiroshi Nakamury: `catch.rb`

```
# $Id: catch.ses,v 1.1 2002/12/16 20:34:12 radek Exp $
ruby: no such file to load -- catch (LoadError)
sh: line 5: quit: command not found
```

A.8.4. Běhové prostředí
Runtime Environment

A.8.4.1. Spustitelné skripty

Na `UNIX` kompatibilních systémech můžeme učinit spustitelným libovolný skript v jazyce Ruby tak, že na prvním řádku tohoto skriptu uvedem „magickou“ sekvenci *hash bang*

```
#!cesta_k_ruby
```

```
#!cesta_k_ruby
```

```
#!cesta_k_ruby
```

Na Debian GNU/Linux Woody je to

```
#!/usr/bin/ruby
```

Výhodnější je ovšem použít

```
#!/usr/bin/env ruby
```

Takovýto *hash bang* bude fungvat i na systémech kde sice `ruby` je, ale nevíme kde se nachází.

Souboru je třeba ještě nastavit příznak *executable*

```
$ chmod a+x skript.rb
```

A nyní jej již můžeme spustit

```
$ ./skript.rb
```

A.8.4.2. Co se děje při startu interpretu

* *userlevel="author"*
* *Popsat které soubory jsou čteny, proměnné zkoumány, ...*

A.8.4.3. Spustitelný kód v knihovnách

Když napíšeme skript, a ten chceme používat jednak jako program a pak jako knihovnu, potřebujeme mechanizmus jak zajistit odlišné chování v obou případech. Při načtení jako knihovny jiným skriptem příkazem `require 'moje_knihovna'` potřebujeme aby se nevykonával žádný kód, a naopak při přímém spuštění

```
$ ./moje_knihovna
```

potřebujeme kód vykonat. Takto odlišné chování v obou případech zajistíme touto konstrukcí přidanou na konec souboru

```
if $0 == __FILE__ ❶
    # kód se provede jen při přímém spuštění
    # například:
    run_test
end
```

❶ Podmínka je splněna je tehdy, pokud je jméno spuštěného programu/skriptu (`$0`) shodné se jménem soubor v němž se nachází (`__FILE__`).

A k čemu že je to dobré? Například je náš velký program rozdělen do řady menších souborů. V každém jsou související metody a třídy. Při spuštění programu se jednotlivé soubory jen načítají, zatímco když spustíme každy soubor samostatně, provedou se testy funkčnosti v něm naprogramované.

* *Více v testování, konkrétně „Primitivní testy“.*

Máme-li soubor s definicí nějakých objektů, který používáme jako knihovnu, potřebujeme do něj někdy umístnit výkonný kód. Jde o to, že se soubor chová jinak když jej nahráváme pomocí `require 'soubor.rb'` z jiného souboru a jinak když jej přímo spustíme

```
$ ./soubor.rb
```

Toto odlišné chování zajistíme přidáním kódu s podmínkou na konce souboru. Vypadá to nějak takto

```
if $0 == __FILE__
    # kód který se provede, například spuštění testovací metody
    test
end
```

Úplný spustitelný příklad

```
#!/usr/bin/env ruby
# $Id: hello-arun.rb,v 1.1 2002/10/06 09:23:58 radek Exp $
# $Source: /home/radek/cvs/ruby-book/example/misc/hello-arun.rb,v $

def hello
    puts "Hello, world."
end

if $0 == __FILE__
    hello
end
```

A.8.4.4. Obsluha chyb

* *userlevel="author"*
* *Popsat jakým způsobem jsou ošetřovány chyby z hlediska uživatelského spouštění programu.*

A.9. Proměnné prostředí

RUBYOPT

Aditional command-line options to Ruby; examined after real command-line options are parsed ($SAFE must be 0)

RUBYLIB

Aditional search path for Ruby programs ($SAFE must be 0)

RUBYPATH

With -S option search path for Ruby programs (defaults to PATH)

RUBYSHELL

Shell to use when spawning a process; if not set, will also check SHELL or COMSPEC

DNL_LIBRARY_PATH

Search path for dynamically loaded modules.

RUBYLIB_PREFIX

(Windows only) Mangle the RUBILIB search path by adding this prefix to each component.

Poznámky

V čase psaní této knihy 1.6.8 FIXME: ověřit

Opět, v čase psaní této knihy je poslední verzí 1.7.3.

Příloha B. Jazyk Ruby

* *V tomto dodatku má být stručný, témeř úplný přehled jazyka.*

B.1. Jazykové konstrukce

*

B.2. Zabudované a speciální proměnné a konstanty

Odkazy:

- Ruby Predefined Constants (http://www.tutorialspoint.com/ruby/ruby_predefined_constants.htm)
-

Proměnné prostředí:

`RUBYOPT`

Dodatečné přepínače na příkazové řádce. Jsou zkoumány po běžných přepínačích. (`$SAFE` musí být `0`)

`RUBYLIB`

FIXME:

`RUBYPATH`

FIXME:

`RUBYSHELL`

Shell (interpret příkazů) který se použije při spuštění procesu. Není-li nastaven, zkoumají se proměnné`SHELL` a `COMSPEC`.

`DNL_LIBRARY_PATH`

Cesta na které se hledají dynamicky zaváděné moduly.

`RUBY_LIBRARY_PREFIX`

FIXME:

Speciální proměnné ruby

`$DEBUG`

Proměnná určuje zdali se mají tisknout ladicí informace. Je nastavována přepínačem `-d` nebo `--debug` na příkazové řádce.

`ARGF`
$<

Parametry a přepínače s nimiž byl program spuštěn.

`ARGV`
`$*`

Parametry a přepínače s nimiž byl program spuštěn.

DATA

Vstupní proud který obsahuje vše co následuje za řádkem s textem `__END__`.

`ENV`

Proměnné prostředí.

`$ARGV` `$*`

FIXME:

`$CHILD_STATUS` `$?`

FIXME:

`$DEFAULT_INPUT` `$<`

FIXME:

`$DEFAULT_OUTPUT` `$>`

FIXME:

`$ERROR_INFO` `$!`

FIXME:

`$ERROR_POSITION` `$@`

FIXME:

`$FIELD_SEPARATOR` `$FS$;`

FIXME:

`$IGNORECASE` `$=`

FIXME:

`$INPUT_LINE_NUMBER` `$.`

FIXME:

`$INPUT_RECORD_SEPARATOR` `$RS` `$/`

FIXME:

`$LAST_MATCH_INFO` `$~`

FIXME:

`$LAST_PAREN_MATCH` `$+`

FIXME:

`$LAST_READ_LINE` `$_`

FIXME:

`$LOADED_FEATURES` `$"`

FIXME:

`$MATCH` `$&`

FIXME:

`$NR` `$.`

FIXME:

`$OUTPUT_FIELD_SEPARATOR` `$OFS` `$,`

FIXME:

`$\`
`$OUTPUT_RECORD_SEPARATPR` — *English*
`$ORS` — *English*

FIXME:

`$$`
`$PROCESS_ID`
`$PID` — *English*

FIXME: identifikační číslo běžícího procesu

`$POSTMATCH`
`$’`

FIXME:

`$PREMATCH`
`$‘`

FIXME:

`$.`

číslo řádku

`$_`

implicitní proměnná

`$=`

FIXME: řídí citlivost na velikost znaků. Má-li hodnotu `true` znamená to že část ruby není citlivá na velikost znaků, tj je. *case insensitive*.

`$/`

FIXME: tuším oddělovač polí, nemůže být regulární výraz.

`$KCODE`

FIXME: Nastaví interpretaci znaků v kódování. Jako hodnota se použije první znak z řetězce. Platné hodnoty jsou:

`"E"`	EUC-JP
`"S"`	Shift-JIS

`"U"`	UTF-8

všechny ostatní hodnoty jsou interpretovány jako "NONE" a znamenají jen ASCII (*ASCII only*)

FIXME: opravit dle http://www.ruby-lang.org/en/man-1.6

`$VERBOSE`

FIXME:

`$LOADPATH`
`$LOAD_PATH`
`$:`

Sezname (Array) adresářů, které se prohledávají když se pokoušíme něco „nahrát“ příkazem **require**. Například, pokud mám nějaké moduly v adresáři `/home/radek/lib/ruby`, rozšířím seznam prohledávaných adresářů o tento příkazem

```
$:.push "/home/radek/lib/ruby
```

`$DEBUG`

FIXME:

`$FILENAME`

FIXME:

`RUBY_VERSION`

FIXME:

`RUBY_RELEASE_DATE`

FIXME:

`RUBY_PLATFORM`

FIXME:

TRUE
FALSE

Synonyma pro logické hodnoty `true` a `false`.

NIL

Synonyma pro `nil`.

Příloha C. Popis některých tříd a modulů

*

Odkazy:

- Ruby Standard Library Documentation (http://www.ruby-doc.org/stdlib/)
-

Hierarchie některých tříd

- BasicSocket
    - IPSocket
        - TCPSocket
            - SOCKSSocket
            - TCPServer
        - UDPSocket
    - Socket
    - UNIXSocket
        - UNIXServer

C.1. UDPSocket

*

Klient

```
s = UDPSocket.new
s.send("Hola", 0, 'localhost', 1234)
```

Server

```
require 'socket'
host = 'localhost'
port = 1234

s = UDPSocket.new
s.bind(nil, port)
loop
    text, sender = s.recvfrom(20)
    remote_host = sender[3]

    puts "#{remote_host} poslal #{text}"
end
```

method	prototype

Tabulka C-1. class methods

method	prototype
new	UDPSocket.new(family = AF_INET) ⟶ aSession
open	UDPSocket.open(family = AF_INET) ⟶ aSession

Tabulka C-2.

method	prototype
bind	aSession.bind(hostName, port) ⟶ 0
connect	aSession.connect(hostName, port) ⟶ 0
recvfrom	aSession.recvfrom(len [, flags]) ⟶ anArray
send	aSession.send(aString, flags [, hostName, port]) ⟶ aFixnum

C.2. Socket

*

Tabulka C-3.

C.3. IPSocket

*

Tabulka C-4. class methods

getaddress	IPSocket.getaddress(hostName) ⟶ aString

Tabulka C-5. instance methods

addr	
peeraddr	

C.4. TCPSocket

*

Odkazy:

- Ruby Socket Programming (http://www.tutorialspoint.com/ruby/ruby_socket_programming.htm)
-

```
require 'socket'

hostname = 'localhost'
port = 2000

s = TCPSocket.open(host, port)
while line = s.gets
    puts line.chomp
end
s.close
```

C.5. File

*

Odkazy:

- Class File (http://ruby-doc.org/core/classes/File.html)
-

C.6. FileUtils

*

Odkazy:

- Module FileUtils (http://ruby-doc.org/core/classes/FileUtils.html)
-

III. Třídy

Bleee

Jméno

bb — účel bb

UDPSocket

SYNOPSIS

```
uaaa
```

DESCRIPTION

*

Příloha D. Přehled lidí jenž se kolem Ruby vyskytovali či vyskytují

Hal E. Fulton

FIXME: Pár slov k člověku.

Adresa: **FIXME:**

FIXME: Autor programu: (xref)

Yukihiro Matsumoto *a.k.a. Matz varlistentry id="PERSON.Yukihiro_Matsumoto" xreflabel="Yukihiro Matsumoto"*

Tvůrcerce Ruby a původce toho všeho.

Adresa:

Akira Tanaka *varlistentry id="PERSON.Akira_Tanaka" xreflabel="Akira Tanaka"*

FIXME: Pár slov k člověku.

Adresa: FIXME:

FIXME: Autor programu: (xref)

Akira Yamada *varlistentry id="PERSON.Akira_Yamada" xreflabel="Akira Yamada"*

Spoluautor [2] a tvůrce stranky [4].

Známé adresy:
FIXME:
FIXME:

Takaaki Tateishi

Pár slov k člověku.

Adresa:

Kirk Haines

Adresa: ,

Eric Hodel *varlistentry id="PERSON.Eric_Hodel" xreflabel="Eric Hodel"*

FIXME: Pár slov k člověku.

Adresa:

FIXME: Autor programu: (xref)

Pár odkazů. resume (http://segment7.net/me/resume.html)

Dave Thomas *varlistentry id="PERSON.Dave_Thomas" xreflabel="Dave Thomas"*

FIXME: Pár slov k člověku.

Adresa:

FIXME: Autor programu: Ruby on Rails(xref)

Pár odkazů.

Nobu Nakada

FIXME: Pár slov k člověku.

Adresa:

FIXME: Autor programu: (xref)

Example Glossary

This is not a real glossary, it’s just an example and work in progress.

CoC

: Convetion Over Configuration

KISS

: Keep It Simple Stupid.

YAGNI

: You Ain’t Gonna Need It.

YAGNI

: You Ain’t Gonna Need It.

Bibliografie

Knihy

[1Making Use Of Ruby] *Making Use Of Ruby*, Suresh Mahadevan, Madhushree Ganguly, Rashi Gupta, Shweta Bashin, Ashok Appu, Wiley, John & Sons, Incorporated, 047121972X.

[2Ruby Application Programming] *Ruby Application Programming (http://ssl.ohmsha.co.jp/cgi-bin/menu.cgi?ISBN=4-274-06461-1)*, Shugo Maeda, Yukihiro Matsumoto, Hidetoshi Nagai, Akira Yamda, 4-274-06461-1.

[3An Invitation to Ruby] *An Invitation to Ruby (http://w3.one.net/~jweirich/development/talks/invitationtoruby/cover.html)*, Jim Weirich.

[4Life with Ruby] *Life with Ruby (http://arika.org/index.en)*, Akiro Yamada.

[5] *Mnemonic (http://www.papermountain.org/MnemonicDoc/): File Mnemonic.rb*, Leslie Hensley.

[6] *TWiki MnemonicHome (http://www.papermountain.org/twiki/bin/view/Stuff/MnemonicHome)*, Leslie Hensley.

[7Nora rwiki] *Div (http://rwiki.moonwolf.com/rw-cgi.cgi?cmd=view;name=Nora)*.

Web Application Library

[8] *Ruby Application Archive*.

http://www.ruby-lang.org/raa/

[9] *ruby-talk*.

Mailing list pro uživatele Ruby.

* *FIXME: Doplnit informace o ruby-talk, adresy, archiv, ...*