

KAPITOLA 19

Ruby a webové aplikace

Ó, jak zamotanou síť jsme to upředli...!

– Sir Walter Scott, *Píseň posledního skotského barda*

Ruby je univerzální jazyk; nemůže být nazýván "jazykem webu". Webové aplikace (a webové nástroje obecně) ovšem patří mezi nejběžnější využití Ruby. Existuje mnoho způsobů, jak provádět vývoj webových aplikací v Ruby – od knihoven, které jsou malé a nízkoúrovňové, až po rozsáhlé frameworky (pracovní rámce), jež určují váš styl myšlení a kódování.

A začneme na spodním konci – podíváme se na knihovnu `cgi.rb`, která je standardem v Ruby.

19.1 – CGI programování v Ruby

Libovolný člověk obeznámený s programováním webových aplikací nepochybně již slyšel o termínu CGI (Common Gateway Interface). CGI bylo vytvořeno brzy po vzniku samotného webu, aby mohly existovat programově implementované stránky, a pro dosažení větší interakce mezi koncovým uživatelem a webovým serverem. Ačkoliv od doby jeho vzniku bylo představeno nespočetné množství náhradních technologií, CGI ve světě webu stále žije a daří se mu dobře. Velká část úspěchu a dlouhověkosti CGI může být přisuzována jeho jednoduchosti. Kvůli této jednoduchosti je například snadné implementovat CGI programy v libovolném jazyce. Standard CGI specifikuje, jakým způsobem bude proces webového serveru předávat data mezi sebou a jeho potomky. Většina této interakce se děje skrze standardní proměnné prostředí a proudy v operačním systému.

CGI programování (a pro úplnost – i samotné HTTP) je založeno na mechanismu bezstavového požadavku a odpovědi. Obecně lze říci, že jakmile je ustanoveno jediné TCP spojení, klient (což je obvykle webový prohlížeč) zahájí konverzaci prostřednictvím jednoduchého HTTP příkazu. Dva nejběžněji používané příkazy v tomto protokolu jsou GET a POST (k jejich významu se dostaneme za chvíli). Po vydání příkazu webový server odpovídá a zavírá výstupní proud.

Následující ukázka kódu, která je o něco málo pokročilejší než standardní "Hello, World!", názorně ukazuje, jak provést vstup a výstup přes CGI.

```
def parse_query_string
  inputs = Hash.new
  raw = ENV['QUERY_STRING']
  raw.split("&").each do |pair|
    name,value = pair.split("=")
    inputs[name] = value
  end
  inputs
end

inputs = parse_query_string
print "Content-type: text/html\n\n"
print "<HTML><BODY>"
print "<B><I>Hello</I>, #{inputs['name']}!</B>"
print "</BODY></HTML>"
```

Použití URL `http://mywebserver/cgi-bin/hello.cgi?name=Dali` vyprodukuje výstup "Hello, Dali!" ve vašem prohlížeči.

Jak už jsem zmínil dříve, existují dva hlavní způsoby, jak přistoupit k URL – HTTP metody GET a POST. Z nedostatku místa bohužel musím dát přednost jejich jednoduchému popisu před pečlivou definicí. Metoda GET je obvykle volána při kliknutí na odkaz nebo při přímém použití URL (jako tomu bylo v předchozím příkladu). Parametry jsou předávány prostřednictvím dotazovacího řetězce URL, který je CGI programům přístupný přes proměnnou prostředí `QUERY_STRING`. Metoda POST se nejčastěji používá v HTML formulářích. Parametry, které jsou odeslány prostřednictvím metody POST, jsou zahrnuty v těle zprávy a nejsou viditelné v URL. CGI programům jsou doručovány přes standardní vstupní proud.

Ačkoliv předchozí příklad byl velmi jednoduchý, cokoliv méně banálního by mohlo vést ke zbytečným zmatkům. Programy, které jsou potřebné pro spolupráci s několika HTTP metodami, nahráváním souborů, cookies, relacemi a dalšími složitostmi, je nejlepší řešit prostřednictvím univerzální knihovny pro práci s CGI prostředím. Ruby naštěstí poskytuje plnohodnotnou sadu tříd, které automatizují velkou část práce, již byste jinak museli dělat ručně.

Spousta dalších nástrojů a knihoven se pokouší vývoj CGI zjednodušit. Mezi nejlepší z nich patří `ruby-web` od Patricka Maye (dříve Narf). Pokud chcete nízkourovňové řízení, ale standardní CGI knihovna není vaší zálibou, vyzkoušejte tuto knihovnu (k nalezení na `http://ruby-web.org`).

Pokud chcete řešení postavené na šablonách, `Amrita` (`http://amrita.sourceforge.jp`) může být pro vás dobrým řešením. Také se podívejte na `Cerise`, což je webový aplikační server založený na `Amritě` (`http://cerise.rubyforge.org`). Pravděpodobně existují i další knihovny, takže pokud jste nenašli to, co jste hledali, zkuste pohledat na webu nebo se někoho zeptejte.

19.1.1 – Představení knihovny cgi.rb

Knihovna CGI je umístěna v souboru `cgi.rb` ve standardní distribuci Ruby. Většina její funkcionality je implementována kolem centrální třídy vhodně pojmenované jako `CGI`. Jednou z prvních věcí, kterou budete chtít udělat při používání této knihovny, je vytvoření instance `CGI`.

```
require "cgi"
cgi = CGI.new("html4")
```

Inicializátor pro třídu `CGI` přijímá jeden parametr, který specifikuje úroveň HTML, jež by měla být podporována metodami pro generování HTML kódu v `CGI` balíčku. Tyto metody zajišťují, aby programátor nemusel řešit vkládání množství HTML kódu do jinak čistého Ruby kódu:

```
cgi.out do
  cgi.html do
    cgi.body do
      cgi.h1 { "Hello Again, " } +
      cgi.b { cgi['name'] }
    end
  end
end
```

Zde jsme použili `CGI` knihovny k téměř přesné reprodukci funkcionality předchozího programu. Jak můžete sami vidět, třída `CGI` se stará o analýzu jakéhokoliv vstupu, přičemž výsledné hodnoty interně ukládá ve struktuře podobné haši. Takže – pokud jste specifikovali URL jako `some.program.cgi?age=4`, hodnota může být být zpřístupněna přes `cgi['age']`.

V předchozím kódu si povšimněte, že je ve skutečnosti použita pouze návratová hodnota bloku. HTML je vybudováno a uloženo postupně (tj. není okamžitě vypsáno). Jinak řečeno – spojování řetězců, které vidíme zde, je absolutně nezbytné. Bez toho by se objevil pouze poslední vyhodnocený řetězec.

Třída `CGI` dále poskytuje užitečné mechanismy pro práci s URL-kódovanými řetězci a pro citace HTML nebo XML kódu. URL kódování je proces, který spočívá v konverzi řetězců s nebezpečnými znaky do formátu, jenž je zobrazitelný v řetězci URL. Výsledkem jsou podivně vypadající "%" řetězce, jež vidíte v některých URL, zatímco prohlížíte web. Tyto řetězce jsou ve skutečnosti numerické ASCII kódy, které jsou reprezentovány hexadecimálně s prefixem "%".

```
require "cgi"
s = "This| is^(aT$test"
s2 = CGI.escape(s)           # "This%7C+is%5E%28aT%24test"
puts CGI.unescape(s2)       # Vytiskne "This| is^(aT$test"
```

Třída `CGI` může být dále použita k ošetření HTML nebo XML kódu, který by měl být zobrazen (tj. nikoliv vykonán) v prohlížeči. Například řetězec `<some_stuff>` nebude v prohlížeči zobrazen požadovaným způsobem. Pokud tedy máte potřebu doslovně zobrazovat HTML nebo XML kód

v prohlížeči (například při výuce HTML), třída CGI vám nabízí podporu pro konverzi speciálních znaků do příslušných entit, viz následující ukázka:

```
require "cgi"
some_text = "<B>This is how you make text bold</B>"
translated = CGI.escapeHTML(some_text)
# "<B>This is how you make text bold</B>"
puts CGI.unescapeHTML(translated)
# Vytiskne "<B>This is how you make text bold</B>"
```

19.1.2 – Zobrazení a zpracování formulářů

Nejběžnější způsob interakce s CGI programy je skrze HTML formuláře. HTML formuláře jsou vytvořeny pomocí specifických značek, které jsou následně přeloženy do vstupních widgetů v prohlížeči. Podrobnější diskuse k tomuto tématu je bohužel mimo rozsah této kapitoly, nicméně na webu (a také v různých knihách) lze nalézt dostatek potřebných informací.

Třída CGI nabízí metody pro generování všech prvků souvisejících s HTML formuláři. Následující fragment kódu ukazuje, jak zobrazit a zpracovat HTML formulář.

```
require "cgi"

def reverse_rambblings(ramblings)
  if ramblings[0] == nil then return " " end
  chunks = ramblings[0].split(/\s+/)
  chunks.reverse.join(" ")
end

cgi = CGI.new("html4")
cgi.out do
  cgi.html do
    cgi.body do
      cgi.h1 { "sdrawkcaB txeT" } +
      cgi.b { reverse_rambblings(cgi['ramblings']) } +
      cgi.form("action" => "/cgi-bin/rb/form.cgi") do
        cgi.textarea("ramblings") { cgi['ramblings'] } + cgi.submit
      end
    end
  end
end
```

Tento příklad zobrazuje textovou oblast (text area) a obsah, který bude rozdělen do slov a obrácen. Například – pokud do formuláře napíšete větu "This is test", po zpracování se objeví text "test is

This". Metoda `form` třídy `CGI` může přijímat parametr `method`, který určuje HTTP metodu (`GET`, `POST` atd.) použitou daným formulářem. Výchozí metoda je metoda `POST`.

Tento příklad předvedl pouze několik málo prvků, které lze použít v HTML stránce. Pro jejich kompletní seznam nahlédněte do libovolné referenční příručky o HTML.

19.1.3 – Práce s cookies

HTTP je bezstavový protokol. To znamená, že poté, co prohlížeč dokončí požadavek na webovou stránku, webový server nemá žádnou možnost, jak rozlišit jeho další požadavky od jakéhokoliv jiného prohlížeče na webu. A to je okamžik, kdy na scénu přichází HTTP cookies. Cookies totiž nabízí způsob (ačkoliv poněkud neohrabaný) pro udržování stavu mezi jednotlivými požadavky ze stejného prohlížeče.

Mechanismus fungování cookie je snadný. Webový server prostřednictvím HTTP záhlaví odpovědi posílá prohlížeči příkaz, aby někde uložil dvojici klíč/hodnota. Tato data mohou být uložena v paměti nebo na disku. Pro každý následující požadavek, který směřuje k doméně specifikované v této cookie, bude prohlížeč posílat data cookie v HTTP hlavičce požadavku.

Ačkoliv všechna tato cookies můžete vytvářet a číst ručně, určitě je vám jasné, že něco takového není potřeba. CGI knihovny Ruby totiž poskytují třídu `Cookie`, která umí pracovat s cookies.

```
require "cgi"
lastacc = CGI::Cookie.new("kabhi", "lastaccess=#{Time.now.to_s}")
cgi = CGI.new("html3")
if cgi.cookies.size < 1
  cgi.out("cookie" => lastacc) do
    "Hit refresh for a lovely cookie"
  end
else
  cgi.out("cookie" => lastacc) do
    cgi.html do
      "Hi, you were last here at: "+
      "#{cgi.cookies['kabhi'].join.split('=')[1]}"
    end
  end
end
```

Prostřednictvím tohoto fragmentu kódu je vytvořena cookie s názvem `kabhi`, která obsahuje klíč `lastaccess` nastavený na aktuální čas. Pokud má prohlížeč předchozí uloženou hodnotu pro tuto cookie, bude zobrazena. Cookies jsou přístupné jako proměnná instance na třídě `CGI` a uloženy jako `Hash`. Každá cookie může ukládat více dvojic klíč/hodnota, takže když přistoupíte k cookie jejím jménem, získáte pole.

19.1.4 – Práce s relacemi uživatele

Cookies jsou fajn, pokud chcete ukládat jednoduchá data a nevádí vám, že za jejich uchování je odpovědný prohlížeč. Ale v mnoha případech máte poněkud vyšší nároky na perzistenci dat. Například v případě, kdy máte k dispozici větší množství dat, která chcete trvale udržovat a jež nechcete posílat sem a tam s každým požadavkem. Co dělat v případě, kdy se jedná o nějaká citlivá data, která potřebujete asociovat s příslušnou relací a kdy nevěříte webovému prohlížeči?

Pro pokročilejší perzistenci ve webových aplikacích použijte třídu `CGI::Sessions`. Práce s touto třídou se velmi podobá práci s třídou `CGI::Cookie` v tom, že hodnoty jsou uloženy a získávány přes strukturu podobnou haši.

```
require "cgi"
require "cgi/session"

cgi = CGI.new("html4")
sess = CGI::Session.new( cgi, "session_key" => "a_test",
                        "prefix" => "rubysess.")
lastaccess = sess["lastaccess"].to_s
sess["lastaccess"] = Time.now
if cgi['bgcolor'][0] =~ /[a-z]/
  sess["bgcolor"] = cgi['bgcolor']
end

cgi.out do
  cgi.html do
    cgi.body ("bgcolor" => sess["bgcolor"]) do
      "The background of this page" +
      "changes based on the 'bgcolor'" +
      "each user has in session." +
      "Last access time: #{lastaccess}"
    end
  end
end
```

Přístup k `"/thatscript.cgi?bgcolor=red"` změní uživateli stránku na červenou pro každé následující načtení až do té doby, než bude v URL specifikována jiná barva pro `bgcolor`. Třída `CGI::Session` je instanciována s objektem `CGI` a sadou nastavení v `Hash`. Nepovinný parametr `session_key` specifikuje klíč, který bude prohlížečem použit při každém požadavku pro identifikaci sama sebe. Data relace jsou uložena v dočasných souborech pro každou relaci, přičemž parametr `prefix` specifikuje řetězec, kterým bude začínat název souboru, což učiní vaši relaci snadno identifikovatelnou v souborovém systému serveru.

Třída `CGI::Session` v současnosti stále postrádá spoustu užitečných rysů – například schopnost ukládat objekty (nikoliv pouze řetězce), ukládat relaci napříč několika servery atd. Dnes již ovšem existuje plugin `database_manager`, který může zjednodušit implementaci některých těchto rysů. Pokud děláte cokoliv vzrušujícího s `CGI::Session`, rozhodně se o podělte s ostatními.

19.2 – Používání FastCGI

Nejčastěji kritizovaný nedostatek CGI je ten, že pro každé nové volání vyžaduje vytvoření nového procesu. To má významný vliv na výkon. Neschopnost ponechat objekty v paměti mezi jednotlivými požadavky může mít negativní dopad i na návrh. Kombinace těchto potíží vedla k vytvoření FastCGI.

FastCGI v podstatě není nic víc než definice protokolu a softwarová implementace tohoto protokolu. FastCGI je obvykle implementováno ve formě pluginu webového serveru (například ve formě modul pro Apache), přičemž v rámci běžících procesů umožňuje zachytávat HTTP požadavků, které jsou přes socket přeměrovávány k trvale běžícímu procesu na pozadí. Tento přístup má pozitivní vliv na rychlost, zejména ve srovnání s tradičním spouštěním nového procesu pro obsluhu požadavku. Dále poskytuje programátorovi možnost ukládat věci do paměti (a při dalším požadavku je tam samozřejmě najít).

Servery pro FastCGI byly implementovány v mnoha programovacích jazycích včetně Ruby. Eli Green vytvořil modul, který je kompletně vytvořen v Ruby, jenž implementuje protokol FastCGI a který rapidně zjednodušuje vývoj FastCGI programů.

Aniž bychom zacházeli do podrobnějších detailů, ve výpisu 19.1 prezentujeme vzorovou aplikaci. Jak sami vidíte, tento kousek kódu poskytuje funkcionalitu shodnou s předchozím příkladem.

Výpis 19.1. Vzorová aplikace ve FastCGI.

```
require "fastcgi"
require "cgi"

last_time = ""
def get_ramblings(instream)
  # Nepěkně získá hodnotu prvního páru jméno/hodnota
  # CGI to pro nás může udělat.
  data = ""
  if instream != nil
    data = instream.split("&")[0].split("=")[1] || ""
  end
  return CGI.unescape(data)
end

def reverse_ramblings(ramblings)
```

```

    if ramblings == nil then return "" end
    chunks = ramblings.split(/\s+/)
    chunks.reverse.join(" ")
end
server = FastCGI::TCP.new('localhost', 9000)
begin
  server.each_request do |request|
    stuff = request.in.read
    out = request.out

    out << "Content-type: text/html\r\n\r\n"
    out << <<-EOF
    <html>
    <head><title>Text Backwardizer</title></head>
    <h1>sdrawkcaB txeT</h1>
    <i>You previously said: #{last_time}</i><BR>
    <b>#{reverse_ramblings(get_ramblings(stuff))}</b>
    <form method="POST" action="/fast/serv.rb">
    <textarea name="ramblings">
    </textarea>
    <input type="submit" name="submit"
    </form>
    </body></html>
    EOF
    last_time = get_ramblings(stuff)
    request.finish
  end
ensure
  server.close
end

```

První věci, které si na tomto kódu nepochybně povšimnete (pokud jste četli předchozí sekci), je několik detailů, které musíte ve FastCGI udělat ručně, a které byste tato nemuseli dělat při použití CGI. (Jedním takovým detailem je například zadrátovaný HTML kód.) Druhou věcí je metoda `get_ramblings`, která ručně analyzuje vstup a vrací pouze relevantní hodnoty. Tento kód mimochodem pracuje pouze s HTTP metodou POST, což je jedna z dalších nevýhod, na kterou narazíte při nepoužívání knihovny CGI.

Použití FastGCi samozřejmě s sebou nese i nějaké výhody. Ačkoliv v tomto příkladu nespouštíme žádné srovnávací testy, FastCGI je rychlejší než normální CGI. Například režii pro vytvoření nového procesu jsme ušetřili ve prospěch vytvoření spojení v místní síti na port 9000 (`FastCGI::TCP.new('localhost', 9000)`). A dále – proměnná `last_time` byla v tomto příkladu použita k uchování stavu v paměti mezi jednotlivými požadavky. Toto je v tradičním CGI nemožné.

Dále zde chci poukázat na skutečnost, že je možné tyto knihovny využívat pouze částečně. Pomocné funkce z `cgi.rb` mohou být použity samostatně (tj. bez použití této knihovny pro řízení aplikace). Například funkce `CGI.escapeHTML` může být použita izolovaně od zbytku knihovny. Toto by udělalo předchozí příklad o něco čitelnější.

19.3 – Ruby on Rails

Jedním z nejznámějších webových frameworků ve světě Ruby je nepochybně Ruby on Rails (nebo jednoduše Rails). Tento framework je dílem Davida Heinemeiera Hansona.

Rails využívá dynamických rysů Ruby. Má svou vlastní filozofii návrhu a umožňuje rychlý vývoj webových aplikací. Rails je nejenom velmi dobře známý, ale také dobře zdokumentovaný. V této knize se mu věnujeme pouze zběžně.

19.3.1 – Principy a technologie

Rails je vybudován na paradigmatu návrhového vzoru MVC (Model-View-Controller). Každá webová aplikace, která je postavena na Rails, je přirozeně rozdělena do modelů (jež modelují doménu problému), pohledů (které prezentují informaci uživateli a umožňují interakci) a ovladačů (jež zajišťují spojení mezi pohledy a modely).

Chování Rails jako frameworku je založeno na několika principech. Jeden princip je méně softwaru – nepište kód k tomu, aby navázal jednu věc na jinou, pokud tyto věci mohou být dohromady navázány automaticky. Další příbuzný princip je přednost konvence před konfigurací. Následováním určitých předdefinovaných stylů programování a pojmenovávání se konfigurace stává méně důležitou, čímž se tak přiblížíte k ideálnímu prostředí s "nulovou konfigurací".

Rails je dobrý v automatizaci úkolů, které vyžadují omezenou úroveň inteligence. Generuje kód vždy, když je to praktické, takže programátor není nucen vytvářet tyto kousky kódu ručně.

Protože webové aplikace se mnohdy neobejdou bez nějaké databáze v pozadí, Rails nabízí hladkou integraci databází. Existuje rovněž silná tendence, aby webové frameworky měly svůj oblíbený ORM (object-relational mapper) a Rails v tomto ohledu není výjimkou. Výchozí ORM pro Rails je ActiveRecord, což jsme si popsali již v kapitole 10.

Databáze jsou popsány v souboru `config/database.yml`, což je jeden z mála konfiguračních souborů, které budete potřebovat. Tento soubor, který je samozřejmě uložen ve formátu YAML, specifikuje tři různé databáze – jednu pro vývoj, jednu pro testování a jednu pro ostrý provoz. Ačkoliv tyto tři vyhrazené databáze mohou na první pohled vypadat nesmyslně, toto schéma je tím, co dává Rails jeho sílu.

Rails pro vás generuje prázdné modely a ovladače (controllers). Když tyto modely editujete, definujete vztah mezi databázovými tabulkami metodami, jako například `has_many` a `belongs_to` (abych uvedl aspoň dvě). Protože modely a tabulky vzájemně korespondují, tento kód rovněž definuje vztah mezi samotnými modely. Platnost dat může být bezproblémová s metodami jako va-