

# Ruby on Rails 实践

## Java 框架到底怎么了？

如果你作为一个 Java 程序员从事 j2ee 开发的话，你一定会使用到众多应用程序框架。没有任何一个语言会象 java 语言社区那样活跃，任何一种新的程序理念都会很快在网上出现相应的开源实现。对应最常用的网站开发模式 M V C，每一层都会有很多框架，Struts, Tapestry 属于控制器层 (C)，Velocity 框架属于视图层(V)，你使用的数据持久层可能是 Hibernate, iBatis, OJB, 或者是 JDO 的众多开源实现中的任何一个，比如 JPOX。但是你的选择太多，未必是件好事，并不是任何人都能采用正确的框架来做正确的事情。如果你的开发平台是 .net，那么你也许会避免这种情况，通常你只要安装一个 Visual Studio .net 作为开发工具，然后安装一个 MSDN 来查找资料就可以了。对于程序开发人员来说，这是非常两难的事情。我本人很喜欢 Java，无论是学习还是实践，它的确给我们提供了很多。但是为什么我觉得 .net 那样“一站式”解决方案在很多时候是正确的呢？

作为一个 Java 程序员，我觉得 Java 一些比较明显的问题，首先是 Java 太复杂，其次 Java 太面向程序员了，而不是面向用户。相对 C++ 来说，Java 已经很简单了。现在 Java 程序员数量如此多就说明了这点。但是正如有人曾经说过的那样，“在 linux 上，你很容易区分出谁是高手”，在 Java 领域中就不那么容易了。我就经常发现身边的同事还在犯很低级的概念性错误，他们甚至在无法准确地区分什么是接口，抽象类和 Servlet 的情况下仍然可以从事多年的 j2ee 开发。但为什么又说 Java 复杂呢，因为它完成一件事情，需要太多不同的技术来实现了。这样对于那些概念不很清楚的程序员来说，你如何能保证他们作出正确的选择呢？而众多框架中有没有多少提供“一站式”服务的。最近冒头的 Spring 框架提供的服务在众多的框架中算是最多的了，但是它又有个新问题，就是它还是太面向程序员了，而不是用户。为什么这么说呢？框架本来就是面向程序员的，这难道不对吗？Spring 虽然提供了众多选择(但是还是不够多，它本身没有 ORM)，但它没有提供简单的使用方式，所以我们只能说它是面向程序员的。绝大多数 java 框架都存在这个问题，就是学习曲线比较高。我觉得学习曲线的高低是区分一个框架是否是面向程序员还是用户的关键，我想这主要表现在框架的易用性上。其实框架最终的用户还是程序员，之所以用“用户”和“程序员”来区分，是因为一些面向“程序员”的框架比较难以使用，虽然提供了大量的基础设施和零件，但是还是要求程序员自己来组装。而面向“用户”的框架就简单一些，用户只要按照说明书来使用就可以了。为什么 Ruby on rails 会在 Java 社区引发轰动，我想原因就在于此，它提供了一个“一站式”面向用户的简单易用的框架，这是 java 框架所缺乏的。为什么 Ruby on rails 能做到这点，难道 java 本身做不到吗？事实是众多 Java 框架的设计者不这么做，可能是他们的思维已经限制在如何用模式设计一个好的框架上了，而没有在框架的易用性上做更多文章。使用过 Spring 的人就知道它的 xml 配置文件会渐渐的膨胀，虽然我们很容易将其分解为更多的小配置文件来解决这个问题。但是在使用 xml 配置文件上，它沿袭了 Java 编程的习惯性概念：“Java 是最好的编程语言，XML 是最好的描述数据的语言，两者的结合是最完美的。如果一个应用不使用 xml 来描述，那么它就不是好的 java 应用”。

但是 ruby on rails 就是在这点上和众多 java 框架区别开来,才达到了框架易用性上的一次突破。这个思想贯穿了 Rails 设计的始终:习惯约定优于配置。举个例子,通常我们写 java web 应用程序,都会按照 MVC 来给对应类做区分,我个人喜欢将 Controller 类放在 web 目录中,将 View 类放在 view 目录中,将 model 类放在 domain 目录中。但是不同的人有不同的设置,不同的命名,如何让框架知道这些不同的目录呢,java 框架的解决之道,只能是通过 xml 配置文件让它了解这些信息。而 rails 的解决方式就是:目录结构我来定义,你只要在我定义好的目录中放东西就可以了。这也就是为什么 rails 中很少有配置文件(但不是没有)的一个重要原因。虽然思想很简单,但是它带来的好处就是, Rails 的开发效率是 java 开发的 10 倍(这是 rails 的 fans 宣称的,不过我相信这点,相信看完这篇文章你也一定会的)。那么光这点就能让 rails 开发比采用 java 更快了吗?不完全是这样,因为这还得益于 rails 的另外一个设计理念:更少的代码。并不是任何语言都能那么宣称的, rails 实现这点完全得益于它的设计语言 Ruby。使用 Ruby 你的确能用很少的语言写很多的功能,这是其他语言所无法实现的。想要掌握 Rails,你一定要了解 Ruby。曾经有人说:Zope(著名的 python web 框架)是 python 的 killer 级应用,python 是 zope 的秘密武器。我想这句话用在描述 rails 和 ruby 的关系再合适不过了。那么我们还是来先了解一下 ruby 的一些基本概念吧。

## Ruby 简介:

在开始写这个教程之前,我原打算简单介绍一下 Rails 的使用就可以了,网上有相应的教程,只要照着翻译一遍就行。但是有两个原因促使我要先介绍一下 ruby:一是:作为一个 Ruby 语言的爱好者,我觉得有必要向大家介绍它的好处。它是学习 Rails 的关键所在。二是:不先学习一些 Ruby 的基本知识, Rails 一些代码你就会不明白原委。

Ruby 是由日本人松本行弘发明的一种面向对象的脚本语言。虽然很多语言都宣称自己是面向对象的,但是每种语言的解释都不一样,大多是以它们自己特有的方式来解释什么是面向对象,而实际情况中,这些面向对象语言又都采用了很多非面向对象的做法。以 Java 为例:如果你想对一个数字取绝对值,你会怎么做呢? java 的做法是

```
int c = Math.abs(-166);
```

也就是将一个数值传递给 Math 类的一个静态函数 abs 处理。为什么这么做?因为在 java 中, 数值是基本类型不是类。

而在 Ruby 中,任何事物都是类, 也就是说,数字 -166 就是对象,取绝对值这样的操作应该属于数字本身,所以"Ruby 方式"的做法就是:

```
c = -166.abs
```

这种做法是不是更直观一点。

接着我们来介绍一下 ruby 语言的一些简单特点。

如何定义函数

```
def hello(name)
  result = "hello, " + name
  return result
end
```

这就是一个最简单的函数定义。Ruby 以 `def` 开始一个函数定义，后跟着 函数名， 然后是参数，但是参数不必非要放在括号中， 你可以这样定义 `def hello name`，之所以要用括号是为了更清晰。在 `ruby` 中你可以用多种方式来完成同一件事情，这也是 `ruby` 的设计思想。

Hello 函数很简单， 将参数和 “hello, “字符串组合在一起，赋值给临时变量 `result`， 然后再返回 `result`。 你会注意到 `result` 变量并没有申明，因为在 `ruby` 中无需申明，变量在赋值中自动使用。 另外 `ruby` 中不需要使用 `;` 来结束每个语言，只要保持每条语句在单独一行就可以。 `Ruby` 也不包含 `{}` 来定义块结构， `if`，`when` 函数都以一个 `end` 关键字来结束。缩进不是语法（`python` 程序员要失望了）。第一次使用 `ruby` 我也有点不习惯，因为我认为 `python` 语言的缩进语法可以使得阅读程序代码更容易一些。但是学习了一段时间后，我发现缩进语法并不是那么重要，`Ruby` 本身带来的帮助远比这个更重要。而且代码的易读性也不是由缩进来改善的。此外上面的代码还可以更简化。

```
def hello (name)
  " hello, #{name}"
end
```

在 `ruby` 中，函数的最后一条语句如果是表达式，那么它就作为返回值。在上面的代码中，用到了另外一个概念，就是表达式插入，字符串可以用单引号和双引号来括起来。但是两者还是有一点区别。区别在于处理时间上的不同。如果使用单引号，那么处理的时间很短。如果使用双引号，那么你可以插入变量，表达式，还有就是转意字符的替换，最常见的是 `\n`，`\t` 等。 以上面的代码为例 字符串中插入了变量 `name`，字符串中插入变量以 `#` 开始，变量放在`{}`中。 但是特殊变量可以不用 `{}`。那么什么是特殊变量呢。这个也是 `ruby` 特别的地方。在 `Ruby` 中，全局变量以 `$`开头，静态变量，也就是类变量以 `@@` 开头，实例变量以 `@` 开头。如果在字符串中引用 实例变量，你可以这么写。

```
def hello
  " hello, #@name"
end
```

关于变量和基本结构我们就简单介绍这些，下面我们要介绍一个重要的 `Ruby` 概念。就是 `iterator` (迭代器)。 这个概念遍布在很多 `ruby` 代码中，也是最常见的 `ruby way`。正是因为 `iterator`，在 `ruby` 程序中，你很少会看到循环的使用。 举个最简单的例子，如何打印从 1 到 6 的结果。

在 `java` 等语言中，我们的实现方法是：

```
for(int i = 1; i<6;i++) {
```

```
    System.out.println(i);
}
```

而在 ruby 中，一切皆为对象，数字本身就是对象(我们在前面已经介绍过了)，数字本身有它自己的迭代器 `times` 。那么让我们看看它的实现。

```
6.times {|i| p i }
```

是不是很简单。

迭代器的概念很有趣，它是如何在 ruby 中实现的呢。每个函数可以不光带有参数，还可以在参数后面带一个代码块。代码块在 ruby 中是以 `{ }` 和 `do end` 来包围起来的。通常如果是单行使用 `{ }` ，如果是多行这使用 `do end` 。这并非强制，只是习惯性用法。

代码块和参数并不相同，它和函数的执行可以说是并行的，确切的说是接替进行。如果在函数内部如何在运行过程中碰到了 `yield` ，它就会将函数执行过程交给函数附带的代码块来执行，代码块执行完了，执行流程转到函数内部继续运行。似乎很难理解，以下面这段代码来理解可以会容易一些。

```
def callBlock
  yield
  yield
end
```

```
callBlock { puts "In the block" }
```

输出:

```
In the block
In the block
```

`CallBlock` 函数中只有两个 `yield` 语句，它在执行过程中将执行权交给 函数附带的 `block` 运行，`block` 输出字符串 "In the block" ,这样最终结果就是输出两行 "In the block" 。 在函数内部使用 `yield` 还可以带任意多参数。比如

```
yield a, b, c
```

这三个参数对应代码块中的三个参数 `{|x,y,z| p x y z }`

好了， Ruby 就介绍到这里。下面正式进入正题，介绍一下 Rails 的使用。

## 什么是 rails ?

在写这个教程的最初，我基本上是在翻译网上的教程。但是 rails 中包含了太多的程序和概

念，如果只是简单的介绍，你虽然会很快学会使用 rails，但是对于它的一些概念了解不深。在这里我希望更多地介绍一些 rails 所包含的工具，比如 rubygems, webrick 等等。

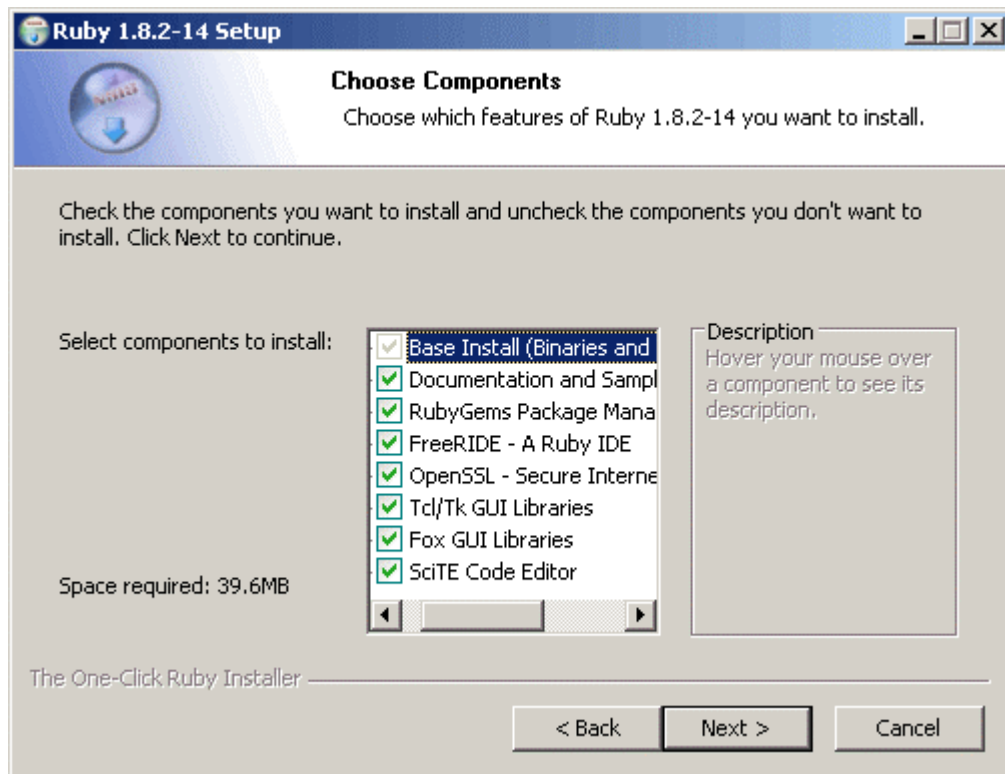
整个教程是基于ruby的 windows 发行版。Ruby 作为跨平台的脚本语言，几乎在所有的平台上都有相应的实现，很多 linux 发行版本都带有了 ruby安装文件，而 windows 平台则需要从 [www.rubyforge.net](http://www.rubyforge.net)上下载一个 one-click installer 安装程序。由于 ruby one-click installer 带有了 rubygems package manager包，所以就省去了我们很多麻烦。我们只需要一条命令就可以从网上下载 rails 了（当然安装 rails 你需要保证你的机器连上 internet）。

简单介绍一下 rubygems. Rubygems 是最近渐渐在 ruby 社区流行起来的包管理工具。在以前，如果你要下载一个 ruby 扩展或者应用程序的话，你需要下载相应的 zip 包，然后解压缩，将应用或者扩展安装到 ruby 对应的目录中。但是有了新的 rubygems. 所有这些麻烦都没有了，你只需要一条命令就可以从远程服务器上下载相应的包，如果相应的应用包含其他扩展，rubygems 也会提示你从远程安装所依赖的扩展。安装后 rubygems 会运行相应的程序生成 rdoc 帮助文档(类似于 javadoc)。当然你也可以将软件包下载到本地运行 rubygems 的本地安装命令。有了 rubygems 包管理器，ruby 应用的安装将变得前所未有的容易。统一化的管理带来的好处就是简单。现在 ruby 社区的应用都在朝着写 gems 的方向发展，很多以前的 RAA 都转化为 gems 了，而 rubygems 也将成为 ruby 事实上的包管理器标准了。记得以前看电影《指环王》的时候记得一句话，那就是 “one ring to rule them all”，有了 rubygems，你可以说是 “one gem to rule them all” 了。

## 安装 ruby

首先我们看看，如何安装 windows 版本的 ruby。

最新版本的 [One-Click Ruby Installer for Windows](http://www.rubyforge.net) 可以从 <http://rubyinstaller.rubyforge.org/>上下载。安装界面如下：



安装没有什么特殊之处，只要选择好安装目录，接受默认的设置连续点击 next 就可以完成。

## 使用 rubygems 安装 rails

现在我们来看看如何用 rubygems 来安装 rails. 刚才我们说过了，因为 [One-Click Ruby Installer for Windows](#) 已经包含了 rubygems 包管理器，所以我们不在下载安装 rubygems，直接利用它来下载 rails.

gem 命令包含很多子命令和相应的选项，比如

gem -h/--help – 显示 rubygem 的帮助

gem -v/--version – 显示 rubygem 的版本号

gem list --local – 用子命令 list 列出本地安装的 gems

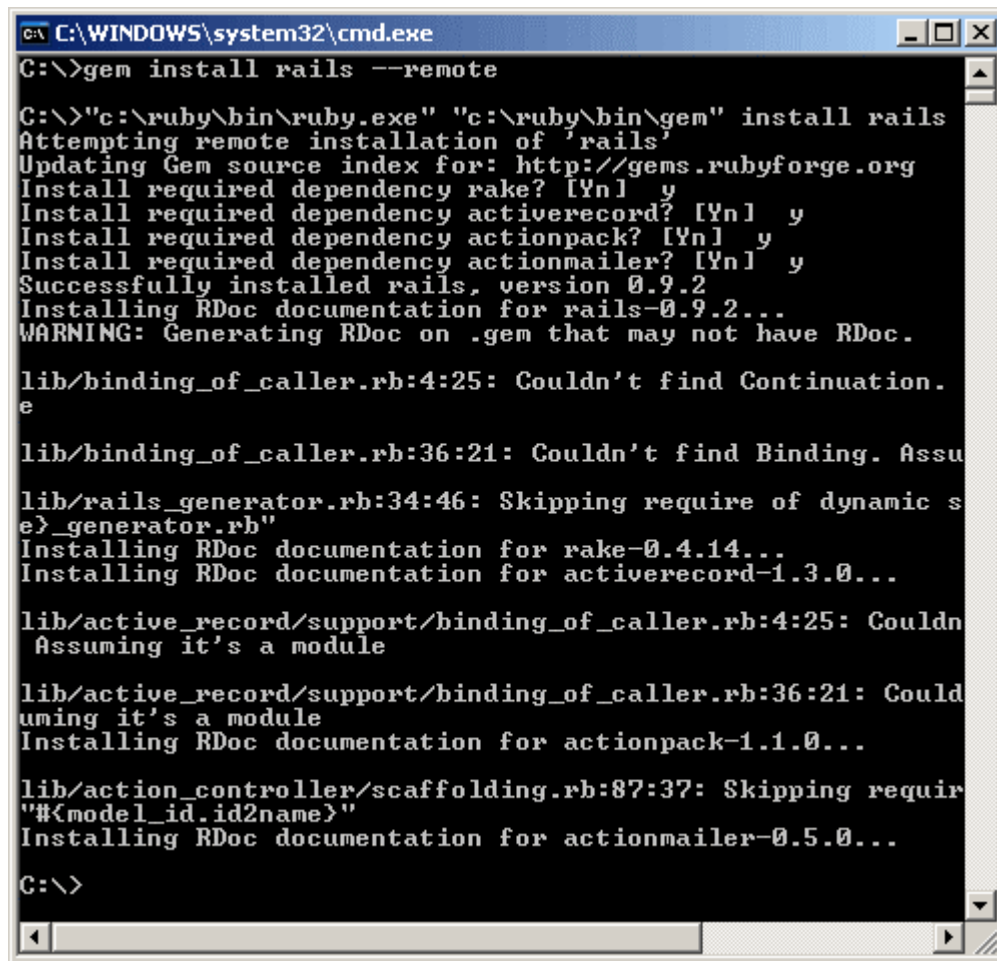
如图显示本机安装的 gems

打开 DOS 命令行窗口，运行如下命令：

```
gem install rails --remote
```

这条命令显式地从远程服务器上安装 rails，你也可以运行 `gem install rails` 来安装，gem 会判断本地是否安装了 rails，由此来决定是否从远程安装 rails.

如图显示



```
C:\WINDOWS\system32\cmd.exe
C:\>gem install rails --remote

C:\>"c:\ruby\bin\ruby.exe" "c:\ruby\bin\gem" install rails
Attempting remote installation of 'rails'
Updating Gem source index for: http://gems.rubyforge.org
Install required dependency rake? [Yn] y
Install required dependency activerecord? [Yn] y
Install required dependency actionpack? [Yn] y
Install required dependency actionmailer? [Yn] y
Successfully installed rails, version 0.9.2
Installing RDoc documentation for rails-0.9.2...
WARNING: Generating RDoc on .gem that may not have RDoc.

lib/binding_of_caller.rb:4:25: Couldn't find Continuation.
e
lib/binding_of_caller.rb:36:21: Couldn't find Binding. Assu
lib/rails_generator.rb:34:46: Skipping require of dynamic s
e>_generator.rb"
Installing RDoc documentation for rake-0.4.14...
Installing RDoc documentation for activerecord-1.3.0...

lib/active_record/support/binding_of_caller.rb:4:25: Couldn
Assuming it's a module
lib/active_record/support/binding_of_caller.rb:36:21: Could
uming it's a module
Installing RDoc documentation for actionpack-1.1.0...

lib/action_controller/scaffolding.rb:87:37: Skipping requir
"#<model_id.id2name>"
Installing RDoc documentation for actionmailer-0.5.0...

C:\>
```

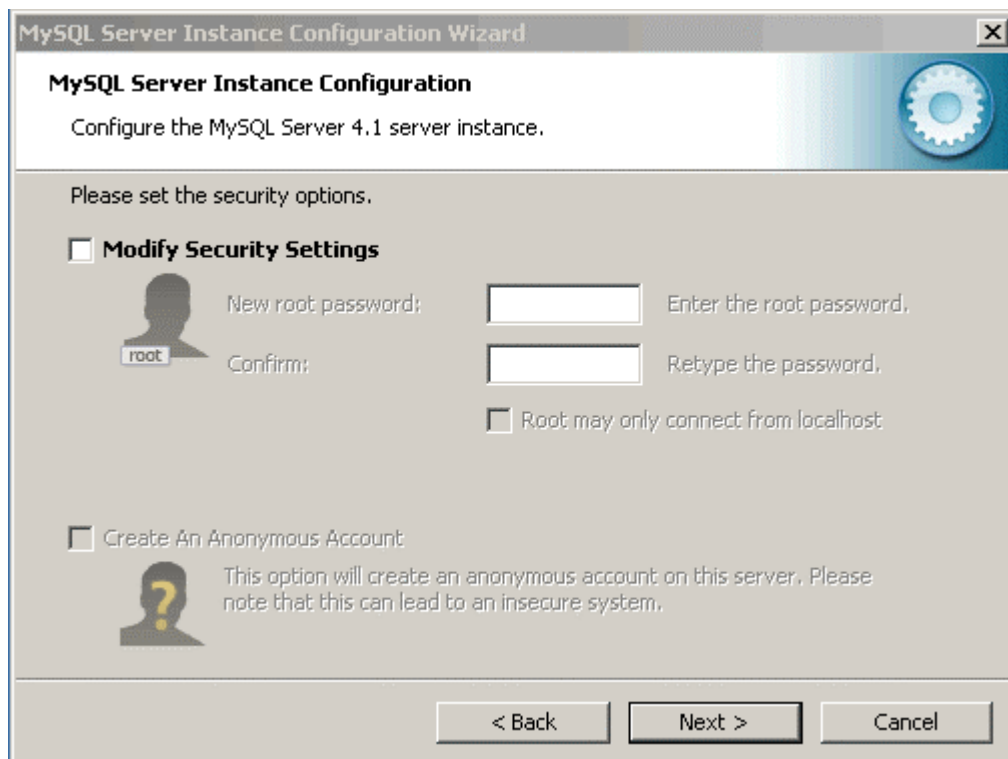
你安装的时间不同，显示的信息略微有所不同。不同之处在于安装 rails 所依赖的扩展包有所不同。一般来说，安装 rails 需要的包有 rake (这是类似 java ant 的 build 程序)，activerecord(rails 依赖的 orm 工具，也就是 MVC 中的 Model 模块)，actionpack (对应的 controller 模块)。你可能会问那么相应的 view 模块怎么没有安装，因为 view 模块已经在默认的 ruby 安装中包含了，就是 eruby, 它的作用就是完成对模板文件 .rhtml 的解释。

**安装 mysql 数据库和管理中心。**



从 [mysql.com](http://dev.mysql.com/downloads/mysql/4.1.html) 上下载 windows 版本的 mysql 服务器, 本文章以 mysql 4.1 版本为例子(下载地址为 <http://dev.mysql.com/downloads/mysql/4.1.html>)。

默认设置安装 mysql, 在运行到结尾处安装程序会启动安装向导, 在安装向导中需要设置安全信息(用户名和口令)。由于 mysql 从 4.1.7 版本后采用了新的认证算法, 这 and 老版本的 rails 不兼容, 所以网上教程不要求在这里设置用户名和口令。但是新版本的 rails 已经解决了这个问题。本教程暂时不设置用户名和口令。



oreily 网站上发布的 rails 教程中是以 [MySQL-Front](#) 为客户端程序的。由于这是个商业程序虽然使用比较方便, 但是会遇到过期的问题。所以本教程采用了免费的 MySQL Control Center



客户端程序(mysqlcc-0.9.4-win32)，本程序可以从 [mysql.com](http://mysql.com) 上下载。安装界面如下：

## 编写 rails 应用

总算到了这激动人心的一步了。

**注明：**以下示例大量参考了 [ONLamp.com](http://www.onlamp.com/) (<http://www.onlamp.com/>) 上的 Ruby on rails 教程 “Rolling with Ruby on Rails” 原网址：<http://www.onlamp.com/lpt/a/5546>

## 任务简介：

我们这里要用 rails 实现的是一个简单的网上书籍管理程序 mybook 。

我们所要完成的任务如下：

- 1：显示本人所收藏的所有书籍
- 2：新增和编辑相应的书籍
- 3：将书籍指定不同的类别

在这里你会看到 rails 框架所展示的前所未有的易用性，整个程序你只需要真正输入一行代码，就可以完成数据库的 CRUD 操作（增删改查），这不仅仅是完成数据库级别的操作，连所有相应的默认视图也都一并完成。Rails 是我所用过的各种 web 应用框架中最最简单的一个，它的开发效率的确远远超过了任何一个 java 框架。

## 创建空的 rails 网页应用程序骨架

在 DOS 命令行窗口运行 rails mybook （本程序的工作目录是 D:\railsdoc>）

```
D:\railsdoc>rails mybook
create
create  app/apis
create  app/controllers
create  app/helpers
create  app/models
create  app/views/layouts
create  config/environments
create  components
```

```
create db
create doc
create lib
create log
create public/images
create public/javascripts
create public/stylesheets
create script
create test/fixtures
create test/functional
create test/mocks/development
create test/mocks/test
create test/unit
create vendor
create Rakefile
create README
create CHANGELOG
create app/controllers/application.rb
create app/helpers/application_helper.rb
create test/test_helper.rb
create config/database.yml
create config/routes.rb
create public/.htaccess
create config/environment.rb
create config/environments/production.rb
create config/environments/development.rb
create config/environments/test.rb
create script/console
create script/console_sandbox
create script/destroy
create script/generate
create script/server
create script/runner
create script/benchmark
create script/profiler
create script/breakpointer
create public/dispatch.rb
create public/dispatch.cgi
create public/dispatch.fcgi
create public/404.html
create public/500.html
create public/index.html
create public/favicon.ico
create public/javascripts/prototype.js
```

```
create doc/README_FOR_APP
create log/server.log
create log/production.log
create log/development.log
create log/test.log
```

D:\railsdoc>

你可以看到， rails 已经为你生成了 mybook 应用程序的完整目录结构。

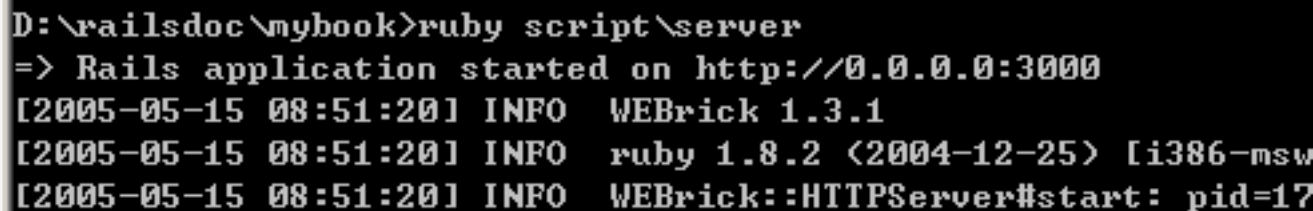
Rails 不仅仅是一个运行时网络应用框架，它本身包含了丰富的脚本来帮你完成程序基本骨架的工作。

运行 rails mybook 后， rails 将生成 mybook 应用程序的默认目录结构和初始化文件。Rails 开发的方式是： 由 rails 来生成默认的应用程序骨架，你所做的工作就是在默认目录结构中编辑文件增加应用程序逻辑就可以了。

紧接着我们来测试一下空的应用程序，看看 rails 默认给我们完成了什么工作。

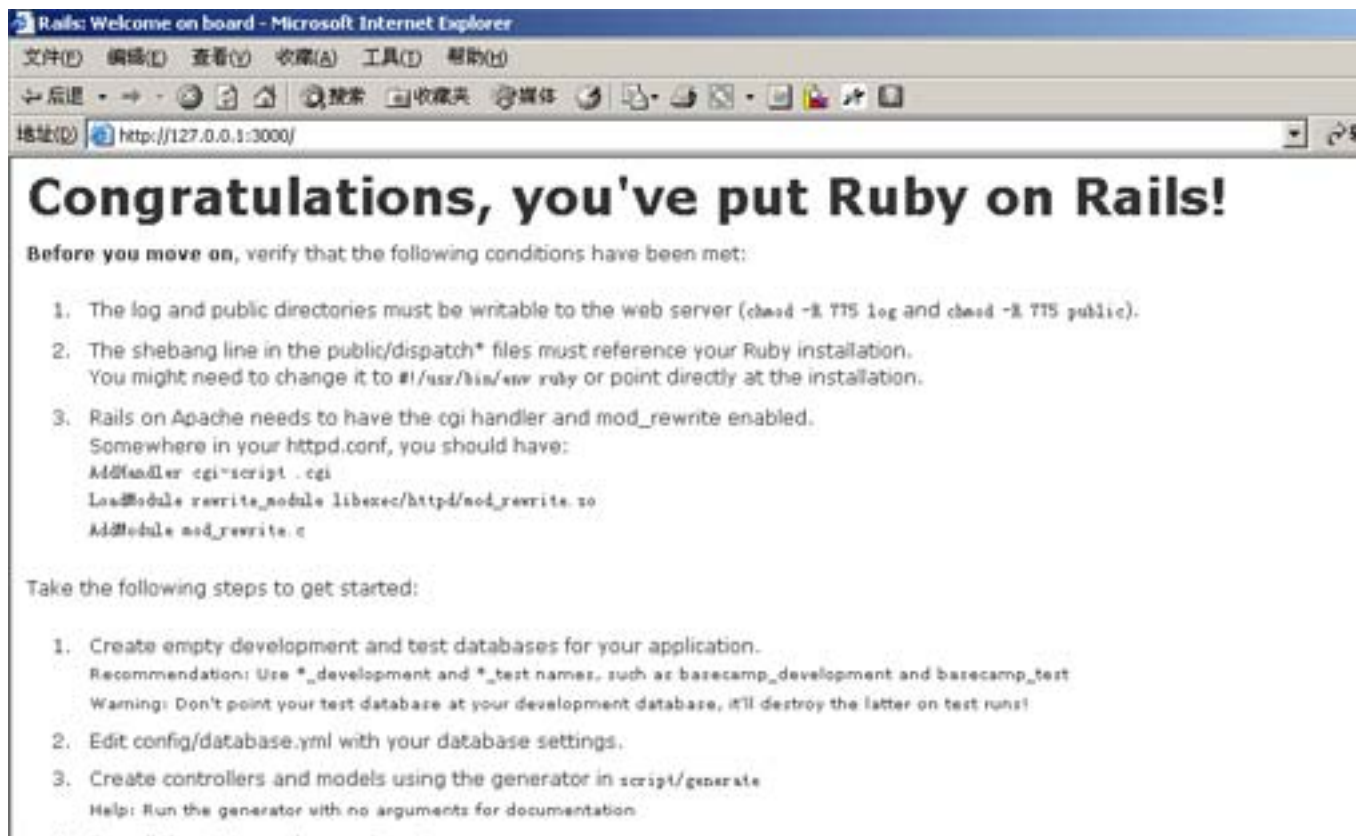
本程序的工作目录是 D:\railsdoc>，上面 rails 已经给我们生成了mybook 目录，进入 mybook 目录。运行 ruby script\server 。 这条命令是运行 script 目录下的 server 命令来启动 webrick 服务器。

启动 webrick 服务器如下：

A terminal window showing the command 'D:\railsdoc\mybook>ruby script\server' and its output. The output indicates the Rails application started on http://0.0.0.0:3000, followed by three INFO logs from WEBrick 1.3.1, ruby 1.8.2, and WEBrick::HTTPServer#start: pid=17.

```
D:\railsdoc\mybook>ruby script\server
=> Rails application started on http://0.0.0.0:3000
[2005-05-15 08:51:20] INFO WEBrick 1.3.1
[2005-05-15 08:51:20] INFO ruby 1.8.2 (2004-12-25) [i386-msw
[2005-05-15 08:51:20] INFO WEBrick::HTTPServer#start: pid=17
```

打开浏览器，输入网址<http://127.0.0.1:3000/> 你会看到类似的网页。



暂时不要关闭这个窗口。

Webbrick 简介:

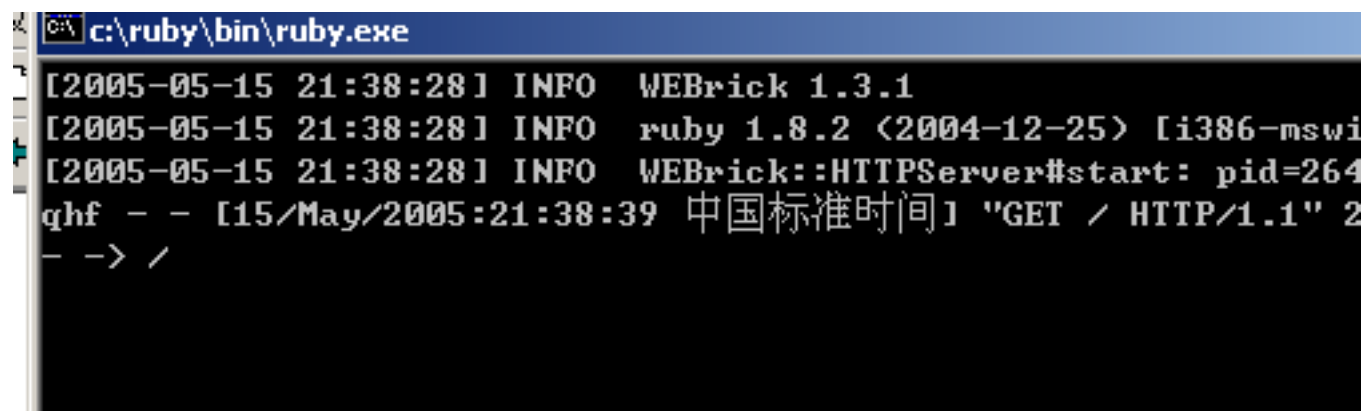
Ruby 默认已经包含了丰富的软件，其中 webrick http 服务器就是其中之一。这个程序包被用作 rubygem 的默认文档服务器。这是一个非常简单的纯 ruby 编写的服务器。如果你掌握了 webrick 你可以用几行代码来运行一个 web 服务器。

打开编辑器，编写以下脚本

```
require "webrick"

httpd = WEBrick::HTTPServer.new(
  :DocumentRoot => Dir::pwd + "/ruby",
  :Port         => 80
)
trap(:INT) { httpd.shutdown }
httpd.start
```

然后保存到 `c:\server.rb` 下，双击 `server.rb`，一个最简单的 `http server` 就运行了，它将 `c:\ruby` 目录作为服务器文档根目录。



```
C:\ruby\bin\ruby.exe
[2005-05-15 21:38:28] INFO WEBrick 1.3.1
[2005-05-15 21:38:28] INFO ruby 1.8.2 (2004-12-25) [i386-mswin32]
[2005-05-15 21:38:28] INFO WEBrick::HTTPServer#start: pid=264
qhf - - [15/May/2005:21:38:39 中国标准时间] "GET / HTTP/1.1" 200
- -> /
```

在浏览器窗口打开 <http://127.0.0.1/>

你将会看到 `c:/ruby` 目录的内容，我们的 8 行 `ruby` 代码就生成了一个简单 `http` 服务器。你不光可以定制文档根目录，你还可以象编写 `java servlet` 那样，为 `webrick` 编写 `ruby servlet` 代码。具体详细信息参看 [www.webrick.org](http://www.webrick.org)

Index of / - Microsoft Internet Explorer

文件(F) 编辑(E) 查看(V) 收藏(A) 工具(T) 帮助(H)

后退 后退 搜索 收藏夹 媒体

地址(D) <http://127.0.0.1/>

# Index of /

<u>Name</u>	<u>Last modified</u>	<u>Size</u>
<a href="#">Parent Directory</a>	1980/01/01 00:00	-
<a href="#">ProgrammingRuby.chm</a>	2005/04/18 00:42	764836
<a href="#">ReleaseNotes.txt</a>	2005/04/18 00:46	5579
<a href="#">bin/</a>	2005/05/15 08:06	-
<a href="#">doc/</a>	2005/05/15 08:06	-
<a href="#">freeride/</a>	2005/05/15 08:06	-
<a href="#">include/</a>	2005/05/15 08:06	-
<a href="#">installer/</a>	2005/05/15 08:06	-
<a href="#">lib/</a>	2005/05/15 08:06	-
<a href="#">man/</a>	2005/05/15 08:06	-
<a href="#">readme.txt</a>	2004/07/31 13:19	4432
<a href="#">samples/</a>	2005/05/15 08:06	-
<a href="#">scite/</a>	2005/05/15 08:06	-
<a href="#">share/</a>	2005/05/15 08:05	-
<a href="#">uninst.exe</a>	2005/05/15 08:06	56572

---

*WEBrick/1.3.1 (Ruby/1.8.2/2004-12-25)*  
*at 127.0.0.1:80*

Rails 开发工作的大部分集中在创建和编辑 apps 目录下的文件。让我们首先来了解一下这个目录下的结构。

Apps 目录下包含四个主要目录，它们分别是  
 Controllers, models, views, helpers

1: Controllers 目录存放 rails 应用中相应的 controller 类, controller 类处理来自用户的 web 请求。

2: views 目录存放相应的模板文件, 模板文件填充数据后, 转换为 html 格式传递给用户的浏览器。

3: models 目录存放数据模型类, 模型类封装了数据库中的数据。很多框架在数据模型层都做的比较复杂, 用过 rails 后, 你会发现它非常容易使用。

4: helpers 目录存放了简化 Controllers, models, views 使用的帮助类。

## 编辑 controller 类

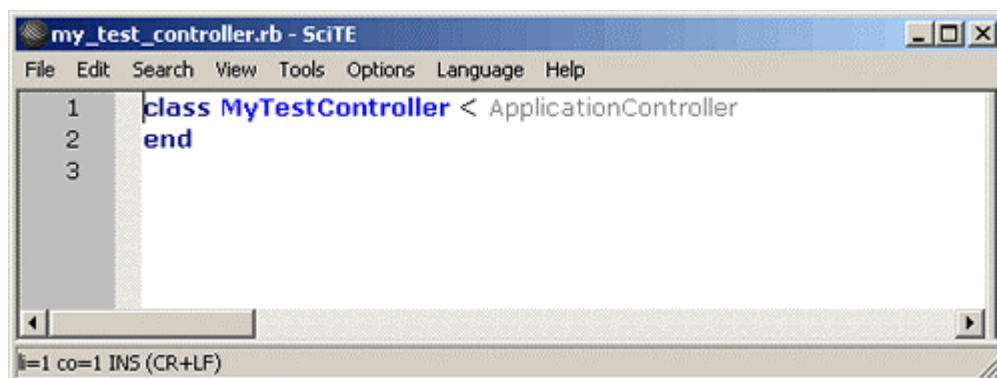
开始使用 Rails 之前, 我们需要了解一下 rails 是如何将 controller 类中的方法和 url 映射到一起的, 这对于理解 rails 工作原理很重要。

rails 在生成的 skeleton 框架中包含了一个 scripts 目录, 这个目录中包含了很多脚本程序, 它们可以帮助我们来简化很多工作, 我们已经使用过了 script\server 这个脚本, 它是用来启动 webrick 服务器的。现在我们看看如何生成默认的 controller 文件

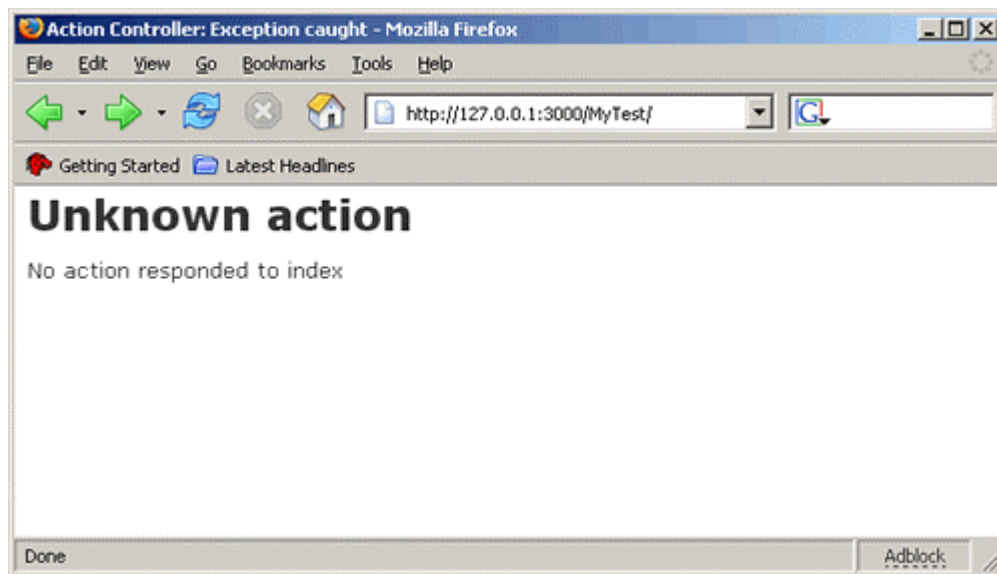
打开新的 DOS 命令行, 运行 `ruby script\generate controller MyTest`

D:\railsdoc\mybook\app\controllers 目录中将生成一个包含 MyTestController 类定义骨架的名为 `my_test_controller.rb` 的文件

鼠标右键点击此文件, 选择“edit” 打开编辑此文件。



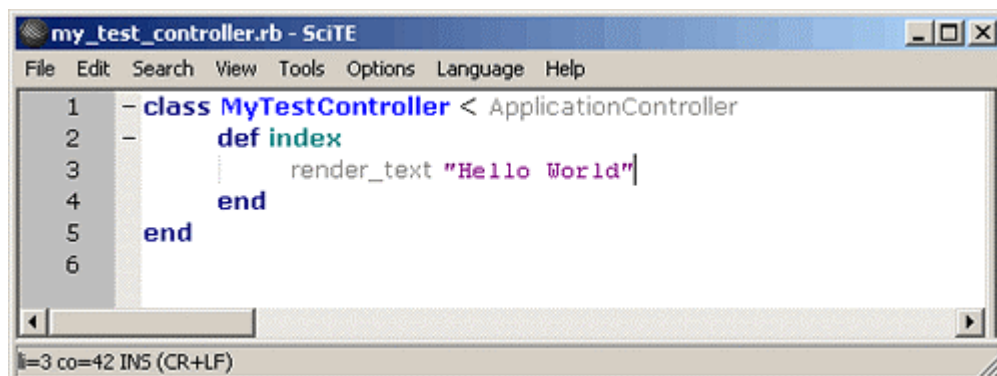
在浏览器中访问 `http://127.0.0.1:3000/My_Test/`,



浏览器响应没有找到对应的 `index` 方法，不要担心，这不是什么大问题，因为默认生成的控制器类，不包含任何方法。不过这里我们了解到，如果不输入任何 `url` 连接的话，默认 rails 需要在控制器类中找一个 `index` 的方法。这很容易理解，类似于 `apache` 服务器中目录下面的 `index.html` 文件一样。

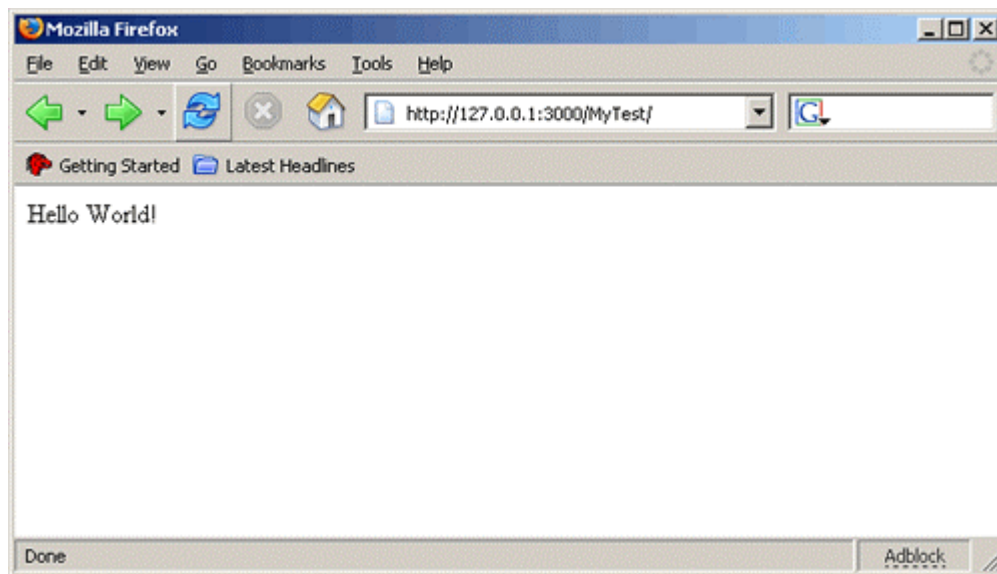
我们现在编辑这个 `MyTestController` 类。

如图：



重新刷新刚才的页面 `http://127.0.0.1:3000/My_Test/`，你会看到如下结果





继续编辑这个控制器类，

A screenshot of the SciTE text editor window. The title bar reads 'my\_test\_controller.rb - SciTE'. The menu bar includes 'File', 'Edit', 'Search', 'View', 'Tools', 'Options', 'Language', and 'Help'. The editor contains the following Ruby code:

```
- class MyTestController < ApplicationController
-   def index
-       render_text "hello world"
-   end
-
-   def hello
-       render_text "hello rails"
-   end
end
```

保存文件后，在浏览器中运行 [http://127.0.0.1:3000/My\\_Test/hello](http://127.0.0.1:3000/My_Test/hello)



这下你了解到 rails 是如何将 url 和控制器中的函数做简单映射的了吧。

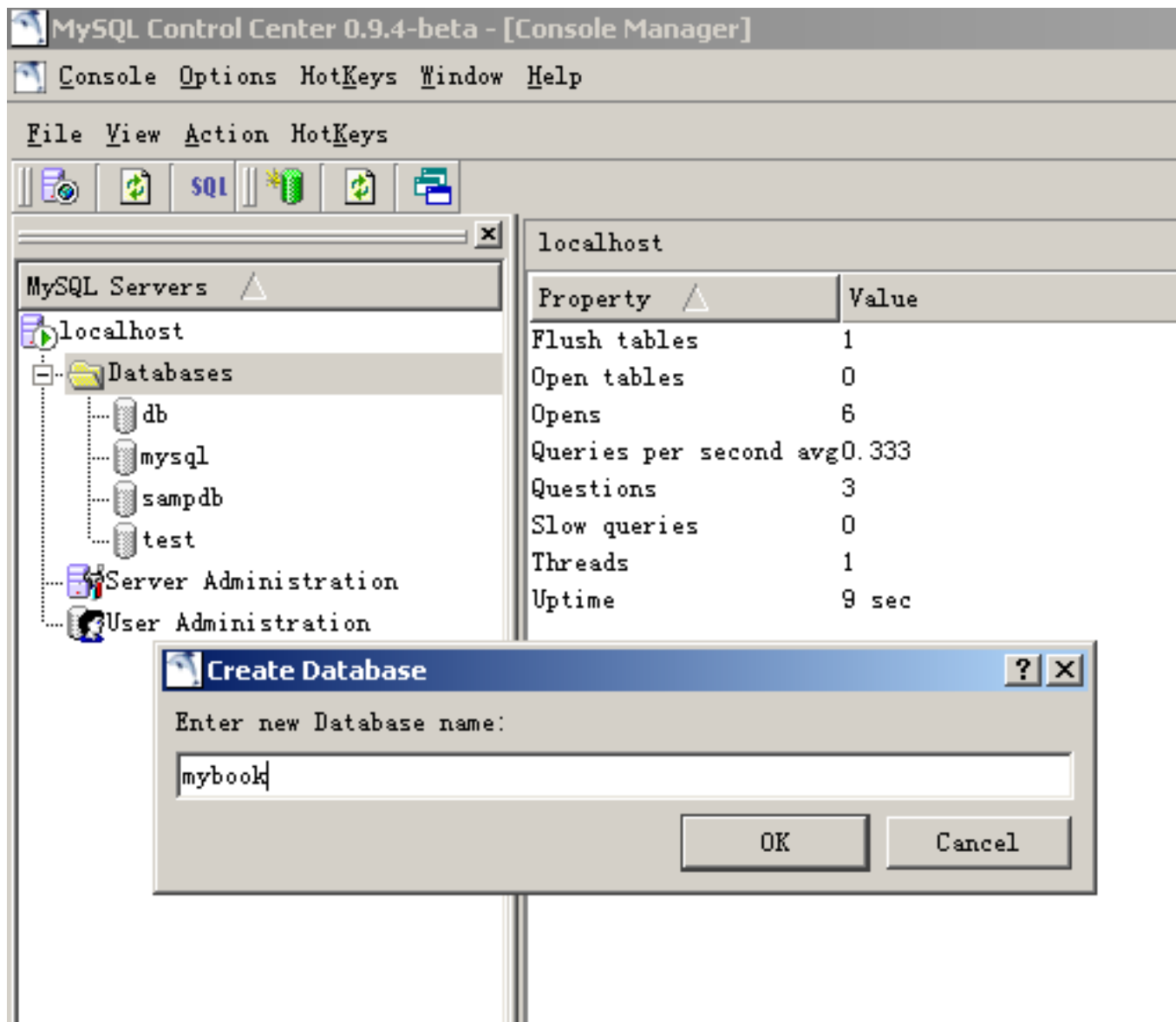
这里需要做一些解释，在运行 `ruby script\generate controller MyTest` 的时候， rails 做了一些默认的假设。MyTest 对应的 controller 文件是 `my_test_controller.rb`，文件中控制器类名是 `MyTestController`，而 url 映射是 `/My_Test/`。

这些默认规定的好处是：我不需要编辑复杂的配置文件，只要对相应的文件做编辑就可以了。我们只需要关注程序逻辑，而不是其它什么，这就是 rails 设计原则中的“习惯约定优于配置”。

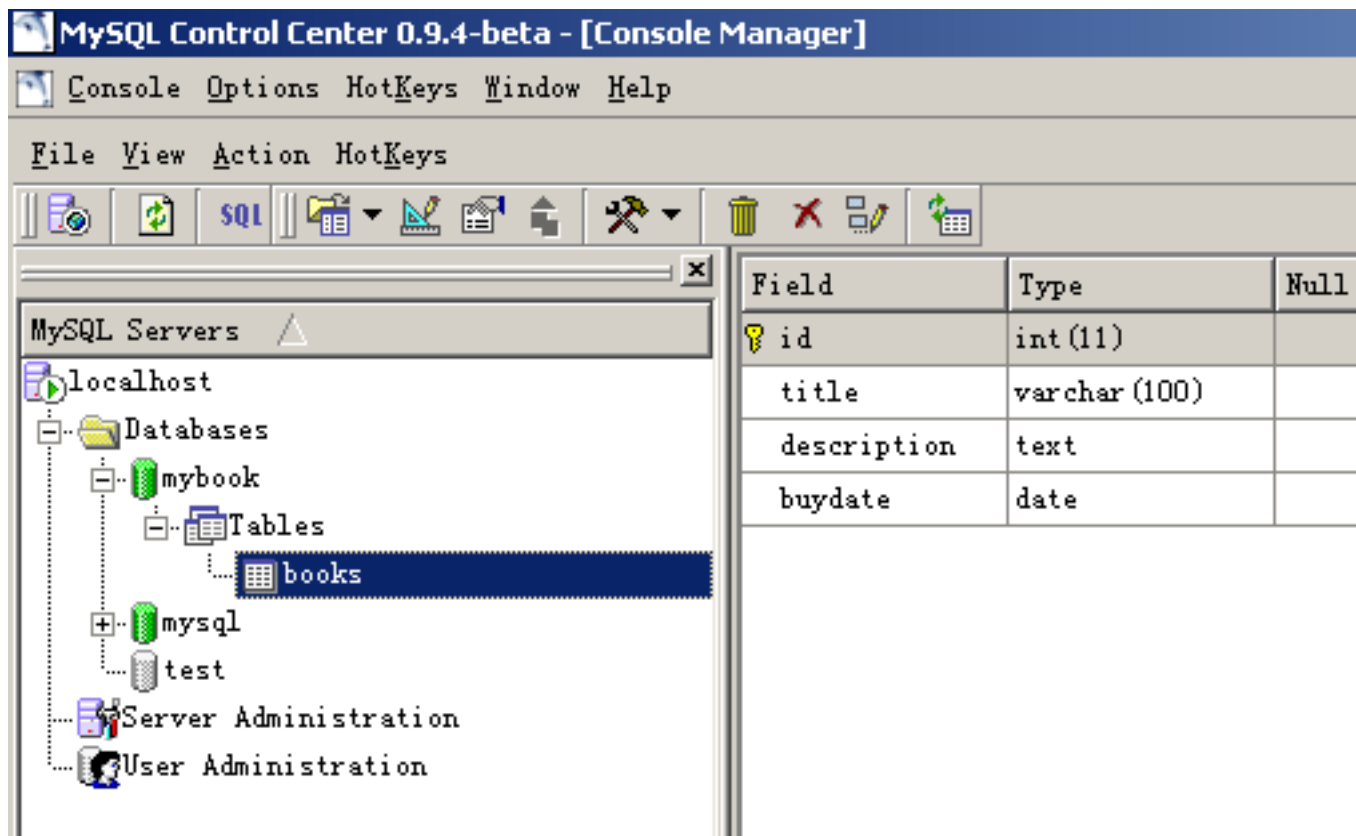
另外我们也感觉到 ruby 动态语言的好处，这点是 java 所无法办到的。我们在编辑完 controller 后，重新刷新页面，就可以使程序生效。对于编译型的 java 语言来说，不重新编译，重新启动服务器是无法做到这点的。Rails 开发的易用性可见一斑了，要知道启动和编译都是非常耗费时间的，尤其对于那些 EJB 服务器来说更是如此。

## 创建 mybook 数据库

1：打开 MySQL Control Center，新建 mybook 数据库



2： 新建数据库表 books



表字段的定义如图所示

id int(11) 主键  
title varchar(100)  
description text  
buydate date

3: 告诉 rails 如何找到数据库信息，打开 D:\railsdoc\mybook\config 目录，编辑 database.yml 配置文件。 ruby 程序喜欢用 yaml 作为配置文件，这种做法就象是 java 程序喜欢用 xml 作配置文件一样，是传统习惯了。

## 新增内容:

YAML 简介:

很多初次接触 Ruby 的人可能会对它采用 yaml 文件作为配置文件感到奇怪。Yaml 库已经内置在 ruby 发行版本中，做为很 “Ruby way” 的方式用在各种读取配置的操作中。希望了解 Yaml 格式的人，可以参看一下 <http://www.yaml.org>

这里翻译一下该网站对 yaml 的解释:

- YAML 适合人们阅读
- YAML 适合与脚本语言交流

- YAML 使用宿主语言的内部数据结构
- YAML 拥有一致性的信息模型
- YAML 使基于流式数据的处理成为可能
- YAML 富于表达和可扩展
- YAML 容易实现

yaml 做为一个可移植的对象序列化方案，可以在不同的 Ruby 进程中将对象以普通文本传递，此外也适合 ruby 与那些支持 yaml 的语言之间交换数据用。

**如何将 ruby 对象保存到 yaml 文件中.**

参考文档: <http://www.ruby-doc.org/core/classes/YAML.html>

```
require 'yaml'

tree = { :name => 'ruby',
         :uses => ['script', 'web', 'testing', 'etc']
       }

File.open("tree.yaml", "w") {|f| YAML.dump(tree, f)}
```

如何在 ruby 程序中读取 yaml 文件。

```
require 'yaml'

tree = YAML.load(File.open("tree.yaml"))
tree[:uses][1]
```

输出结果: "web"

yaml 格式的一个大用途就是用在存储配置信息，因为它是纯文本格式的，非常适合阅读，同时又很适合 Ruby 等脚本程序将其读取到对象中。

下面就是个例子:

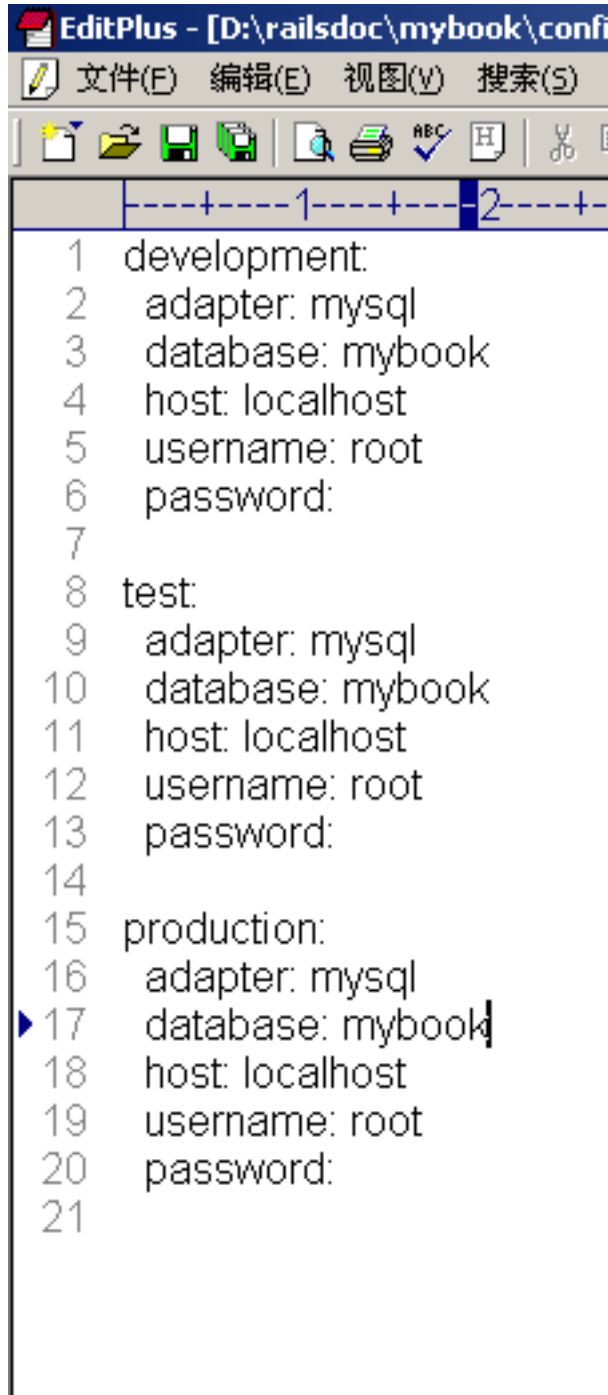
conf/config.yaml 文件内容如下:

```
host: 127.0.0.1
prefs:
  username: mulder
  password: trustno1
  filename: xfiles
```

我们可以在 ruby 中这样使用它

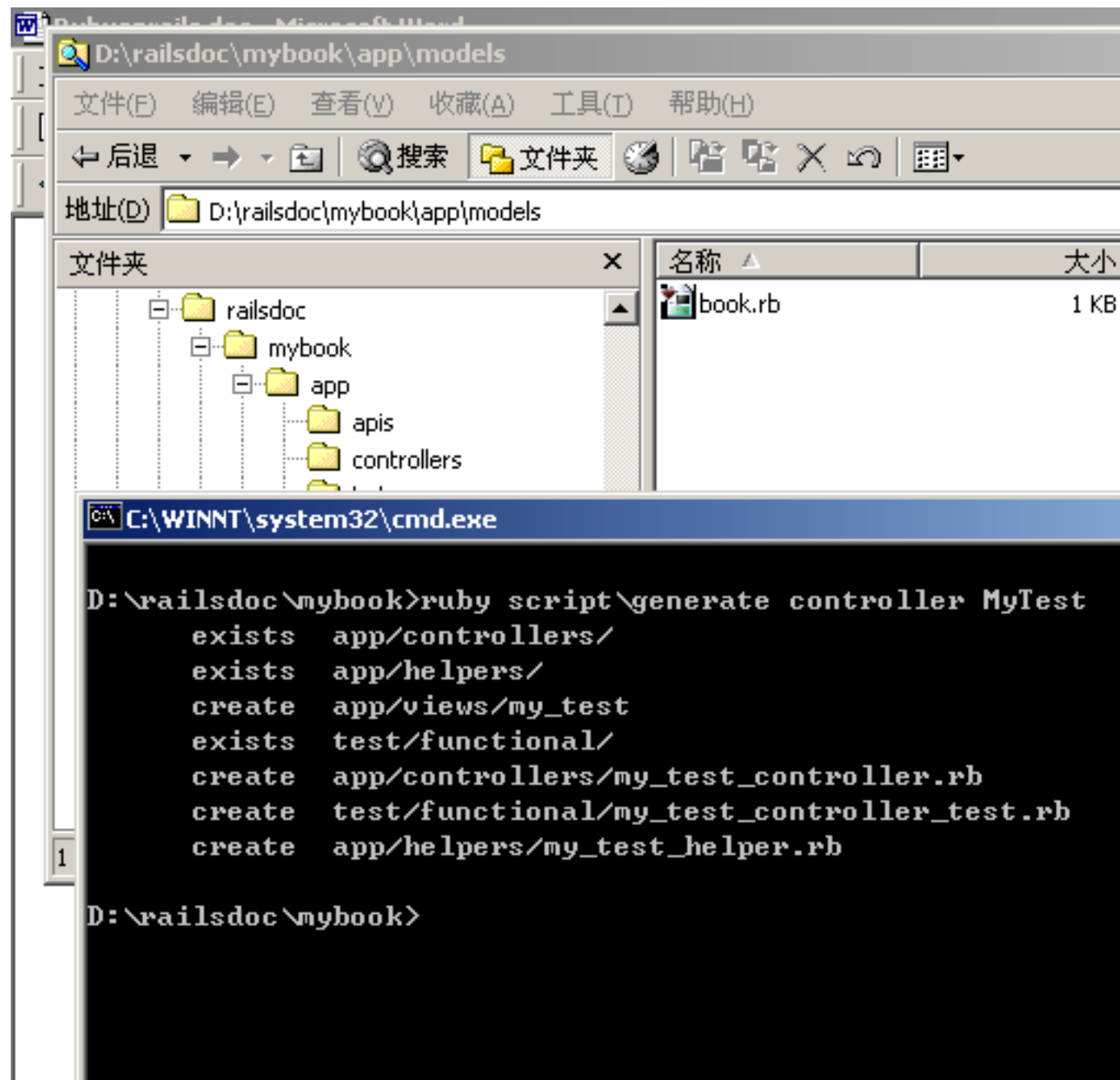
```
require 'yaml'
```

```
config = YAML.load(File.open("conf/config.yaml"))
config["host"]          -- 输出结果 "127.0.0.1"
config["prefs"]["password"] -- 输出结果 "trustno1"
```



由于重新定义了数据库配置，需要重新启动 webrick 服务器才能让 rails 找到数据库配置信息。

4：在 Mybook 目录下打开 DOS 命令行窗口，  
运行 `ruby script\generate model book`



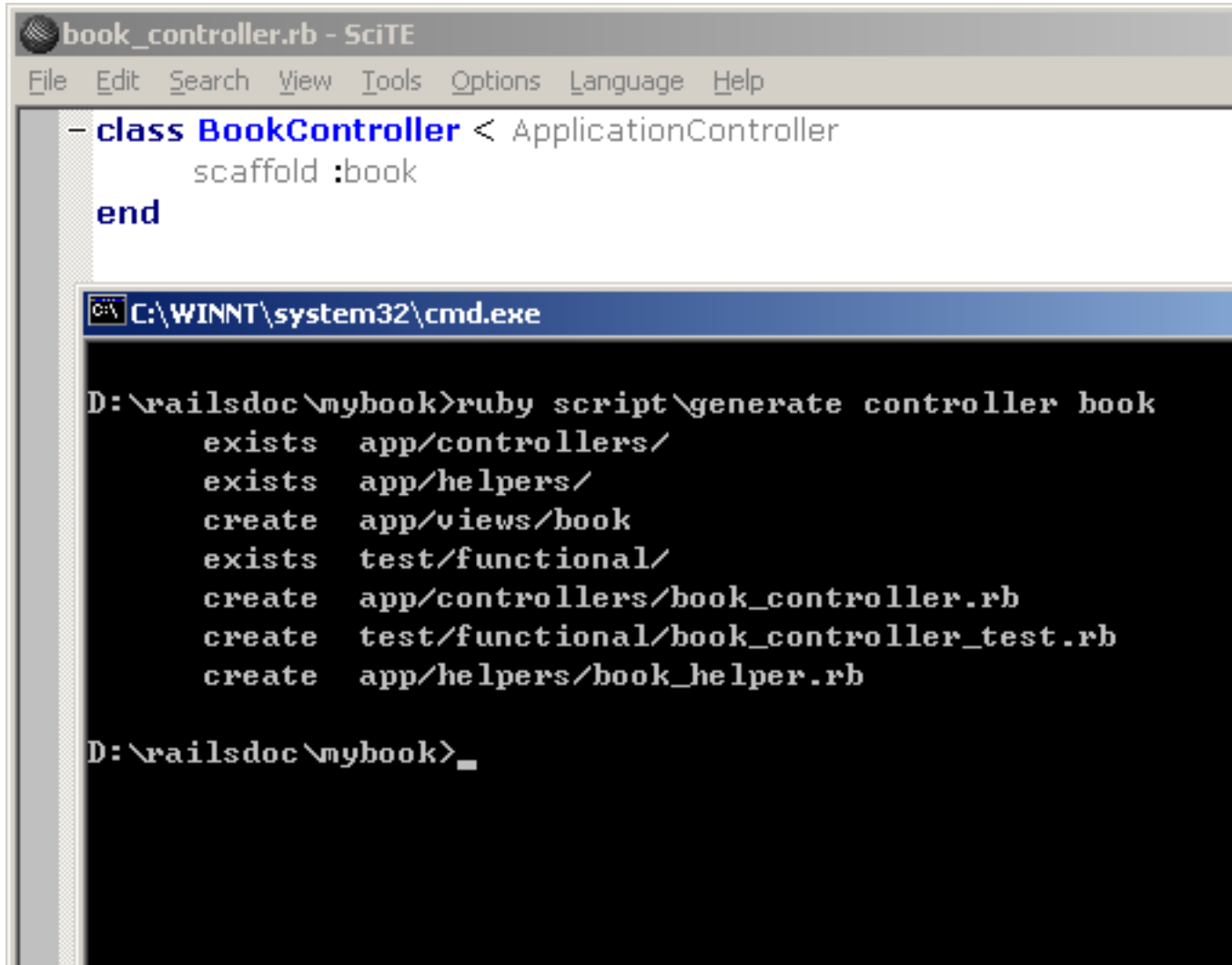
rails 生成 Model 类 Book 的骨架文件 `book.rb`。在这里 Rails 将数据库中的 `books` 表映射到 Book 类。这里 Rails 再次使用了习惯约定，那就是数据库表以复数形式命名，而与之对应的 Model 类，使用单数形式。这里 rails 非常智能，它理解英文的复数规则，会自动将 `person` 类和复数形式 `people` 数据库表做映射。

打开 `book.rb`

```
class Book < ActiveRecord::Base
```

end

5：在 Mybook 目录下打开 DOS 命令行窗口，  
运行 `ruby script\generate controller book` 生成 controller 骨架文件 `book_controller.rb`。



```
book_controller.rb - SciTE
File Edit Search View Tools Options Language Help
- class BookController < ApplicationController
  scaffold :book
end

C:\WINNT\system32\cmd.exe

D:\railsdoc\mybook>ruby script\generate controller book
exists app/controllers/
exists app/helpers/
create app/views/book
exists test/functional/
create app/controllers/book_controller.rb
create test/functional/book_controller_test.rb
create app/helpers/book_helper.rb

D:\railsdoc\mybook>
```

编辑 `D:\railsdoc\mybook\app\controllers\` 下的 `book_controller.rb`

好了，魔法正式开场了。到现在我们还没有编写一行代码。如上所示

```
class BookController < ApplicationController
```



```
scaffold :book  
end
```

`scaffold :book` – 这句是你惟一需要写的代码，它生成了数据库表对应的 **CRUD** 操作，而且令人激动的是它生成了对应的视图模板文件。

我们来测试一下，在浏览器中打开 <http://127.0.0.1:3000/book/new>

Scaffolding - Microsoft Internet Explorer

文件(F) 编辑(E) 查看(V) 收藏(A) 工具(T) 帮助(H)

后退

前进

停止

刷新

搜索

收藏夹

媒体

时钟

打印

地址(D) http://127.0.0.1:3000/book/new

# New book

Title

Description

Buydate

2005

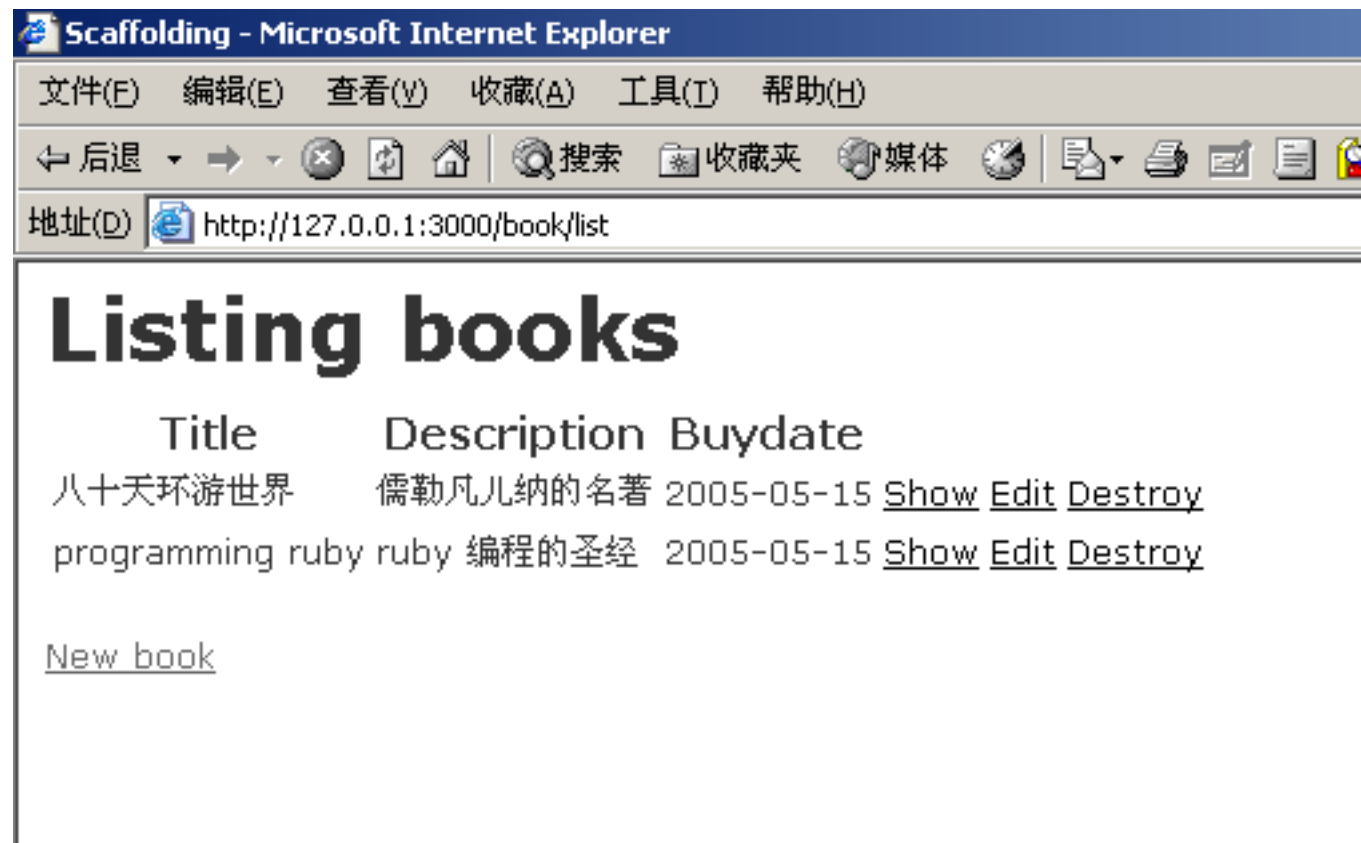
May

15

Create

[Back](#)

整个数据库表的登记界面已经为你做好了，你现在连 `html` 页面都没有打开过，这就是 rails 给我们带来的神奇之处。我们试着登记一些信息。



你会看到我们已经拥有了 `book` 表的所有页面操作，`show`（显示表记录），`edit`（编辑表记录），`destroy`（删除表记录）。

`scaffold :book` 为我们完成了所有这一切，`scaffold` 是一个函数，它接受 `:book` 作为参数，这里 `:book` 是一个 `symbol`，在 `ruby` 中，`symbol` 可以理解为变量名，比如一个变量 `book = 1`，如果直接引用 `book`，会得到 `1`，如果 `:book`，就指变量名本身。

`scaffold :book` 会生成 `list`, `show`, `edit`, `delete` 四个操作，对应表的操作。默认的操作页面很平常。如果我们想定制，也很简单，只要我们自定义自己的四个函数就可以了。

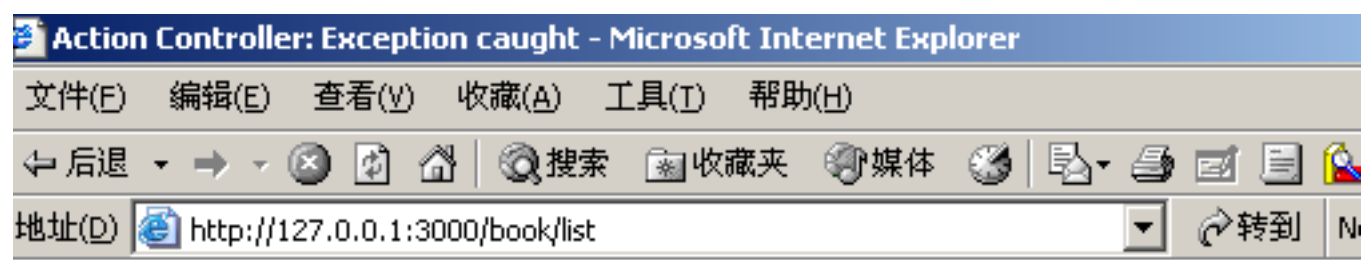
我们重新编辑 `book_controller.rb` 文件

```
class BookController < ApplicationController
  scaffold :book

  def list

  end
end
```

刷新浏览器页面



# Template is missing

Missing template ./script/../config/../app/views//book/list.rhtml

提示说我们缺少模板文件。因为我们自定义了 `list` 函数，所以 rails 将不再使用 scaffold 给我们生成的 `list` 版本。所以我们必须编写自己的模板文件。

当我们用命令 `ruby script\generate controller book` 生成 `controller` 文件的时候，rails 同样在 `view` 目录下生成了 `book` 目录用于放置我们自己写的用于 `controller` 调用显示的模板文件。这里我们需要自己写一个 `list.rhtml`

代码如下：

```
<html>
<head>
```

```

<title>All books</title>
</head>
<body>

<h1>Online Mybook - All books</h1>
<table border="1">
  <tr>
    <td width="80%"><p align="center"><i><b>book</b></i></td>
    <td width="20%"><p align="center"><i><b>Date</b></i></td>
  </tr>

  <% @books.each do |book| %>
    <tr>
      <td><%= link_to book.title, :action => "show", :id => book.id %></td>
      <td><%= book.buydate %></td>
    </tr>
  <% end %>
</table>
<p><%= link_to "Create new book", :action => "new" %></p>

</body>
</html>

```

我们重新编辑 book\_controller.rb 文件

```

class BookController < ApplicationController
  scaffold :book
  def list
    @books = Book.find_all
  end
end

```

再次重新刷新页面



## Online Mybook - All books

<i>book</i>	<i>Date</i>
<a href="#">八十天环游世界</a>	2005-05-15
<a href="#">programming ruby</a>	2005-05-15

[Create new book](#)

OK 了，现在我们很容易的定制了列表页面。  
在这里稍微做一些解释：

```
@books = Book.find_all
```

这条语句告诉 rails ，调用 Book 模型类中的 find\_all 函数从数据库表 books 中把所有记录读出赋值到 @books 实例变量中。接着 rails 寻找 list.rhtml 以生成结果页面传递到浏览器页面显示。在 list.rhtml 中

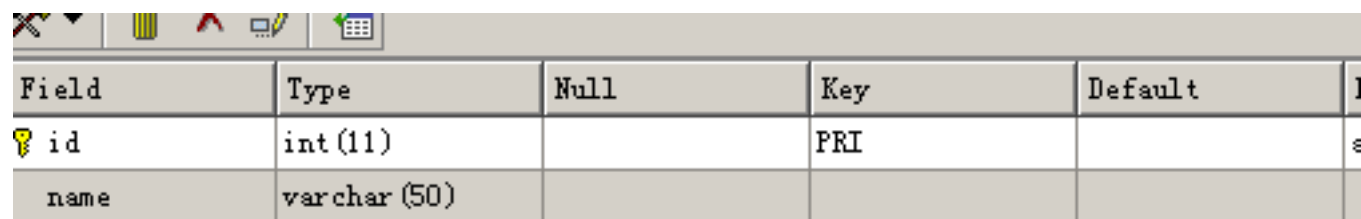
```
<% @books.each do |book| %>
  <tr>
    <td><%= link_to book.title, :action => "show", :id => book.id %></td>
    <td><%= book.buydate %></td>
  </tr>
<% end %>
```


从 controller 中传递过来的 @books 变量调用 each iterator 循环边历 @books 中的数据

集。熟悉 asp, jsp 的人对上面的代码一定不陌生。这里使用的代码只不过是嵌入的 ruby 代码而已。即使不太了解嵌入 ruby 脚本，上面的代码也很直观。

## 6：创建 categories 数据表

定义如下：



Field	Type	Null	Key	Default	
 id	int (11)		PRI		s
name	varchar (50)				

在 D:\railsdoc\mybook 目录中打开 DOS 命令行，运行下列命令为 Category 生成模型和控制器类

```
ruby script\generate controller Category
ruby script\generate model Category
```

打开 D:\railsdoc\mybook\app\controllers 中自动生成的 category\_controller.rb 编辑

```
class CategoryController < ApplicationController
  scaffold :category
end
```

保存后，重新打开浏览器窗口，访问 <http://127.0.0.1:3000/category/new>

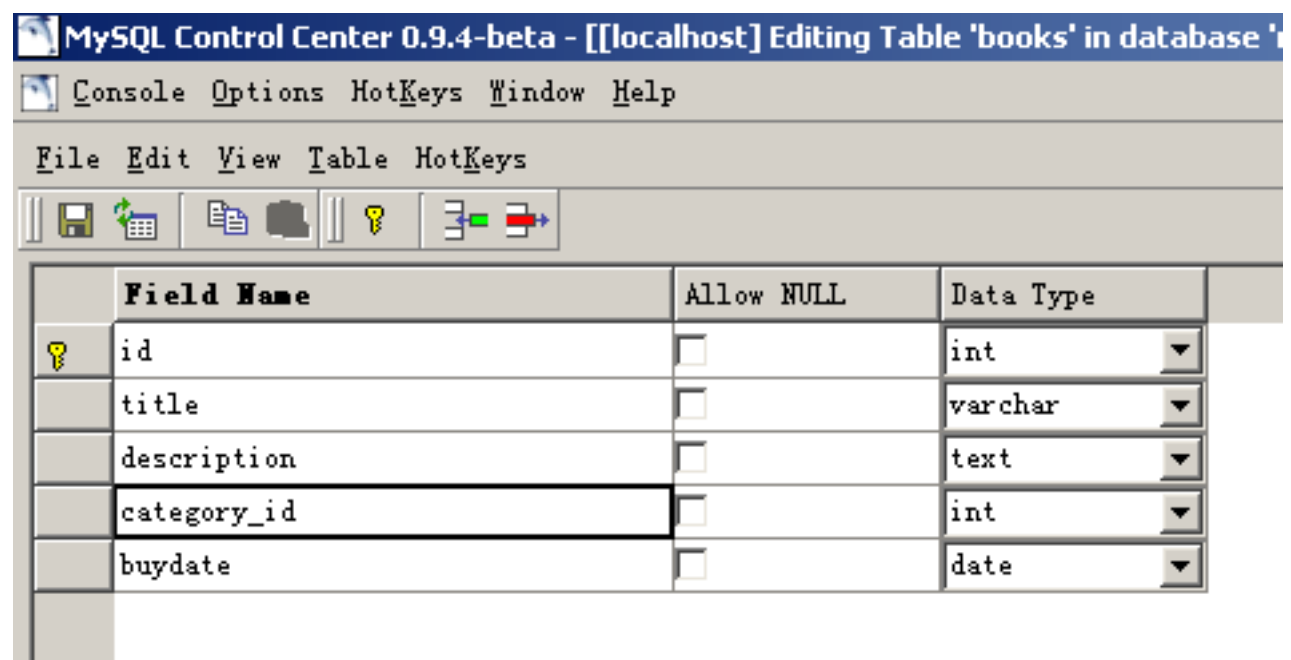


添加书籍类别数据





为了让 `books` 数据表包含类别字段, 我们需要新增加一个 `category_id` 字段, 字段定义为 `int` 型与 `categories` 表的 `id` 字段类型相同。



打开 `Model` 目录中的两个 `model` 类文件 `book.rb` 和 `category.rb` 进行编辑, 各增加一行

`book.rb` 修改如下:

```
class Book < ActiveRecord::Base
  belongs_to :category
end
```

这行代码告诉 `Rails`, 一本书属于一个类别

`category.rb` 修改如下:

```
class Category < ActiveRecord::Base
  has_many :books
end
```

这行代码告诉 `Rails`, 一个类别可以包含很多本书。

这两行申明就可以给我们生成在表关系之间导航的各种方法。例如, `@book` 变量中保存了一本书的信息, 我们想知道它属于的类别名, 我就可以使用代码 `@book.category.name`, 再比如, `@category` 保存了一个类别, 我们想知道这个类别下有多少本书, 那么我们可以通过 `@category.books` 来获得。

现在我们从新定义 edit 函数，来覆盖它的 scaffolds 版本。

编辑 book\_controller.rb 文件

```
class BookController < ApplicationController
  scaffold :book
  def list
    @books = Book.find_all
  end
  def edit
    @book = Book.find(@params["id"])
    @categories = Category.find_all
  end
end
```

edit 函数中定义的两行代码产生两个实例变量，这两个变量 @book 和 @categories 将在 edit book 页面中用到，@book 变量存放我们要编辑的书籍对象，id 参数取自网页的访问请求。@categories 变量存放数据表 categories 中所包含的所有记录。

因为我们已经从新定义了 edit action，那么我们同样需要自己编写 edit.rhtml。和前面 list action 对应 list.rhtml 类似。

打开 D:\railsdoc\mybook\app\views\book 目录，新建一个 edit.rhtml 输入如下内容

```
<html>
<head>
  <title>Edit book</title>
</head>
<body>
<h1>Edit book</h1>

<form action="../update" method="POST">
  <input id="book_id" name="book[id]" size="30"
    type="hidden" value="<%= @book.id %>" />
  <p><b>Title</b><br>
  <input id="book_title" name="book[title]" size="30"
    type="text" value="<%= @book.title %>" />
  </p>
  <p><b>Description</b><br>
  <input id="book_description" name="book[description]"
    size="30" type="text"
    value="<%= @book.description %>" />
  </p>
```

```

<p><b>Category:</b><br>

<select name="book[category_id]">
  <% @categories.each do |category| %>
    <option value="<%= category.id %>"
      <%= ' selected' if category.id == @book.category.id %>>
      <%= category.name %>
    </option>
  <% end %>
</select></p>

  <input type="submit" value="Update" />
</form>

<a href="/book/show/<%= @book.id %>">
  Show
</a> |
<a href="/book/list">
  Back
</a>

</body>
</html>

```

注意红字的内容，我们正是通过编辑 `book_controller.rb` 文件中的 `edit` 函数建立的变量 `categories` 来循环生成“类别”下来菜单内容。

打开浏览器，<http://127.0.0.1:3000/book/list> 点击任一和一条记录进入编辑页面，选择 `edit` 按钮，显示如下编辑页面



The screenshot shows a Microsoft Internet Explorer window titled "Edit book". The address bar displays "http://127.0.0.1:3000/book/edit/1". The main content area contains a form with the following fields:

- Title**: A text input field containing "八十天环游世界".
- Description**: A text input field containing "儒勒凡儿纳的名著".
- Category:**: A dropdown menu with "小说" selected.
- Update**: A button to submit the form.
- Show** and **Back**: Two links at the bottom of the form.

选择相应类别，点击 `update` 按钮。重新编辑修改现有的两个记录，从而在 `books` 表记录中增加类别信息。

我们现在编辑的 `list.rhtml` 并没有包含 `category` 显示栏目，现在从新加上

从新打开 `l1ist.rhtml` ， 增加如下红字显示的两行

```
<html>
<head>
<title>All books</title>
</head>
```

```

<body>

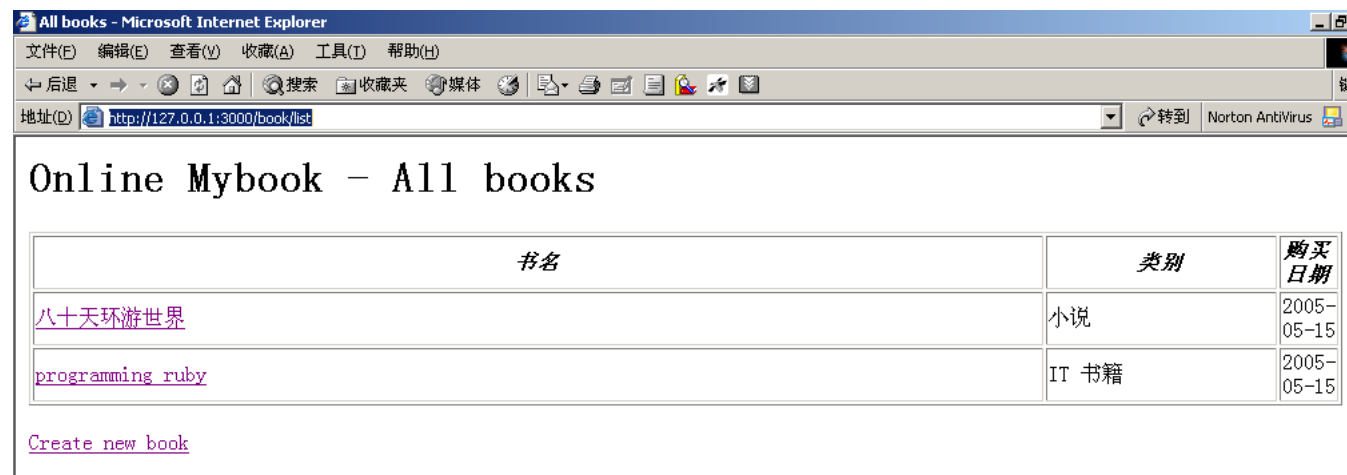
<h1>Online Mybook - All books</h1>
<table border="1">
  <tr>
    <td width="80%"><p align="center"><i><b>书名</b></i></td>
    <td width="20%"><p align="center"><i><b>类别</b></i></td>
    <td width="20%"><p align="center"><i><b>购买日期</b></i></td>
  </tr>

  <% @books.each do |book| %>
    <tr>
      <td><%= link_to book.title, :action => "show", :id => book.id %></td>
      <td><%= book.category.name %></td>
      <td><%= book.buydate %></td>
    </tr>
  <% end %>
</table>
<p><%= link_to "Create new book", :action => "new" %></p>

</body>
</html>

```

从新刷新浏览器内容（http://127.0.0.1:3000/book/list）



我们开始定义三个任务已经完成。

## 关于 Ruby on rails 的思考（新增内容）

我是从 [theserverside.com](http://theserverside.com) 上了解到 Ruby on rails 的。[oreilly.com](http://oreilly.com) 的教程的确对它的推广起了很大的作用。我本能地立刻打算翻译它，[csdn](http://csdn) 上立刻出来的两篇介绍性文章让我打消了这个念头，但是它们写的不够详细，从网友的回复来看，大家对 rails 持怀疑的态度。我觉得还是应该写一份更详细的教程来让大家认识 Ruby on rails，于是就有了这篇文档。我希望把它做成国内最详细的 Rails “入门”教程，rails 中 ruby 语言的知识点还是很多的。

我是 Ruby 语言的爱好者，但是并不是 Rails 的狂热分子。我觉得 rails 的一些缺点会影响大家采用它，有必要在这里给大家列出来。

1: Rails 毕竟还没有大型项目的成功案例。

最近出现的个人目标管理网站 [www.43things.com](http://www.43things.com) 是用 rails 制作的（国内的 [www.aimi.cn](http://www.aimi.cn) 是用 asp 技术实现的），相信不久会有更多由 Rails 制作的网站出台。好的成功案例对一门语言或技术来说是最好的广告，从当年 google 对 python 的推广就能看出来。

2: Rails 是最近刚刚出来的框架，还不够成熟。[Theserverside.com](http://theserverside.com) 就有一篇关于 active record（Rails 内部的 ORM 组件）和 hibernate ORM特性的比较文章，大家可以参看 <http://www.theserverside.com/articles/article.tss?l=RailsHibernate>。

3: Rails 是用 ruby 开发的，掌握一门新语言对很多人来说是难以接受的。这种情况表现在两部分人中，一部分是已经掌握了 java, c#, c++ 的高级程序员，他们认为这些语言已经足够成熟和完善，没必要再学习另外一种编程语言。另一部分人是刚刚入门 java, c#, c++ 的初级程序员，他们还正在努力学习流行的编程语言，更没精力来了解一种中文资料很少的脚本语言了。其实以我个人的感觉来说，一个程序员掌握一门静态编译型语言和一门动态脚本语言是非常必要的。脚本语言对于开发小程序的效率是非常高的，用 ruby 来实现代码生成器和文本文件处理的工作远比 java 要容易的多，实际上我就是在 java 项目中使用自己编写的 ruby 工具的。我相信未来也不会出现什么全能型的语言，而且掌握两种语言对一个真正的程序员来说不是那么困难的事情，而且很必要。

4: ruby 缺少组件的架构。

我觉得未来应用程序框架应该具备两个特点，那就是：“易用性”和“组件”。从现在的情形看，Rails 具备了“易用性”的特点，而大多数 Java 框架具备了“组件”的特点。我觉得：“好的框架的复杂性应该被组件封装在“黑箱”中，而“易用性”应该体现在组件的使用上”。所以 Rails 的未来应该向组件上发展，而 Java 框架应该继续在组件易用性上多做文章，两者的趋势是融合。

我觉得这个文档引起 Java 程序员的反思要多于实际使用 Rails 的意义。

## 总结:

仔细看看,使用 rails 开发程序,我们并没有写很多代码,对比起其他的 java 框架来说,它异常的简单易用。在这里 rails 给我们充分展示了“习惯约定优于配置”和“编写少量代码”的设计思想,它带来的好处是显而易见的。Ruby on rails 虽然是一个新兴的框架,尽管还不够成熟,但是它在 java 社区引起的骚动足以让我们重新审视 java 世界的一些编程思路,rails 和 ruby 在很多地方借鉴了 java 的思想,而它们自身有保持了自己特有的优势。相信会有越来越多的人采用 rails 来开发网络程序的。如果你是最新安装 rails 你会发现它依赖的模块中又新增加了 actionwebservice 模块,相信 rails 会继续给我们带来惊喜的。

作者 Eiffel Qiu

Email: [eiffelqiu@gmail.com](mailto:eiffelqiu@gmail.com)

WWW: <http://www.koalant.com>

## 参考资源: (新增内容)

### Ruby 网站资源:

ruby 官方网站: <http://www.ruby-lang.org>

ruby doc 官方网站: <http://www.ruby-doc.org>

rubyonrails 官方网站: <http://www.rubyonrails.org>

programming ruby 在线版 ( Ruby 学习的“圣经” ) :  
<http://www.rubycentral.com/book/index.html>

Ruby 开源项目的主站: <http://rubyforge.org/> , rails, rake, rubygem, one click installer 都在此下载。

Webrick 纯 Ruby服务器: <http://www.webrick.org>

Yaml 官方网站: <http://www.yaml.org>

**ONLamp.com** ROR 教程: (<http://www.onlamp.com/>) <http://www.onlamp.com/lpt/a/5546>

注: 本文示例大量参考了此教程。

### 引用的相关网站:

[www.43things.com](http://www.43things.com)

[www.aimi.cn](http://www.aimi.cn) (爱米网)

<http://www.theserverside.com/articles/article.tss?l=RailsHibernate>