Apostila de Ruby on Rails

Conceitos Básicos

Diogo Cezar Teixera Batista

Cornélio Procópio

10 de novembro de 2009

Sumário

Lista de Figuras

Lista de Códigos

1 Protocolo HTTP

1.1 O que é o protocolo HTTP

O Hyper Text Transfer Protocol (HTTP) é o protocolo usado na *World Wide Web* (www ou internet) para a distribuição e recuperação de informações.

Na ciência da computação, um protocolo é uma convenção ou padrão que controla e possibilita uma conexão, comunicação ou transferência de dados entre dois sistemas computacionais. De maneira simples, um protocolo pode ser definido como "as regras que governam"a sintaxe, semântica e sincronização da comunicação. Os protocolos podem ser implementados pelo hardware, software ou por uma combinação dos dois.

A troca de informações entre um *browser* (navegador) e um servidor *Web* é feita através desse protocolo, que foi criado especificamente para a World Wide Web.

O protocolo HTTP define uma forma de conversação no estilo pedido-resposta entre um cliente (o *browser*) e um servidor (o servidor *Web*). Toda essa conversação se dá no formato ASCII (texto puro) através de um conjunto de comandos simples baseados em palavras da língua inglesa.

1.2 Cliente HTTP

Os clientes de uma conexão HTTP são os *browsers* ou navegadores;

Dentre os navegadores atuais podemos citar:

- Mozilla Firefox;
- Internet Explorer;
- Google Chrome;
- Safari;
- Opera.

1.3 Servidores HTTP

Os servidores de em uma conexão HTTP são os servidores *Web*. Esses servidores são máquinas com grande potencialidade de armazenamento e processamento. Esses computadores devem ser configurados para poderem receber as visitas dos clientes HTTP.

Quando utilizamos a linguagem Ruby aliada com o *framework* Rails, ao instalarmos esses aplicativos um servidor *Web* local é automaticamente configurado.

1.4 Pedido HTTP

Um exemplo de pedido HTTP (é totalmente transparente para o usuário do browser):

Código 1: Exemplo de um protocolo de pedido HTTP

```
1 GET /internet/index.html HTTP/1.0
2 User-agente: Mozilla /4.5 [en] (WinNT; I)
3 AcceptP: text/plain, text/html, image/gif, image/x-xbitmap,
4 image/jpeg, image/pjpeg, image/png, */*
5 Accept-Charset: isso-8859-1, *, utf-8
6 Accept-Enconding: gzip
7 Accept-Language: en
```

Um pedido HTTP é composto de quatro partes básicas:

- o método - ação a ser realizada;
- a URI (Universal Resource Identifier) - informação requisitada;
- a versão do protocolo HTTP;
- informações adicionais - informações complementares às demais.

1.4.1 Método

O método definido será aplicado no objeto (a informação requisitada) definido pela URI; Esse método pode ser:

- *GET* - retorna o objeto;
- *HEAD* - retorna somente informações sobre o objeto, como tamanho, data de criação etc;
- *POST* - envia informações para o servidor Web. Método utilizado por scripts.

1.5 Resposta HTTP

O servidor *Web* ao receber o pedido, processa-o de modo a determinar o que deverá ser feito e monta um protocolo de resposta ao pedido solicitado;

Um exemplo de resposta HTTP (também transparente para o usuário do browser):

Código 2: Exemplo de um protocolo de resposta HTTP

```
1 HTTP/1.0 200 Document follows
2 Date: Thu, 20 Aug 1998 18:47:27 GMT
3 Server: NCSA/1.5.1
4 Content-type: text/html
5 Last-modified: Fri, 14 Aug 1998 20:14:23 GMT
6 Content-length:5807
7 <html>
8 <head><title> Navegando na Internet</title></head>
9 <body>
```

Pode-se notar que ao final do protocolo o servidor anexa o HTML que deverá ser interpretado por nosso navegador.

1.6 URL

É um acrônimo para *Uniform Resource Locator*, localizador que permite identificar e acessar um serviço na rede *Web*.

É formado por 3 partes:

- *protocolo* - qual o protocolo será utilizado? no nosso caso http;
- *nome de domínio* - onde está localizada a página na Internet? é o nome ou apelido que se dá para localizar um servidor *Web*;
- *caminho para a informação* - ao achar esse servidor qual caminho seguir para encontrar a informação desejada?

Um exemplo para ilustrar essa situação é: http://www.microsoft.com/contato
Nesse exemplo:

- *http://* - é o protocolo;
- *www.microsoft.com* - nome de domínio;
- */contato* - caminho para a informação.

2 Padrão de Projeto MVC

Model-view-controller (MVC) é um padrão de arquitetura de software. Com o aumento da complexidade das aplicações desenvolvidas torna-se fundamental a separação entre os dados (*Model*) e o *layout* (*View*). Desta forma, alterações feitas no *layout* não afetam a manipulação de dados, e estes poderão ser reorganizados sem alterar o layout.

O model-view-controller resolve este problema através da separação das tarefas de acesso aos dados e lógica de negócio, lógica de apresentação e de interação com o utilizador, introduzindo um componente entre os dois: o *Controller*.

MVC é usado em padrões de projeto de software, mas MVC abrange mais da arquitetura de uma aplicação do que é típico para um padrão de projeto.

2.1 Por que MVC?

Atualmente muitos softwares utilizam o padrão MVC como base de funcionamento e, no caso, não é à toa, pois a abordagem é realmente muito boa e a lógica por trás faz "valer a pena" adotar o MVC.

O MVC pode ser entendido como uma divisão de tarefas em um aplicativo. Cada um dos 3 tem sua função bem definida (na teoria) e executa exatamente o que deve; nada além, nada aquém.

Com o desenvolvimento e evolução dos programas e, consequentemente, da forma de se fazer os programas, novas abordagens tiveram que ser pensadas para facilitar a programação e garantir que os softwares, depois de prontos, fossem mais facilmente manuteníveis. A partir disso surgiu o conceito de dividir tarefas, de garantir com que cada "camada" da aplicação tenha seu próprio escopo e definição e que a comunicação entre todas elas se dê de maneira eficiente e controlada.

2.2 Entendendo o Model, o View e o Controller

Na maioria das fontes que você pesquisar sobre MVC, geralmente vai encontrar primeiramente a explicação de "View" provavelmente por ser a mais fácil de entender e/ou para não "assustar" muito no primeiro contato com o padrão de projeto. Mas, para seguir corretamente o acrônimo, serão apresentados, respectivamente, o Model (Modelo), o View (Visualização) e o Controller (Controle).

- *Model* - Tenha uma coisa em mente: quando pensar em *Model*, pense em estruturas de dados! Num software baseado em MVC, é o *Model* que tem o contato com as informações

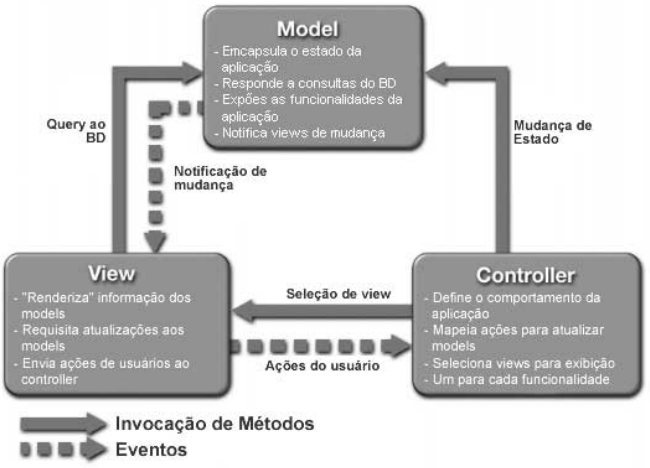
armazenadas e que são mostradas, estejam elas em um banco de dados, arquivo XML, ou onde quer que estejam. É no *Model* e somente no *Model* que as operações de CRUD devem acontecer.

- *View* - É a apresentação, é o que aparece, é o que é visualizado por quem usa o sistema. É no View que as informações, sejam elas quais forem e de de qual lugar tenha vindo, serão exibidas para a pessoa logicamente acompanhadas de um bom design, uma boa estrutura organizacional, um ambiente agradável para quem está vendo, e muitos outros.

- *Controller* - Como sugere o nome, é responsável por controlar todo o fluxo do programa. É o "cérebro" e o "coração" do aplicativo; é no Controller que se decide "se", "o que", "quando", "onde"e tudo o mais que faz com que a lógica funcione. Desde o que deve ser consultado no banco de dados à tela que vai ser exibida para quem usa o programa/sistema, é no Controller que tudo isso deve ser definido.

Para facilitar ainda mais, veja esta representação esquemática do modelo MVC:

Figura 1: Representação esquemática do modelo MVC



3 Introdução a Rails

3.1 O que é Rails?

Rails é um *framework*[1] feito em Ruby que funciona no conceito MVC: Model, View, Controller, onde é separado o modelo de dados, a interface do usuário e o controle lógico do programa, permitindo que alterações em qualquer uma dessas partes tenham pouco impacto nas outras.

3.2 O que é Ruby?

Ruby é uma linguagem de script interpretada para programação orientada a objetos de um modo fácil e rápido. Ela tem vários recursos para processar arquivos de texto e para fazer tarefas de gerenciamento de sistema (assim como o Perl). Ela é simples, direto ao ponto, extensível e portável. Oh, preciso mencionar, é totalmente livre, o que significa não só livre de precisar pagar para usá-la, mas também a liberdade de usar, copiar, modificar e distribuí-la.

Dentre seus recursos podemos citar:

- Ruby tem uma sintaxe simples, parcialmente inspirada por Eiffel e Ada;
- Ruby tem recursos de tratamento de exceções, assim como Java e Python, para deixar mais fácil o tratamento de erros;
- os operadores do Ruby são açúcar sintático para os métodos. Você pode redefini-los facilmente;
- Ruby é uma linguagem completa e pura orientada a objetos. Isso significa que todo dado em Ruby é um objeto, não do jeito de Python e Perl, mas mais do jeito do SmallTalk: sem exceções. Por exemplo, em Ruby, o numero 1 e uma instância da classe *Fixnum*;
- A orientação a objetos do Ruby é desenhada cuidadosamente para ser completa e aberta a melhorias. Por exemplo, Ruby tem a habilidade de adicionar métodos em uma classe, ou até mesmo em uma instância durante o runtime! Então, se necessário, a instância de uma classe pode se comportar diferente de outras instâncias da mesma classe;
- Ruby tem heranca única, de propósito. Mas entende o conceito de modulos (chamados de Categories no Objective-C). Módulos são coleções de métodos. Toda classe pode importar um modulo e pegar seus métodos;

[1] pode ser definido como uma software estrutura de auxílio ao desenvolvimento de outros softwares, visando prover agilidade e eficiência para que o programador se livre da implementação de código repetitivo e ... chato.

- Ruby tem um *garbage collector* que realmente é do tipo marca-e-limpa. Ele atua em todos os objetos do Ruby. Voce não precisa se preocupar em manter contagem de referências em libraries externas;
- Escrever extensões em C para Ruby e mais fácil que em Perl ou Python, em grande parte por causa do garbage collector, e em parte pela boa API de extensões. A interface SWIG também esta disponível;
- Ruby não precisa de declaracão de variáveis. Apenas usa a convenção de nomenclatura para delimitar o escopo das variáveis;
- Ruby pode carregar bibliotecas de extensão dinamicamente, se o sistema operacional permitir;
- Ruby tem um sistema de threading independente do sistema operacional;
- Ruby é altamente portável: ela é desenvolvida em sua maioria no Linux, mas funciona em muitos tipos de UNIX, DOS, Windows 95/98/Me/NT/2000/XP, MacOS, BeOS, OS/2, etc.

4 Instalação

Para instalar o Rails, devemos primeiro instalar o Ruby.

Para isso abra o executável e siga todos os passos até o final da instalação;

A linguagem Ruby conta com um auxiliar chamado gem que funciona de forma parecida com os instaladores de pacotes nas distribuições linux;

Para instalar o gem, abra o prompt de comando (Iniciar ¿ executar ¿ "cmd") vá até a pasta onde o gem foi extraído, e execute o comando:

```
ruby setup.rb
```

Depois de instalar o pacote gem basta executar o seguinte comando para instalar o Rails:

```
gem install rails
```

Por fim, é necessário copiar uma dll de acesso ao banco de dados MySql. Para isso, copie o arquivo libmysql.dll para a pasta de instalação do Ruby: ruby/bin

Os arquivos podem ser obtidos através so site: *http://www.rubyonrails.pro.br/down*