Criando aplicações web com Ruby on Rails

Plínio Balduino

August 11, 2012

Contents

1

Introdução

1.1 A quem se destina esse curso

Este curso foi criado para pessoas que já tenham alguma experiência, mínima que seja, em desenvolvimento de software e que tenham alguma mínima experiência com desenvolvimento para web.

1.2 Termos técnicos e nomenclaturas

Alguns termos técnicos serão apresentados em inglês, ao invés de forçar alguma tradução que fuja do jargão normalmente encontrado nos livros e sites. Se por um lado eu posso utilizar design patterns ao invés de padrões de projeto, que é uma tradução incompleta e não de toda correta, por outro eu posso utilizar padrão arquitetural ao invés de architetural pattern, por considerar que a tradução corresponde ao significado original.

Sinta-se a vontade para discordar e apresentar sugestões de melhora, uma vez que a escolha dos termos tem caráter pessoal e subjetivo, apesar de sempre visar o máximo de clareza e didatismo possível, baseando-me em minha experiência profissional e nos termos utilizados na literatura especializada.

Terminal e Prompt de Comando

Sempre que eu disser *terminal*, entenda que estou me referindo ao **Prompt de Comandos** do *Windows* e ao **Terminal** do *Linux* e do *MacOS X*, dependendo do sistema operacional que você estiver utilizando.

1.3 O que é uma aplicação web

Basicamente, uma aplicação web é um aplicação executada via HTTP através da Internet ou de uma Intranet (rede local) utilizando um browser como ambiente de acesso ou de execução.

Segundo a Wikipédia, "uma aplicação web também é definida em tudo que se é processado em algum servidor, [como por] exemplo: quando você entra em um e-commerce a página que você acessa antes de vir até seu navegador é processada em um computador ligado a Internet que retorna o processamento das regras de negócio nele contido."

1.4 Por que usar Rails e não xyz

O framework Ruby on Rails permite que você crie aplicações web de maneira relativamente simples e de forma mais rápida do que vários dos frameworks em uso no mercado. Minha preferência pelo Rails está muito ligada ao poder e expressividade da linguagem Ruby.

Porém, após o surgimento do Rails, várias de suas boas idéias foram aproveitadas por outras ferramentas, plataformas e linguagens, aumentando também a facilidade e a produtividade dos frameworks "concorrentes".

Por exemplo: para PHP você tem o CakePHP, que foi fortemente influenciado pela arquitetura do Rails; em Java você tem o VRaptor, que é o framework que eu usaria em um projeto Web nessa linguagem; para Python você tem o Django, que também é parecido com Rails.

Recomendo sinceramente que você aprenda e experimente Rails, e também aprenda e experimente outras plataformas equivalentes, mais recentes ou não, para que você possa tirar suas próprias conclusões a respeito de qual ferramenta é melhor para determinada situação.

De qualquer maneira, evite as discussões sobre X ser melhor do que Y. Experimente e tire suas próprias conclusões.

1.5 Quais ferramentas devo usar

No campo das discussões improdutivas, existem os embates entre "editor A contra editor B", "editor contra IDE" e equivalentes.

Seguindo o conselho do tópico anterior, recomendo que você experimente as diversas opções que existem, muitas delas gratuitas, e não tenha medo de trocar de ferramenta se não se achar confortável o suficiente.

Eu já utilizei IDEs como NetBeans (gratuito), RubyMine (pago) e Aptana Studio (basicamente um conjunto de plugins para Eclipse) e achei excessivamente pesadas, apesar de todas elas apresentarem recursos muito bons para depuração e refactoring.

Atualmente, boa parte da comunidade dá preferência para editores de texto avançados, dos quais já utilizei MED (pago), Notepad++ no Windows, gedit no Ubuntu, KEdit no KDE (ambos para Linux), ViM para Windows, Linux e MacOS e, atualmente tenho usado o Sublime Text, também para as três plataformas, que considero o melhor de todos até o momento, apesar de ser pago. É esse

editor que vou utilizar no decorrer do curso, mas isso não afeta em absolutamente nada o conteúdo, nem impede a utilização de qualquer outro editor por parte do aluno.

Ainda encontra-se na comunidade quem utilize TextMate, para MacOS, e Emacs, disponível para praticamente todas as plataformas.

Para todos os editores e IDEs que eu citei aqui você vai encontrar elogios e críticas, portanto eu novamente recomendo que você encontre aquele que mais se adequa às suas necessidades e que te deixe mais confortável e produtivo.

De qualquer maneira, independente do editor/IDE escolhido, acostume-se a deixar pelo menos uma janela do Terminal/Prompt de comandos aberta para que os comandos possam ser executados por lá.

2

Ruby, a linguagem

2.1 Do Japão para o resto do mundo

2.1.1 Matz e o MRI

Em 1995, Yukihiro Matsumoto, conhecido normalmente como Matz, lançou a primeira versão pública do Ruby, a 0.9.5.

Matz disse que queria uma linguagem de script que fosse funcional e imperativa ao mesmo tempo, que fosse mais poderosa que Perl e mais orientada a objetos do que Python.

A linguagem se tornou muito popular no Japão, e só foi se tornar conhecida no Ocidente a partir de 2000, quando passaram a ser publicados livros em inglês sobre o assunto.

A explosão de popularidade veio mesmo com o lançamento do Ruby on Rails, que é o objeto desse curso e cuja história vou explicar daqui a pouco.

A implementação de referência do Ruby chama-se MRI (Matz's Ruby Interpreter, ou Interpretador Ruby do Matz), e foi a implementação oficial até o lançamento da versão 1.9, quando foi substituida pelo YARV (Yet another Ruby VM)

Outras implementações do Ruby

JRuby é uma implementação Ruby que usa a JVM (Máquina Virtual Java) para ser executada IronRuby é uma implementação Ruby que é executada no ambiente .Net (CLI) Rubinius se propõe a ser uma implementação Ruby que utilize o máximo possível de código escrito em Ruby MacRuby é uma implementação que gera código nativo para o sistema operacional MacOSX e, mais recentemente, também para iOS (iPhone e iPad) com o nome de RubyMotion, ambos da Apple. HotRuby e JsRuby são implementações Ruby que rodam sobre interpretadores ou máquinas virtuais JavaScript, como por exemplo V8, da Google, ou Rhino, que foi desenvolvido para executar sobre a JVM. mruby é uma implementação mínima da linguagem, feita pelo próprio Matz, que tem por objetivo executar scripts Ruby em dispositivos móveis Instalando Daqui para frente vou mostrar uma série de trechos de código Ruby para ilustrar melhor o que estou explicando. Caso você ainda não tenha o Ruby instalado em sua máquina, recomendo que você instale e execute os comandos dentro da ferramenta

IRB. Linux e MacOS X Você pode instalar o Ruby através de uma ferramenta chamada RVM. O passo a passo está nesse site: https://rvm.io/rvm/install/ Existem pacotes prontos para a instalação do Ruby nas principais distribuições Linux, mas estes costumam estar desatualizados. O RVM permite que você tenha sempre a versão mais recente sendo executada na sua máquina. Windows No site http://www.rubyinstaller.org/ existem instaladores para a versão do Windows que você estiver usando no momento, seja 32 ou 64 bits. Caso seja questionado, instale a versão 1.9.3 do Ruby. A versão 1.8.7 não é compatível com boa parte das bibliotecas que são escritas atualmente. Um breve guia para quem está chegando agora ao Ruby Ruby se parece com o que? Ruby é, basicamente, uma mistura de Smalltalk, Python e Perl, o que não é motivo de preocupação para quem está chegando do Java, C#, PHP ou qualquer outra linguagem que tenha algum suporte a orientação a objetos. Ruby é uma linguagem dinâmica, orientada a objetos e com algum suporte a programação funcional. Sintaxe e tipos de dados O IRB IRB é a ferramenta do Ruby que vai nos acompanhar nessa primeira parte do curso. Basicamente, o IRB permite que você digite comandos Ruby num prompt e receba o resultado imediatamente, tornando muito mais ágil o aprendizado e a experimentação dos comandos que vamos ver. Recomendo fortemente que você teste as instruções no IRB assim que forem sendo ensinadas, para que você fixe o conteúdo. Repetição é comprovadamente uma forma eficaz de aprendizado. Para usá-lo, uma vez que você tenha instalado corretamente o Ruby, abra o terminal, digite irb e pressione ENTER. O comando irb deve ser escrito em letras minúsculas. Para sair do irb, use a combinação CONTROL+D. Nos exemplos a seguir, uma linha que começa com =¿ indica a saída ou o resultado do comando, e não deve ser digitada. Exemplos: 1 + 1 =¿ 2

PI = 3.14159 =¿ 3.14159

radius = 5 =¿ 5

2 * PI * radius =¿ 31.4159

Padrões de nomenclatura Assim como a maioria das linguagens, Ruby tem uma padronização clara e simples sobre como nomear variáveis, classes e arquivos. Seguindo essas regras, seu codigo se tornará naturalmente mais claro aos olhos de outros desenvolvedores, e você conseguirá compreender também os códigos escritos por terceiros. Nomes de arquivos Os arquivos geralmente são nomeados com letras minúsculas e um caracter de sublinhado para separar palavras ou termos. Exemplos: title_spec.rb

products.rb

product_controller.rb

application_helper.rb

Classes e modules Classes e modules são nomeados utilizando-se uma convenção chamada Pascal Case, onde a primeira letra de cada palavra é escrita em maiúscula, mas sem o uso de espaços ou sublinhado. Exemplos: class Products

class ProductController

module ApplicationHelper

Constantes Constantes são valores que não devem e não podem ser alterados durante a execução da

aplicação. São geralmente utilizados para tornar o código mais legível e evitar o uso de 'números mágicos', que são valores em um comando que não tem signficado algum para outros desenvolvedores. Constantes são sempre escritos em letras maiúsculas, com sublinhado como separador de termos. Exemplos: circumference = 2 * PI * radius

puts APPLICATION_NAME

Variáveis Variáveis são nomeadas utilizando-se apenas letras minúsculas e, quando necessário, sublinhado para separar os termos. Exemplos: circle_circumference = 2 * PI * radius

name = first_name + " " + last_name

Métodos e Funções Métodos e funções são nomeados utilizando a mesma regra das variáveis: apenas letras minúsculas e, quando necessário, sublinhado para separar os termos. Exemplos: circumference = calculate_circle_circumference(radius) products.find :all

Comentários Comentários são textos de caráter meramente informativo, que servem como lembrete ou documentação para o desenvolvedor. Normalmente são ignorados pelo compilador ou interpretador. Exemplos: # este é um comentário de uma linha # outra linha de comentário =begin Aqui temos um bloco de comentários, que pode ter quantas linhas forem necessárias Você pode escrever aqui seu nome, a data, o que você quiser, terminando o bloco com a linha abaixo =end

Atribuição e comparação Como acontece em várias outras linguagens, Ruby tem símbolos diferentes para atribuição e para comparação. Os exemplos daqui por diante podem ser testados no IRB. var1 = 3 =¿ 3 var1 =¿ 3

Vemos aqui que var1 passou a valer 3 Os exemplos daqui por diante podem ser testados no IRB. var1 == 3 =¿ true var1 != 4 =¿ true

Vemos aqui que var1 é igual a 3, e que var1 é diferente de 4. == é o sinal para comparação de igualdade != é o sinal para comparação de diferença Muito cuidado para não confundir = e == dentro de um código. Isso acontece com uma frequência bem maior do que gostariamos. Condicionais Ruby tem os seguintes operadores condicionais: if / else / elsif Os exemplos daqui por diante podem ser testados no IRB. var1 = 3 # caso em que apenas uma condição é verificada if var1 == 3 puts "entrou aqui" end # será exibido o texto "entrou aqui"

caso a condição seja verdadeira executa uma instrução, caso contrário executa outra. if var1 != 3 puts "não deveria entrar aqui" else puts "mas aqui sim" end # será exibido o texto "mas aqui sim"

existe também o caso em que você queira checar mais de dois cenários possíveis: idade = 59 if idade ¡ 6 puts "criança que não paga passagem" elsif idade ¡ 65 puts "passageiro pagante" else puts "idoso que não paga passagem" end # será exibida uma mensagem de acordo com a idade informada

unless Unless é uma instrução característica do Ruby que funciona como um if invertido. Pode ser lido como "a menos que". var1 = 3 # a menos que var1 seja igual a 4 unless var1 == 4 puts "var1 é diferente de 4" end # imprime "var1 é diferente de 4"

a construção acima funciona exatamente como a abaixo if var1 != 4 puts "var1 é diferente de 4"

end

case / when Você utiliza a instrução case quando precisa verificar se uma entre muitas condições é válida. É uma forma mais organizada do que o if/elsif onde você pode atribuir o resultado a uma variável. Usando o mesmo exemplo: idade = 32 mensagem = case idade when 0..5 "criança que não paga passagem" when 5..64 "passageiro pagante" when 65..150 puts "idoso que não paga passagem" else "idade inválida" end puts mensagem =¿ passageiro pagante

condicional no fim da instrução Outra característica do Ruby que torna a linguagem mais expressiva e legível é a possibilidade de se utilizar condicionais no final da instrução. var1 = 3 puts "var1 é igual a 3" if var1 == 3 =¿ var1 é igual a 3 puts "var1 é diferente de 4" unless var1 == 4 =¿ var1 é diferente de 4

Avaliação de expressões em uma Strings Apesar do nome pomposo, significa apenas que é possível executar instruções e retornar resultados dentro de uma String, eliminando em parte a necessidade de converter dados para String e facilitando a formatação de textos. var1 = 3 # assim você causa um erro, por String e Fixnum são tipos diferentes puts "var1 é igual a " + var1 =¿ TypeError: can't convert Fixnum into String # você pode fazer assim, que é como a maioria das linguagens funciona puts "var1 é igual a " + var1.to_s =¿ var1 é igual a 3 # ou você pode fazer do jeito Ruby, usando avaliação de expressões de String puts "var1 é igual a #{var1}" =¿ var1 é igual a 3 # bem mais limpo e simples puts "o dobro de var1 é #var1 * 2" =¿ o dobro de var1 é 6

Porém, esse recurso só funciona se você utilizar aspas (double quotes) como separador de Strings. Se você usar apóstrofos (single quotes - não existem aspas simples e aspas duplas), será exibido o valor literal do texto, sem qualquer avaliação. var1 = 3 puts 'var1 é igual a #var1' =¿ var1 é igual a #var1 puts 'o dobro de var1 é #var1 * 2' =¿ o dobro de var1 é #var1 * 2

Blocos Blocos são construções usadas para passar trechos de código para dentro de funções, personalizar comportamentos e iterar listas e coleções de itens. Por mais confuso que pareça explicando, é muito simples de entender fazendo. Blocos são separados por e quando tem apenas uma linha, ou por do / end quando tem várias linhas. lista = [1, 2, 3, 4, 5] lista.each—item— puts item =¿ 1 =¿ 2 =¿ 3 =¿ 4 =¿ 5 =¿ 6

Veremos inúmeros outros casos de uso de blocos no decorrer do curso Loops e iterações Ruby permite que você percorra uma lista ou execute determinada ação por um número específico de vezes de várias formas. for while / until each times Operadores lógicos Criando funções e métodos Linguagem dinâmica e fortemente tipada Ruby é considerada uma linguagem dinâmica porque checa os tipos de dados em tempo de execução, ao contrário de Pascal/Delhi, Java ou C#, por exemplo. Isso significa, basicamente, que você não informa o tipo do dado ao declarar uma variável, assim como acontece no PHP. Porém, diferente do PHP, mas assim como Pascal, Java ou C#, Ruby é uma linguagem fortemente tipada. a = 0 puts a.class =¿ Fixnum a + "texto" =¿ TypeError: String can't be coerced into Fixnum

Orientação a objetos Praticamente tudo na linguagem é tratado como um objeto. Isso significa que você pode tratar, por exemplo, um número como um objeto. Executando os comandos no IRB, temos os resultados abaixo: 1.next =¿ 2 21.*(2) =¿ 42 # ou simplesmente 21 * 2 =¿ 42 "Rails rocks".size

=¿ 11 (1..20).to_a.delete_if{—item— item.odd?} =¿ [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]

As principais diferenças do Ruby em relação a outras linguagens orientadas são: Ruby não tem suporte a heranças múltiplas, como acontece em Python e C++ Ruby não adota os conceitos de Interface e Classes Abstratas, como acontece em Java, C# e C++ Ao invés disso, Ruby utiliza Mixin e Duck Typing, que vou explicar a seguir Classes e modules Herança, mixin e duck typing

Rails, o framework Um acampamento e sinais alienígenas Quem usa Rails? Coisas que você precisa saber antes de começar O que é MVC A estrutura de diretórios de uma aplicação Rails Instalando Gerando sua primeira aplicação O arquivo Gemfile Controllers: o meio de campo da sua aplicação Entendendo o conceito de controllers e actions Gerenciando rotas A página inicial Criando e alterando views Cada view com sua action Alterando e criando layouts Simplificando a view com partials Renderizando para outros formatos Utilizando HAML para gerar as views Models Models e tabelas Migrations Adicionando models Modificando models: adicionando, alterando ou excluindo campos Desfazendo modificações Helpers Eles ajudam mesmo Indo além do feijão com arroz Testes Testando controllers Testando models Preparar sua aplicação para múltiplos idiomas I18N: Internationalization L10N: Localization Asset Pipelines O que é Coffeescript e Dart SASS, SCSS e LESS Imagens Enviando emails

E agora, o mundo real Controle de versão com Git GitHub BitBucket DropBox Instalando sua aplicação num servidor de verdade Nginx Apache Heroku Webbynode Utilizando sub-domínios Criando sua própria aplicação Agora é com você

Onde aprender mais Literatura recomendada Cursos Comunidades e canais de comunicação