Rails 3
Básico

Cloves Carneiro Jr.
Rida Al Barazi

Novatec

Editor: Rubens Prates
Tradução: Rafael Zanolli
Revisão gramatical: Jeferson Ferreira
Editoração eletrônica: Camila Kuwabata e Carolina Kuwabata

ISBN: 978-85-7522-265-2

Histórico de impressões:

Abril/2011 Primeira edição

Novatec Editora Ltda.
Rua Luís Antônio dos Santos 110
02460-000 – São Paulo, SP – Brasil
Tel.: +55 11 2959-6529
Fax: +55 11 2950-8869
Email: novatec@novatec.com.br
Site: www.novatec.com.br
Twitter: twitter.com/novateceditora
Facebook: facebook.com/novatec
LinkedIn: linkedin.com/in/novatec

CAPÍTULO 1

Introdução ao framework Rails

O Rails é um framework de aplicação web para a linguagem de programação Ruby. Ele é bem elaborado e prático, podendo ajudar-lhe a construir sites poderosos rapidamente, com código limpo e de fácil manutenção.

O objetivo deste livro é dar-lhe um entendimento extenso e completo da construção de aplicações web com o Rails. Isso significa mais do que simplesmente mostrar-lhe como utilizar recursos e capacidades do framework, e mais do que somente dar-lhe um conhecimento funcional da linguagem Ruby. O Rails não é apenas outra ferramenta: ele representa uma forma de pensar. Para compreendê-lo completamente, é essencial que você conheça seus dados subjacentes, sua cultura, estética e filosofia de desenvolvimento web.

Caso ainda não a conheça, você certamente perceberá que a expressão "o jeito Rails" ("the Rails way") surgirá de vez em quando. Ela se assemelha a um termo que existe na comunidade Ruby há muitos anos: o jeito Ruby. O jeito Rails é geralmente o modo mais fácil – o caminho de menor resistência, se preferir. Isso não quer dizer que você não possa fazer tudo do seu jeito, nem deve significar que o framework é limitador. Quer dizer apenas que se você escolher sair do caminho conhecido, não deve esperar que o Rails torne tudo mais fácil. Se você está habituado ao mundo do Unix, talvez considere que esta noção se assemelhe ao mantra: "Faça a coisa mais simples e funcional possível" (Do the simplest thing that could possibly work). Pois você está certo. A meta deste capítulo é introduzi-lo ao "jeito Rails".

Ascensão da aplicação web

Aplicações web estão se tornando cada vez mais importantes. Conforme o mundo se torna mais conectado, elementos de nossa rotina fazem, cada vez mais, parte da web. É nela que verificamos nossos e-mails e vamos ao banco. Participamos de cursos, compartilhamos fotos, fazemos o upload de vídeos, gerenciamos projetos e nos conectamos com pessoas de todo o mundo, sem sair do conforto de nossos navegadores. Conforme as conexões se tornam mais rápidas, e cresce a adoção de recursos de banda larga, softwares com base na web e aplicações similares cliente/servidor parecem estar destinados a substituir os softwares distribuídos por meios mais tradicionais (leia-se superados).

Para os clientes, software com base na web é mais conveniente, permitindo que as pessoas produzam mais, onde quer que estejam. Também funcionam nas plataformas que aceitam um navegador web (o mesmo que dizer todas elas), e não exigem que algo seja instalado ou baixado. Da mesma forma, se o valor das ações do Google serve de indicação, aplicações web estão cada vez mais em destaque. De fato, a mudança na web foi tão significativa nos anos recentes que seu atual estágio é conhecido como Web 2.0. Por todo o mundo, muitas pessoas estão conhecendo a nova web, e percebendo as vantagens das criações feitas com base nela. De recursos de e-mail, calendários, fotos e vídeo, à marcação de favoritos, operações bancárias e apostas, estamos vivendo cada vez mais dentro de nossos navegadores.

Em razão da facilidade de distribuição, o ritmo das mudanças no mercado de software com base na web é rápido. Diferente do software tradicional, que deve ser instalado em cada computador individualmente, alterações em aplicações web podem ser entregues com rapidez, tendo os recursos adicionados gradualmente. Não há necessidade de gastar meses ou anos aperfeiçoando a versão final, ou acrescentando todos os recursos antes da data de lançamento. Em vez de gastar meses em pesquisa e desenvolvimento, você pode entrar rapidamente na produção e refinar seu produto posteriormente, mesmo sem ter todos os recursos em seus devidos lugares.

Imagine produzir e enviar um milhão de CDs, e encontrar um bug em seu software apenas quando o caminhão do SEDEX desaparecer no horizonte? Esse erro lhe custaria caro. Software distribuído dessa forma costuma demorar muito para ser lançado, pois antes que uma empresa distribua o produto, deve estar certa de que ele está livre de bugs. É evidente que não existe software verdadeiramente livre de bugs, e aplicações web não estão imunes a essas características indesejadas, mas, no caso delas, correções são fáceis de implantar.

Quando uma correção é inserida no servidor que hospeda uma aplicação web, todos os usuários se beneficiam ao mesmo tempo da atualização, geralmente sem que haja qualquer interrupção no serviço. Esse é um nível de garantia de qualidade que você não pode oferecer em software comprado em lojas. Não há "service packs" que têm de ser distribuídos incansavelmente, ou updates críticos que devem ser instalados. Para aplicar as correções, muitas vezes basta recarregar o navegador. Além disso, em vez de gastar grandes quantias de dinheiro e recursos no empacotamento e na distribuição, desenvolvedores de software podem se concentrar em qualidade e inovação.

Software com base na web tem as seguintes vantagens:

- maior facilidade de distribuição;
- maior facilidade de implantação;
- maior facilidade de manutenção;

- independência de plataforma;
- podem ser acessados de qualquer lugar.

A web não é perfeita

Mesmo sendo uma ótima plataforma, a web ainda tem suas limitações. Um dos maiores problemas é o próprio navegador. Quando se trata de navegadores, há diversos candidatos, cada um com uma forma levemente diferente de exibir o conteúdo de uma página web. Mesmo que se perceba um movimento em direção à unificação, e que o estado atual de conformidade aos padrões entre os navegadores esteja melhorando continuamente, ainda há muito a ser feito. Mesmo hoje, é praticamente impossível obter 100% de compatibilidade em todos os navegadores. Algo que funciona no Internet Explorer, não necessariamente funciona no Firefox, e vice versa. Essa falta de uniformidade dificulta que desenvolvedores criem aplicações que funcionem em diversas plataformas, além de tornar mais difícil para os usuários o trabalho em seus navegadores preferidos.

Colocando de lado esses problemas com os navegadores, talvez a principal limitação que aflige o desenvolvimento na web seja sua complexidade inerente. Uma aplicação web típica tem dezenas de partes em movimento: protocolos e portas, o HTML e as CSS, o banco de dados e o servidor, o designer e o desenvolvedor, e uma enormidade de outros participantes, todos aumentando a complexidade.

Ainda assim, apesar desses problemas, o novo foco na web como uma plataforma significa que o campo do desenvolvimento está crescendo rapidamente, e superando seus obstáculos. Conforme o meio segue amadurecendo, as ferramentas e processos, há tanto tempo habituais no desenvolvimento de software lado servidor tradicional, começam agora a fazer parte do mundo do desenvolvimento web.

O bom framework web

Dentre as ferramentas que começam a fazer parte do desenvolvimento web está o framework, uma coleção de bibliotecas e ferramentas feitas para auxiliar o desenvolvimento. Projetado visando à produtividade, um bom framework oferece uma infraestrutura básica, mas completa, sobre a qual você pode construir uma aplicação.

Ter um bom framework é quase como ter uma parte de sua aplicação pronta. Em vez de começar do zero, você pode partir de uma base já estabelecida. Caso uma comunidade de desenvolvedores utilize o mesmo framework, você tem uma comunidade de apoio que pode ser consultada quando necessário. Também há uma segurança maior em relação à base utilizada, menos propensa a bugs e vulnerabilidades que desaceleram o processo de desenvolvimento.

Um bom framework web pode ser descrito da seguinte maneira:

- **Pilha completa (full stack)**: tudo de que você necessita para a construção de aplicações completas deve estar incluído em seu framework. Ter de instalar diversas bibliotecas ou configurar múltiplos componentes não é algo agradável. As camadas diferentes devem se encaixar perfeitamente.
- **Código aberto**: um framework deve ser de código aberto, preferivelmente licenciado sob uma licença liberal, como BSD ou MIT.
- **Multiplataforma**: um bom framework é independente de plataforma. A plataforma em que você decide trabalhar é uma escolha pessoal. Seu framework deve permanecer o mais neutro o possível.

Um bom framework web oferece o seguinte:

- **Lugar para tudo**: estrutura e convenções orientam um bom framework. Em outras palavras, a não ser que um framework ofereça uma boa estrutura e um conjunto prático de convenções, não se trata de um bom framework. Tudo deve ter seu devido lugar no sistema; eliminando adivinhações e elevando a produtividade.
- **Camada de abstração do banco de dados**: você não deve ser obrigado a lidar com detalhes de baixo nível do acesso ao banco de dados, nem deve ficar limitado a um engine de banco de dados específico. Um bom framework cuida de grande parte do trabalho "braçal" para você, e também funciona com praticamente todos os bancos de dados.
- **Cultura e estética que ajudem na tomada de decisões de programação**: em vez de enxergar a estrutura imposta por um framework como algo limitador, veja-a como libertadora. Um bom framework codifica suas opiniões, guiando você gentilmente. Muitas vezes, decisões difíceis são tomadas para você com base em convenções. A cultura do framework faz com que você tenha de tomar menos decisões triviais, para que possa se concentrar no que é mais importante.

Surge o Rails

O Rails é um framework de ponta para construção de aplicações web. É completo, de código aberto e tem compatibilidade de plataforma cruzada. Oferece uma poderosa camada de abstração de dados, chamada Active Record, que funciona com todos os populares sistemas de bancos de dados. Vem com um conjunto considerável de padrões e oferece um sistema comprovado, em múltiplas camadas, para organização e arquivos de programas e interesses (*concerns*)[1].

1 N.T.: a separação de interesses, ou de responsabilidades (separation of concerns, ou SoC), é o processo de separação de um programa de computador em atributos diferentes e que se sobrepõem o mínimo possível em funcionalidade. Um concern é qualquer parte de interesse ou foco em um programa, sendo geralmente sinônimo de recursos ou comportamentos (fonte: Wikipédia).

Acima de tudo, o Rails é um software de opinião, com uma filosofia que leva a sério a arte do desenvolvimento web. Felizmente, essa filosofia está centrada em beleza e produtividade. Você perceberá que, conforme aprende sobre o Rails, a codificação de suas aplicações web se tornará mais prazerosa.

Criado originalmente por David Heinemeier Hansson, o Rails surgiu primeiro como uma aplicação wiki chamada Instiki. A primeira versão, lançada em julho de 2004, do que hoje é o framework Rails foi extraída de uma aplicação funcional do mundo real: a Basecamp, da 37signals. Os criadores do Rails removeram todas as partes específicas da Basecamp, e o que restou foi o Rails.

Como foi extraído de uma aplicação do mundo real, e não construído isoladamente, o Rails é prático e destituído de recursos desnecessários. Sua meta como framework é solucionar 80% dos problemas que ocorrem no desenvolvimento web, presumindo que os 20% restantes são problemas individuais do domínio da aplicação. Pode parecer surpreendente que cerca de 80% do código de uma aplicação seja infraestrutura, mas isso não é tão absurdo quanto parece. Considere todo o trabalho envolvido na construção de uma aplicação. Desde sua estrutura de diretórios e convenção de nomenclatura, até a camada de abstração do banco de dados e a manutenção do estado.

O Rails tem padrões específicos para estrutura de diretórios, nomenclatura de arquivos, estruturas de dados, argumentos de métodos, e, bem, para praticamente tudo. Quando escrever uma aplicação Rails, espera-se que você siga as convenções que foram apresentadas. Em vez de se concentrar nos detalhes sobre como unificar toda sua aplicação, você pode se voltar aos 20% que realmente importam.

À medida que cresceu em popularidade, o Rails também se tornou maior. Isso motivou um grupo de desenvolvedores inteligentes a criar o Merb, um framework web Ruby, alternativo e modular. Depois do Merb 1.0 ter sido lançado, as equipes de desenvolvimento do Rails e do Merb concordaram em unir suas forças no que hoje é conhecido como Rails 3, que continua a oferecer todos os ótimos recursos do Merb, mas agora sob o nome do Rails.

Rails é Ruby

Há diversas linguagens de programação à disposição. Você provavelmente já conhece muitas delas. C, C#, Lisp, Java, Smalltalk, PHP e Python são todas escolhas populares. Também há aquelas que você talvez ainda não conheça: Haskel, IO e talvez até mesmo Ruby. Como as outras, a Ruby também é uma linguagem de programação, utilizada para escrever programas de computadores, incluindo, mas não apenas, aplicações web.

Antes do Rails surgir, não havia muitas pessoas escrevendo aplicações web com Ruby. Outras linguagens, como PHP e ASP, eram mais utilizadas, e grande parte da web era

construída com elas. O fato de o Rails usar Ruby é significativo, pois a Ruby é consideravelmente mais poderosa que PHP ou ASP, em termos de suas habilidades como linguagem de programação. Este é, em grande parte, outro sintoma da maturidade da web. Agora que ela está atraindo maior audiência, linguagens e ferramentas mais poderosas estão sendo utilizadas.

A Ruby é parte-chave do êxito do Rails. O Rails utiliza Ruby para criar o que é conhecido como uma linguagem de domínio específico (*domain specific language*, ou DSL). Neste caso, o domínio é o do desenvolvimento web; quando você trabalha com o Rails, é quase como se estivesse escrevendo em uma linguagem especificamente projetada para construir aplicações web – uma linguagem com seu próprio conjunto de regras e gramática. O Rails faz isso tão bem que às vezes é fácil esquecer que estamos escrevendo código em Ruby. Isso serve para atestar o poder da Ruby. O Rails aproveita por completo a expressividade dessa linguagem para criar ambientes verdadeiramente belos.

Para muitos desenvolvedores, o Rails é seu primeiro contato com a Ruby – uma linguagem que tinha poucos seguidores antes do Rails, ao menos no ocidente. Ainda que a Ruby estivesse gradualmente se tornando conhecida por programadores fora do Japão, foi o framework Rails que verdadeiramente a tornou popular.

Inventada por Yukihiro Matsumoto em 1994, é surpreendente que a Ruby tenha permanecido obscura por tanto tempo. No que se refere a linguagens de programação, a Ruby está entre as mais belas. Interpretada e orientada a objetos, elegante e expressiva, o trabalho com a Ruby é realmente prazeroso. Grande parte da graça do Rails se deve à Ruby e à cultura e estética que permeiam sua comunidade. Conforme começa a trabalhar com o framework, você rapidamente aprende que a Ruby, assim como o Rails, é rica em idiomas e convenções, que atuam para criar um ambiente de programação agradável e produtivo.

Em resumo, a Ruby pode ser descrita da seguinte maneira:

- uma linguagem de codificação interpretada e orientada a objetos;
- com sintaxe elegante e concisa;
- poderosos recursos de metaprogramação;
- indicada como linguagem hospedeira para criação de DSLs.

O apêndice A deste livro inclui uma introdução completa à Ruby. Caso queira ver sua aparência, pule agora para esse apêndice e dê uma olhada. Não se preocupe se ela lhe parecer não muito convencional à primeira vista. Você perceberá que sua leitura é fácil, mesmo que não seja um programador. Da mesma forma, não há problemas em ler este livro, e aprender com ele, na ordem em que ele se apresenta, apenas buscando o

apêndice quando necessitar de esclarecimentos. Caso esteja procurando um guia mais profundo, Peter Cooper escreveu um livro fabuloso, *Beginning Ruby: From Novice to Professional*, Second Edition (Apress, 2009). Você também perceberá que a comunidade Ruby pode sempre ajudá-lo em suas pesquisas. Não deixe de visitar *http://ruby-lang.org/pt/* para ter acesso a muitos recursos relacionados à linguagem.

Rails estimula a agilidade

Aplicações web não são conhecidas tradicionalmente por sua agilidade, tendo a reputação de dificultarem o trabalho e de serem de difícil manutenção. Talvez seja como resposta a essa noção que o Rails surgiu, ajudando a impulsionar o movimento em direção a metodologias ágeis de programação no desenvolvimento web. O Rails auxilia na obtenção dos seguintes princípios básicos do desenvolvimento de software:

- indivíduos e interações, mais que processos e ferramentas;
- software em funcionamento, mais que documentação abrangente;
- colaboração com o cliente, mais que negociação de contratos;
- responder a mudanças, mais que seguir um plano.

Assim determina o Manifesto Ágil[2], resultado de uma discussão entre 17 figuras de destaque (incluindo Dave Thomas, Andy Hunt e Martin Fowler) no campo do que era então conhecido como as "metodologias leves" para desenvolvimento de software. Hoje, o Manifesto Ágil é tido como a principal definição do desenvolvimento ágil.

O Rails foi projetado tendo em mente a agilidade, e justamente por isso obedece fielmente a cada um dos princípios do manifesto. Com ele, você pode responder às necessidades de seus clientes com rapidez e facilidade; além disso, o framework também funciona a contento durante desenvolvimento colaborativo. O Rails é capaz de tudo isso aderindo ao seu próprio conjunto de princípios, todos os quais tornam possível o desenvolvimento ágil.

O trabalho definitivo de Dave Thomas e Andy Hunt sobre a arte da programação, *The Pragmatic Programmer* (Addison-Wesley, 1999), pode ser lido quase como um guia para o Rails, que segue o princípio do "não se repita" (*don't repeat yourself*, ou DRY), os conceitos de prototipagem rápida e a filosofia do "você não precisará disso" (*you ain't gonna need it*, ou YAGNI). Manter dados importantes em texto simples, utilizar convenção, em vez de configuração, diminuir a distância entre cliente e programador e, acima de tudo, adiar decisões na expectativa de mudanças são princípios institucionalizados no Rails. Essas são algumas das razões que justificam por que ele é uma ferramenta tão adequada ao desenvolvimento ágil. Não deve surpreender que um de seus primeiros defensores tenha sido o próprio Dave Thomas.

2 http://agilemanifesto.org/iso/ptbr/

As seções a seguir apresentam a você alguns dos mantras do Rails, demonstrando quão indicado ele é ao desenvolvimento ágil. Ainda que não queiramos ser filosóficos demais, alguns destes pontos são essenciais para entender o que torna o Rails tão importante.

Menos software

Um dos princípios centrais da filosofia do Rails é o de *menos software*. Mas o que isso significa? Significa utilizar convenção, em vez de configuração, escrever menos código e eliminar pontos que elevem desnecessariamente a complexidade de um sistema. Em resumo, menos software significa menos código, menos complexidade e menos bugs.

Convenção em vez de configuração

Convenção em vez de configuração (*convention over configuration, ou CoC*) significa que você deve definir apenas configurações que não sejam convencionais.

A programação está relacionada à tomada de decisões. Se você tivesse de escrever um sistema do zero, sem o auxílio do Rails, teria de tomar muitas decisões: como organizar seus arquivos, que convenções de nomenclatura adotar, e como manipular o banco de dados, apenas para citar algumas. Se decidisse utilizar uma camada de abstração do banco de dados, teria de dedicar tempo para escrevê-la, ou encontrar uma implementação em código aberto que satisfizesse às suas necessidades. Tudo isso teria de ser feito antes mesmo de começar a modelar seu domínio.

O Rails permite que você inicie imediatamente, oferecendo um conjunto de decisões inteligentes sobre a forma como seu programa deve funcionar e reduzindo a quantidade de decisões de baixo nível que você deve tomar de início. Como resultado, você pode se concentrar nos problemas que está tentando resolver e realizar seu trabalho mais rapidamente.

O Rails vem praticamente sem nenhum arquivo de configuração, o que pode surpreendê-lo, caso esteja acostumado a outros frameworks, ou mesmo se nunca tiver utilizado um, pois sabemos que, em alguns desses casos, configurar um framework é quase metade do trabalho.

Em vez de configuração, o Rails depende de estruturas comuns e convenções de nomenclatura, sendo que todas empregam o muitas vezes citado "princípio da menor surpresa" (*principle of least surprise*, ou POLS). Tudo se comporta de forma previsível, com fácil entendimento. Há valores padrão escolhidos com inteligência para praticamente todos os aspectos do framework, evitando que você tenha de explicitamente dizer como ele deve se comportar. Isso não quer dizer que você não possa dizer ao Rails como quer que o framework se comporte: a maioria dos comportamentos

pode ser personalizada de acordo com sua vontade, satisfazendo suas necessidades específicas. Mas você terá melhores resultados e boa produtividade com os defaults, e o Rails sempre o encoraja a utilizá-los para que você possa se dedicar aos problemas mais pertinentes.

Ainda que possa manipular praticamente tudo na configuração e no ambiente do Rails, quanto maior for o número de situações nas quais você aceita os defaults, mais rápido poderá desenvolver aplicações e prever como elas funcionarão. A velocidade com a qual você pode desenvolver sem que tenha de efetuar configurações explícitas é um dos principais motivos que explicam por que o Rails funciona tão bem. Se você coloca seus arquivos no lugar certo e os nomeia de acordo com as devidas convenções, tudo *simplesmente funciona*. Se estiver disposto a concordar com os defaults, geralmente terá de escrever menos código.

O motivo de o Rails adotar essa prática se resume ao princípio de menos software. Menos software significa a tomada de menos decisões de baixo nível, tornando sua vida como desenvolvedor muito mais fácil; o que é sempre algo desejável.

Não se repita

O Rails se vale muito do princípio DRY (não se repita, ou *don't repeat yourself*), que declara que informação em um sistema deve ser expressa em um único local.

Por exemplo, considere os parâmetros de configuração de um banco de dados. Quando se conecta a um banco de dados, você geralmente necessita de credenciais, como um nome de usuário, uma senha e o nome do banco de dados com o qual deseja trabalhar. Talvez pareça aceitável incluir essa informação de conexão na consulta do banco de dados, e essa abordagem pode até funcionar quando você faz apenas uma ou duas conexões. Todavia, quando tem de fazer mais do que isso, você acaba com muitas instâncias de nome de usuário e senha espalhadas por todo o código. É preferível manter as informações de conexão em um arquivo individual, fazendo referência a elas quando necessário. Dessa forma, se as credenciais mudam, você tem de modificar apenas um único arquivo. É disso que trata o princípio DRY.

Quanto mais duplicações existem em um sistema, maior é o espaço para os bugs se esconderem. Quanto maior o número de locais em que reside uma informação, mais a aplicação tem de ser modificada quando uma alteração é necessária, e mais difícil se torna monitorar essas mudanças.

O Rails é organizado de uma forma que utiliza ao máximo este princípio. Você geralmente especifica informações em um único local e segue para trabalhar em algo mais importante.

Rails é software de opinião

Frameworks codificam opiniões. Não deve ser nenhuma surpresa que o Rails tem opiniões firmes quanto à forma como sua aplicação deve ser construída. Quando se trabalha com uma aplicação Rails, essas opiniões são impostas, quer você tenha ciência disso ou não. Uma das formas pela qual o Rails se faz ouvir é quando gentilmente (por vezes, forçosamente) conduz você na direção certa. Já mencionamos essa forma de encorajamento quando falamos sobre convenção, em vez de configuração. Você é convidado a fazer o que é certo, pois o errado é geralmente mais difícil.

A Ruby é conhecida por fazer com que certas construções programáticas pareçam mais naturais, usando o que chamamos de açúcar sintático[3] (*syntactic sugar*), que significa que a sintaxe de algo é alterada para parecer mais natural, mesmo que se comporte da mesma forma. Algo sintaticamente correto, mas que parece estranho quando digitado, costuma ser tratado com a prática do açúcar sintático.

O Rails tornou conhecido o termo vinagre sintático (*syntactic vinegar*), o exato oposto do açúcar sintático: construções programáticas indesejadas são desencorajadas, fazendo com que suas sintaxes pareçam ainda mais "azedas" (como o vinagre). Quando você escreve um trecho de código de aparência ruim, é bem provável que ele, de fato, *seja* ruim. O Rails é ótimo em tornar óbvio qual o certo a se fazer, dando a uma boa construção uma aparência mais bela, enquanto um código errado é visto como tal pelo incremento de sua feiúra.

Você pode verificar a opinião do Rails por meio de três atitudes: com o que ele faz automaticamente, com as formas a partir das quais ele encoraja o usuário a fazer o que é certo e mediante as convenções que ele pede que sejam aceitas. Você descobrirá que o Rails tem opinião própria sobre praticamente tudo que está relacionado à construção de aplicações web: como você deve nomear suas tabelas de bancos de dados, como deve nomear seus campos, quais softwares de bancos de dados e servidores devem ser utilizados, como fazer sua aplicação crescer, do que ela necessita e o que deve ser considerado um vestígio de desenvolvimento web de legado. Se aceitar sua visão de mundo, você não deverá ter problemas com o Rails.

Assim como uma linguagem de programação, um framework deve ser algo com o qual você se sinta à vontade – que reflita seu estilo pessoal e a forma como você trabalha. Costuma-se dizer na comunidade do Rails que se você está tendo dificuldades em seu uso, provavelmente não sabe o que era desenvolver aplicações web antes dele. Isso não é dito para desanimar desenvolvedores; apenas significa que para poder dar o

3 N.T.: açúcar sintático (syntactic sugar) é um termo de ciência da computação que se refere à sintaxe dentro de uma linguagem de programação projetada para facilitar a leitura e expressão de algo, ainda que existam formas alternativas de fazê-lo. Torna a linguagem mais adequada ao uso de seres humanos e permite que algo possa ser expresso mais claramente, de modo mais conciso, ou em um estilo alternativo, que alguns possam preferir (fonte: Wikipédia).

devido valor ao Rails, talvez seja necessário uma aula de história sobre as tecnologias que o antecederam. Às vezes, até que você experimente a dor, não será capaz de dar valor à cura.

Rails é de código aberto

A cultura do Rails está imersa na tradição do código aberto. Seu código-fonte é, evidentemente, aberto. Também é preciso notar que o Rails está licenciado sob uma licença MIT, uma das mais "livres" existentes.

O Rails também defende o uso de ferramentas de código aberto e encoraja o espírito colaborativo desse tipo de opção. O código que forma o Rails é 100% livre, podendo ser baixado, modificado e redistribuído por qualquer pessoa, a qualquer momento. Além disso, qualquer um pode enviar patches para bugs ou recursos, e centenas de pessoas em todo o mundo já contribuíram com o projeto nos últimos dois anos.

Você provavelmente notará que muitos dos desenvolvedores do Rails utilizam Macs. O Mac é, claramente, a plataforma preferida pela maioria dos membros da equipe de desenvolvimento do Rails, e muitos utilizam variantes do Unix (uma das quais é o Mac OS). O sistema operacional Unix é enaltecido por hackers e utilizado praticamente como opção única entre a elite desses usuários. Há diversos motivos para isso, dentre eles o fato de que o Unix é um sistema testado e comprovado, construído em um ecossistema de código aberto, com contribuições de alguns dos mais inteligentes programadores do planeta. Tendo surgido nos anos 1970, o sistema operacional Unix evoluiu, tornando-se um exemplo poderoso e sem excessos da capacidade do código aberto. Sua beleza, simplicidade e individualidade não passam despercebidas aos criadores do Rails.

Ainda que haja uma tendência evidente favorecendo as variantes do Unix quando falamos dos desenvolvedores do Rails, não se engane, o Rails tem, de fato, compatibilidade multiplataforma. Com o número crescente de desenvolvedores que utilizam o Rails em um ambiente Windows, o trabalho com o framework tornou-se fácil em todos os ambientes. Não importa qual sistema operacional seja o seu: você poderá utilizar o Rails nele. Da mesma forma, o Rails não exige nenhum editor ou IDE especial para escrever o código. Qualquer editor de texto é suficiente, desde que você possa salvar o trabalho como texto simples. O pacote do Rails inclui até um servidor web integrado e autônomo, o WEBrick, para que você não tenha de se preocupar em baixar e configurar um servidor para sua plataforma web. Quando quiser executar sua aplicação Rails em modo de desenvolvimento, basta iniciar o servidor integrado e abrir seu navegador web. Para que dificultar?

O próximo capítulo conduz você, passo a passo, pelo processo relativamente indolor da instalação e do funcionamento inicial do Rails em seu sistema. Ainda assim, antes de avançar e iniciar a criação de sua primeira aplicação, que tal falarmos sobre a arquitetura do framework Rails. Isso é importante, pois, como você verá em breve, tem forte relação com a forma como você organiza seus arquivos e onde os coloca. O Rails é um subconjunto de uma categoria de frameworks nomeados de acordo com a forma como dividem os interesses do design de programação: o padrão Model-View-Controller. Não deve surpreendê-lo que o padrão MVC seja o tópico de nossa próxima seção.

Padrão MVC

O Rails emprega um notório e reconhecido padrão de arquitetura que defende a divisão da lógica e do trabalho da aplicação em três categorias distintas: o model (modelo), a view (visão) e o controller (controlador). No padrão MVC, o model representa os dados, a view representa a interface do usuário e o controller dirige toda a ação. O verdadeiro poder reside na combinação das camadas MVC que o Rails manipula para você. Posicione seu código no lugar certo, siga as convenções de nomenclatura e tudo deverá ocorrer adequadamente.

Cada parte do MVC – o model, a view e o controller – é uma entidade separada, capaz de ser projetada e testada isoladamente. Uma alteração no model não tem de afetar as views; da mesma forma que uma alteração em uma view não deve ter efeito sobre o model. Isso significa que alterações em uma aplicação MVC costumam ser localizadas e de baixo impacto, facilitando consideravelmente a manutenção e elevando o nível de reusabilidade dos componentes.

Contraste este cenário com o de uma aplicação fortemente acoplada que mistura acesso aos dados, lógica de negócios e código de apresentação (PHP, estamos falando de você). Algumas pessoas chamam isso de código espaguete, por sua aparência bagunçada. Em tais sistemas, a duplicação é comum, e mesmo pequenas alterações podem produzir grandes efeitos em cascata. O MVC foi projetado para solucionar este problema.

O MVC não é o único padrão de design para aplicações web, mas é o que foi escolhido pelo Rails para implementação. No final das contas, é ótimo para desenvolvimento. Ao separar os interesses (*concerns*) em diferentes camadas, alterações em uma delas não têm necessariamente de causar impacto sobre as outras, resultando em ciclos de desenvolvimento mais rápidos e facilitando a manutenção.

Ciclo MVC

Ainda que o MVC opere de diferentes formas, o fluxo de controle geralmente funciona da seguinte maneira (Figura 1.1):

- O usuário interage com a interface e dispara um evento (por exemplo, envia um formulário de registro).
- O controller recebe a entrada de dados da interface (por exemplo, os dados do formulário enviado).
- O controller acessa o model, muitas vezes atualizando-o de alguma forma (por exemplo, criando um novo usuário com os dados do formulário).
- O controller invoca uma view que renderiza a interface atualizada (por exemplo, uma tela de boas vindas).
- A interface aguarda por mais interações do usuário e o ciclo se repete.

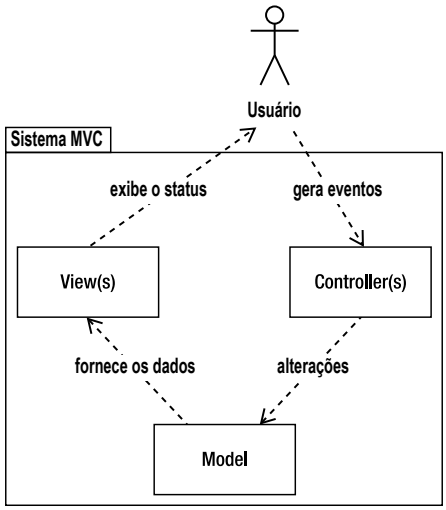

Figura 1.1 – Ciclo MVC.

Se o conceito lhe parece exageradamente participativo a princípio, não se preocupe. Ainda que livros inteiros possam ser escritos sobre este padrão, e as pessoas possam sempre discutir sua implementação mais pura, não é difícil entendê-lo – especialmente da forma como é aplicado pelo Rails.

A seguir, você examinará cada "letra" do MVC e aprenderá como o Rails lida com cada uma delas.

Camadas do MVC

As três camadas do padrão MVC trabalham juntas da seguinte maneira:

- **Model (modelo)**: a informação com a qual trabalha a aplicação.
- **View (visão)**: a representação visual da interface do usuário.
- **Controller (controlador)**: o diretor da interação entre o model e a view.

Models (Modelos)

No Rails, a camada model representa o banco de dados. Ainda que chamemos toda a camada de model, aplicações Rails geralmente são formadas por diversos models individuais, sendo que cada um (habitualmente) pode ser mapeado a uma tabela do banco de dados. Por exemplo, um model `User` pode mapear para uma tabela de nome `users`. O model `User` assume responsabilidade por todo o acesso à tabela `users` no banco de dados, incluindo criação, leitura, atualização e exclusão de linhas. Dessa forma, se quiser trabalhar com a tabela e, digamos, procurar alguém por seu nome, você o faz pelo model, da seguinte maneira:

```
User.find_by_name('Linus')
```

Esse trecho, ainda que básico, procura na tabela de usuários a primeira linha que tenha o valor Linus na coluna de nome e retorna os resultados. Para fazê-lo, o Rails utiliza sua camada de abstração do banco de dados integrada, a Active Record, uma biblioteca poderosa; desnecessário dizer, isso é somente uma pequena amostra do que pode ser feito.

Os capítulos 4 e 5 apresentam a Active Record detalhadamente e também o que pode ser esperado dela. Por ora, o mais importante é lembrar que models representam dados. Todas as regras para acesso, associações, validações, cálculos e rotinas de dados, que devem ser executadas antes e depois de operações para salvar, atualizar ou destruir dados, estão devidamente encapsuladas no model. O mundo de sua aplicação é pre-enchido com objetos Active Record: individuais, em listas, novos e velhos. Da mesma forma, a Active Record permite que você utilize construções da linguagem Ruby para manipular todos esses objetos, o que significa que você pode trabalhar com somente uma linguagem em toda a sua aplicação.

Controllers (Controladores)

Agora, vamos temporariamente reordenar as iniciais MVC, colocando o C antes do V. Como você verá a seguir, no Rails, controllers são responsáveis pela renderização de views, por isso faz sentido introduzi-los primeiro.

Controllers são os condutores de uma aplicação MVC. No Rails, são eles que aceitam requisições do mundo exterior, realizam o processamento necessário e então passam o controle para a camada de view, exibindo os resultados. É trabalho do controller lidar com requisições web, como variáveis do servidor de processamento e dados de formulário, pedindo informações ao model e reenviando informações a serem salvas no banco de dados. Podemos estar simplificando demais, mas controllers geralmente executam uma requisição do usuário para criar (*create*), ler (*read*), atualizar (*update*) ou excluir (*delete*) um objeto de model. Essas palavras são vistas com frequência no contexto do Rails, geralmente abreviadas como CRUD. Em resposta a uma requisição, o controller tipicamente executa uma operação CRUD no model, define variáveis para serem utilizadas na view, e então avança para desenhar a interface ou redirecionar outra ação, depois que o processamento estiver completo.

Controllers tipicamente gerenciam uma única área da aplicação. Por exemplo, em uma aplicação de receitas (*recipes*), você provavelmente terá um controller apenas para gerenciá-las. Dentro dele, você pode definir o que chamamos de *ações*, que descrevem o que um controller pode fazer. Se deseja ser capaz de criar, ler, atualizar e excluir receitas, crie ações com nomes adequados no controller das receitas. Um simples controller desse tipo poderia ter a seguinte aparência:

```
class RecipesController < ApplicationController
    def index
        # lógica para listar todas receitas
    end

    def show
        # lógica para exibir uma receita específica
    end

    def create
        # lógica para criar nova receita
    end

    def update
        # lógica para atualizar receita existente
    end

    def destroy
        # lógica para apagar uma receita específica
    end
end
```

É evidente que se você deseja que este controller faça algo, deve colocar instruções em cada ação. Quando uma requisição vem para seu controller, ela usa um parâmetro URL para identificar a ação a ser executada; quando termina, envia uma resposta ao navegador. A resposta é o que veremos a seguir.

Views (Visões)

A camada view, no MVC, forma a parte visível da aplicação. No Rails, views são *templates* que (na maioria das vezes) contêm a marcação HTML a ser renderizada em um navegador. É importante notar que views devem ser livres de toda lógica de programação (salvo a mais básica). Qualquer interação direta com a camada do model deve ser delegada para a camada do controller, mantendo a view livre e desacoplada da lógica de negócio da aplicação.

Geralmente, views têm a responsabilidade de formatar e apresentar objetos do model para saída na tela, além de fornecerem os formulários e as caixas de entrada que aceitam os dados do model, como uma caixa de login com um nome de usuário e senha, ou um formulário de registro. O Rails também oferece a conveniência de um conjunto extenso de *helpers* (auxiliares) que facilitam a conexão dos models e das views, sendo capazes de preencher previamente um formulário com informações do banco de dados, ou de exibir mensagens de erro caso um registro falhe nas regras de validação, como pela falta do preenchimento de campos obrigatórios.

Se você frequentar ambientes que tratam do Rails, certamente ouvirá que algumas pessoas consideram que a interface *é* o software. Concordamos com elas. Como a interface é tudo que o usuário vê, é a parte mais importante. Independentemente do que o software esteja fazendo nos bastidores, as únicas partes às quais o usuário final tem acesso são as que ele pode ver e com as quais pode interagir. O padrão MVC auxilia ao manter a lógica de programação fora da view. Com essa estratégia, programadores são capazes de lidar somente com o código, e designers, com o HTML. Ter um ambiente claro para o design do HTML significa melhores interfaces e melhor software.

Bibliotecas que compõem o Rails

O Rails é uma coleção de bibliotecas, cada uma com uma tarefa especializada. Reunidas, essas bibliotecas individuais formam o framework Rails. Das diversas bibliotecas que compõem o Rails, três mapeiam diretamente ao padrão MVC:

- **Active Record**: biblioteca que manipula a abstração e interação do banco de dados.
- **Action View**: sistema de *templating* que gera os documentos HTML que o visitante recebe como resultado de uma requisição para uma aplicação Rails.
- **Action Controller**: biblioteca para manipulação tanto do fluxo da aplicação, quanto dos dados vindos do banco de dados para serem exibidos em uma view.

Essas bibliotecas podem ser utilizadas de modo independente. Juntas, formam o conjunto de desenvolvimento Model-View-Controller do Rails. Como o Rails é um framework de pilha completa (*full-stack*), todos os componentes estão integrados, e você não precisa ligá-los manualmente.

Rails é modular

Um dos melhores recursos do Rails 3 é o fato de que ele foi construído tendo em mente, desde o início, a modularidade. Ainda que muitos desenvolvedores gostem de receber um framework de pilha completa, talvez você tenha suas próprias preferências em termos de bibliotecas, seja para acesso ao banco de dados, manipulação de templates, ou bibliotecas JavaScript. Conforme descrevemos os recursos do Rails, mencionamos alternativas às bibliotecas padrão. Você pode preferi-las, à medida que se familiariza com o funcionamento interno do Rails.

Rails não é uma solução milagrosa

Não há dúvida de que o Rails oferece aos desenvolvedores web muitos benefícios. Depois de utilizá-lo, é difícil imaginar o desenvolvimento web sem ele. Felizmente, ao que tudo indica, o Rails veio para ficar, por isso você não deve se preocupar. Mesmo assim, esses pontos positivos levantam uma importante questão.

Por mais que destaquemos os benefícios trazidos pelo Rails, é importante que você perceba que não há soluções milagrosas no ramo de design de software. Não importa quão bom ele seja, o Rails nunca será perfeito, nem resolverá todos os problemas. Mais importante do que isso, o Rails nunca substituirá o papel do desenvolvedor. Seu propósito é auxiliar os desenvolvedores na realização de seu trabalho. Mesmo sendo impressionante, o Rails é uma mera ferramenta que, quando usada devidamente, pode produzir resultados incríveis. Nossa esperança é que, continuando a ler este livro e aprendendo a utilizar o Rails, você se torne capaz de aproveitar sua força para produzir softwares baseados na web que sejam, ao mesmo tempo, criativos e de alta qualidade.

Resumo

Este capítulo apresentou uma visão geral e introdutória do Rails. Os tópicos trataram da crescente importância das aplicações web, até a história, filosofia, evolução e arquitetura do framework. Você tomou conhecimento dos recursos do Rails que fazem com que ele seja perfeito para o desenvolvimento ágil, estudando os conceitos de menos software, de convenção, em vez de configuração, e de DRY. Por fim, aprendeu o básico sobre o padrão MVC e recebeu uma introdução à forma como o Rails utiliza MVC.

De posse de toda essa informação, podemos dizer que você está pronto para utilizar o Rails. O próximo capítulo apresenta sua instalação. Nela, você irá experimentá-lo e ver por que ele está envolvido em todo esse agito. Muito em breve, você certamente estará utilizando este framework.