RESTful Serviços Web

Leonard Richardson
Sam Ruby

Rio de Janeiro . 2007

RESTful Serviços Web

Produção Editorial: Editora Alta Books
Coordenação Editorial: Fernanda Silveira
Tradução: Eveline Vieira e Patricia Azeredo
Revisão: Mônica Mourão
Revisão Técnica: Eduardo Velasco
Diagramação: Fernanda Silveira

Impresso no Brasil

ALTA BOOKS

Rua Viúva Cláudio, 291 – Jacaré
Rio de Janeiro – RJ. CEP: 20970-031
Tel: 21 3278-8069/ Fax: 3277-1253
site: www.altabooks.com.br
e-mail: altabooks@altabooks.com.br

Para Woot, Moby e Beet.

— Leonard

Para Christopher, Catherine e Carolyn.

— Sam

Sumário

Preâmbulo

O mundo dos serviços web tem caminhado em direção à supernova em uma rota rápida desde que os astronautas-arquitetos descobriram outra unidade de informação, migrando do pragmatismo para o universo empresarial. Mas, felizmente, nem tudo está perdido. Uma renascença da apreciação do HTTP está formando-se e, sob a bandeira do REST, mostra uma alternativa confiável para o que os negociantes da complexidade estão tentando nos empurrar; um conjunto simples de princípios que os desenvolvedores comuns podem usar para conectar as aplicações em um estilo nativo à web.

O livro *Serviços Web REST* mostra como usar esses princípios sem drama, palavras pomposas e milhares de vias indiretas que têm alarmado uma geração de desenvolvedores web – eles pensam que os serviços web são tão difíceis a ponto de se ter de contar com as implementações BigCo para ter algo feito. Todo desenvolvedor que trabalha com a Web precisa ler este livro.

— David Heinemeier Hansson

Prefácio

Um sistema complexo que funciona é, invariavelmente, considerado como evoluído a partir de um sistema simples que funcionava.

— John Gall

Systemantics

Escrevemos este livro para mostrar uma nova tecnologia surpreendente. Está aqui, é moderna e promete mudar radicalmente o modo como escrevemos os sistemas distribuídos. Estamos falando sobre a World Wide Web.

Tudo bem, não é uma tecnologia nova. Não é tão moderna quanto costumava ser e, de um ponto de vista técnico, não é incrivelmente surpreendente. Mas todo o resto é verdade. Em 10 anos, a Web mudou o modo como vivemos, mas há mais mudança a fazer. A Web é uma plataforma simples, onipresente, porém negligenciada. O objetivo deste livro é disseminar essa mudança para o mundo.

Pode parecer estranho afirmar que o potencial da Web para a programação distribuída foi negligenciado. Afinal, este livro luta pelo espaço nas prateleiras com quaisquer outros livros sobre os serviços web. O problema é que a maioria dos "serviços web" de hoje não tem nenhuma relação com a Web. Em oposição à simplicidade da Web, eles adotam uma arquitetura pesada para o acesso distribuído dos objetos, parecido com COM ou CORBA. As arquiteturas dos "serviços web" de hoje reinventam ou ignoram cada recurso que tornam a Web bem-sucedida.

Não precisa ser assim. Sabemos que as tecnologias contidas na Web podem conduzir serviços remotos úteis, que existem e são utilizados todos os dias. Sabemos que eles podem ser dimensionados para serem enormes, porque já são. Considere o mecanismo de pesquisa Google. O que ele é a não ser um serviço remoto para consultar um banco de dados poderoso e obter uma resposta formatada? Normalmente, não consideramos os sites web como "serviços" porque isso é uma conversa de programação e o cliente final de um site web é um ser humano, mas os serviços, de fato, serviços.

Toda aplicação web, todo site web, é um serviço. Se trabalhar com a Web, em vez de contra ela, e não se perder em camadas de abstração, você poderá aproveitar esta capacidade para as aplicações programáveis. É hora de colocar a "web" de volta nos "serviços web".

Os recursos que facilitam o uso de um site web para um internauta também facilitam o uso da API do serviço web para um programa. Para encontrar os princípios de desenvolvimento destes serviços, podemos traduzir os princípios dos sites web inteligíveis pelos humanos para princípios inteligíveis para os programas de computador.

É isso que fazemos neste livro. Nosso objetivo, do começo ao fim, é mostrar a capacidade (e, onde forem apropriados, os limites) das tecnologias básicas da web: o protocolo da aplicação HTTP, o padrão de nomenclatura URI e a linguagem de marcação XML. Nosso assunto é o conjunto de princípios sob a Web: a Transferência de Estado Representacional ou REST. Pela primeira vez, registramos as melhores práticas para os serviços web "REST". Retiramos a confusão e a adivinhação, substituindo o folclore e o conhecimento implícito por uma recomendação concreta.

Introduzimos a Arquitetura Orientada a Recursos (Resource-Oriented Architecture - ROA), um conjunto criterioso de regras para desenvolver os serviços web REST. Também mostramos a visão do lado do cliente: como você pode

escrever programas para usar os serviços REST. Nossos exemplos incluem serviços REST reais como, por exemplo, o Simple Storage Service (S3) do Amazon, as diversas variações do Atom Publishing Protocol e os Google Maps. Também pegamos os serviços populares com um REST insuficiente, como a API do marcador de páginas social del.icio.us, e restauramo-nos.

A web é simples

Como somos obcecados pela Web, não pensamos que ela pode fazer tudo? Talvez estejamos iludidos, vítimas da euforia. A web é, certamente, a parte mais excitante da Internet, apesar do HTTP não ser seu protocolo mais popular. Dependendo do ponto de vista, grande parte do tráfego da Internet do mundo vem do e-mail (graças ao spam) ou do BitTorrent (graças à violação dos direitos autorais). Se a Internet fosse desaparecer amanhã, o e-mail seria a aplicação que as pessoas mais sentiriam falta. Portanto, por que Web? O que torna o HTTP, um protocolo designado a carregar notas de projetos em um laboratório de física, também adequado para as aplicações Internet distribuídas?

Na verdade, dizer que o HTTP foi designado para *nada* é elogiar-lhe enormemente. O HTTP e a HTML têm sido chamados de "almofada divertida e campainha da alegria dos protocolos Internet, apenas compreensíveis como brincadeiras práticas elaboradas" - e isto por alguém que *goste* deles*. A primeira versão do HTTP certamente parecia uma brincadeira. Eis uma interação de amostra entre o cliente e o servidor:

Solicitação do Cliente	Resposta do Servidor
`GET/hello.text`	`Hello, world!`

É tudo. Você conectou o servidor, forneceu-lhe o caminho para um documento e, então, o servidor envia-lhe o conteúdo deste documento. Você poderia fazer pouco mais com o HTTP 0.9. Parecia uma imitação barata sem recursos dos protocolos de transferência de arquivos mais sofisticados, como o FTP.

Isto é, surpreendentemente, uma parte maior da resposta. Com um pouco de exagero, podemos dizer que o HTTP é, exclusivamente, adequado para as aplicações Internet distribuídas porque não tem nenhum recurso a mencionar. Você informa-o sobre o que deseja e ele retorna-lhe. Comparado a um filme de kung fu +, a fraqueza do HTTP é sua força; sua simplicidade, seu poder.

Nessa primeira versão do HTTP, habilmente disfarçada como uma linguagem sem recursos, há o *endereçamento* e a *falta de estado*: as duas decisões básicas de desenvolvimento que tornaram o HTTP melhor que seus concorrentes e o mantiveram dimensionável até os mega sites atuais. Muitos dos recursos ausentes no HTTP 0.9 acabaram sendo desnecessários ou improdutivos. Adicioná-los de volta realmente enfraquece a Web. A grande parte restante foi implementada nas revisões 1.0 e 1.1 do protocolo. As outras duas tecnologias essenciais para o sucesso da Web, URIs e HTML (e, posteriormente, XML), também são simples em aspectos importantes.

Obviamente, estas tecnologias "simples" são poderosas o bastante para fornecerem-nos a Web e as aplicações que usamos nela. Neste livro, vamos além e afirmamos que a World Wide Web é um ambiente simples e flexível para a *programação* distribuída. Também afirmamos saber a razão disso: não há nenhuma diferença essencial entre a web humana, feita para nosso próprio uso, e a "web programável", feita para o consumo feito pelos programas de software. Dizemos: se a Web é boa o bastante para os seres humanos, é boa o bastante para os robôs. Apenas precisamos fazer algumas concessões. Os programas de computador são bons ao desenvolverem e analisarem estruturas de dados complexas, mas não são tão flexíveis quanto os humanos ao interpretarem os documentos.

Os grandes serviços web não são simples

Há vários protocolos e padrões, a maioria baseada no HTTP, designados para desenvolver os Serviços Web (observe as letras maiúsculas). Estes padrões juntos são chamados de pilha WS-*. Eles incluem a WS-Notification, a WS-

Security, a WSDL e o SOAP. Neste livro, damos o nome "Big Web Services" a essa coleção de tecnologias como um termo suavemente depreciativo.

Este livro não cobre esses padrões com grandes detalhes. Acreditamos que você pode implementar os serviços web sem implementar os Big Web Services: a Web deve ser todo o serviço necessário. Acreditamos que as tecnologias básicas da Web são boas o bastante para serem consideradas a plataforma padrão para os serviços distribuídos.

Alguns padrões WS-* (como o SOAP) podem ser usados de maneiras compatíveis com o REST e nossa Resource-Oriented Architecture. Na prática, entretanto, eles podem ser usados para implementar as aplicações Remote Procedure Call no HTTP. Algumas vezes, um estilo RPC é adequado e outras, outras necessidades têm prioridade sobre as virtudes da Web. Isso é ótimo.

O que não gostamos é da complexidade desnecessária. Com muita freqüência, um programador ou uma empresa produz os Big Web Services para um serviço com o qual o velho HTTP comum poderia lidar muito bem. O efeito é que o HTTP é reduzido a um protocolo de transporte para um peso XML enorme que explica o que "realmente" está ocorrendo. O serviço resultante é muito complexo, impossível de depurar e não funcionará, a menos que seus clientes tenham a configuração exatamente igual à sua.

Os Big Web Services têm uma vantagem: as ferramentas modernas podem criar um serviço web a partir de seu código com um único clique, especialmente se você estiver desenvolvendo no Java ou no C#. Se estiver usando essas ferramentas para gerar serviços web RPC com a pilha WS-*, provavelmente não importará para você se um serviço web REST for muito mais simples. As ferramentas ocultam toda a complexidade, portanto quem se importa? A largura de banda e a CPU são baratas.

Essa atitude funciona quando você está trabalhando em um grupo homogêneo, fornecendo serviços através de um firewall para os outros grupos como o seu. Se seu grupo tiver um poder político suficiente, você poderá fazer com que as pessoas ultrapassem o firewall. Mas se quiser que seu serviço cresça até a escala da Internet, terá que lidar com clientes que nunca pretendeu, usando pilhas de softwares desenvolvidos e personalizados para fazer coisas a seu serviço, que nunca imaginou que fossem possíveis. Seus usuários desejarão integrar seu serviço em outros serviços sobre os quais nunca ouviu falar. Parece difícil? Isso já acontece na Web atual.

As abstrações nunca são perfeitas. Toda camada nova cria pontos de falha, confusões de interoperabilidade e problemas de dimensionamento. As novas ferramentas podem esconder a complexidade, mas não podem justificá-la - e elas sempre aumentam. Fazer com que um serviço funcione com a Web como um todo significa prestar atenção na adaptabilidade, dimensionamento e sustentabilidade. A simplicidade - a virtude menosprezada do HTTP 0.9 - é um pré-requisito de todos os três. Quanto mais complexo for o sistema, mais difícil será corrigir quando algo der errado.

Se você fornecer os serviços REST, poderá gastar sua complexidade em recursos adicionais ou promover a interação entre diversos serviços. O sucesso em fornecer serviços também significa fazer parte da Web, ao invés de apenas estar "na" Web: tornando suas informações disponíveis sob as mesmas regras que governam os sites web bem desenvolvidos. Quando mais perto você estiver dos protocolos web básicos, mais fácil isto será.

A história do REST

O REST é simples, mas é bem definido e não é uma desculpa para implementar os serviços web como sites web imbecis porque "eles são iguais". Infelizmente, até agora a principal referência REST foi o capítulo cinco na tese de doutorado de Roy Fielding em 2000. Apesar de ser uma boa leitura para uma tese de doutorado, deixa a maior parte das questões reais sem resposta [++]. É porque apresenta o REST não como uma arquitetura, mas como um modo de julgar as arquiteturas. O termo "REST" é como o termo "orientado a objetos". Uma linguagem, uma estrutura ou uma aplicação pode ser designada de uma maneira orientada a objetos, mas isso não torna sua arquitetura *a* arquitetura orientada a objetos.

Até nas linguagens orientadas a objetos como o C++ e o Ruby, é possível escrever programas que não sejam verdadeiramente orientados a objetos. O http, geralmente, funciona muito bem nos critérios do REST. (Deveria, uma vez que Fielding co-escreveu o padrão HTTP e escreveu sua tese para descrever a arquitetura da Web). Mas os sites web reais, as aplicações web e os serviços web, geralmente, revelam os princípios do REST. Como você pode assegurar que está aplicando corretamente os princípios no desenvolvimento de um serviço web específico?

A maioria das outras fontes de informações sobre o REST é informal: listas de correspondências, wikis e registros da web (listo alguns dos melhores no Apêndice A). Até o momento, as melhores práticas do REST têm sido uma questão de folclore. É preciso uma arquitetura concreta baseada na meta-arquitetura REST: um conjunto de regras simples para implementar os serviços típicos que satisfazem o potencial da Web. Apresentamos tal arquitetura neste livro como a Resource-Oriented Architecture (veja capítulo 4). Certamente, não é a única arquitetura REST de alto nível possível, mas consideramo-na boa para desenvolver os serviços web de uso fácil para os clientes.

Escrevemos a ROA para tornarem-se reais as melhores práticas do desenvolvimento do serviço web. O que escrevemos é uma base sugerida. Se antes você já tentou descobrir o REST, esperamos que nossa arquitetura lhe assegure que esteja fazendo "realmente" o REST. Também esperamos que a ROA ajude a comunidade como um todo a progredir mais rápido ao sugerir e codificar melhores práticas. Desejamos facilitar para os programadores a criação de aplicações web distribuídas elegantes, que façam o trabalho para o qual foram designadas e que participem da Web, em vez de apenas residirem nela.

Porém, sabemos que não é suficiente ter todos estes fatos técnicos à sua disposição. Trabalhamos em organizações onde as decisões arquiteturais maiores não são adotadas. Você não poderá ter sucesso com uma arquitetura REST se nunca teve uma chance de usá-la. Além do conhecimento técnico, temos que lhe fornecer vocabulário para argumentar sobre as soluções REST. Posicionamos a ROA como uma alternativa simples para a arquitetura do tipo RPC usada pelos serviços SOAP+WSDL de hoje. A arquitetura RPC exibe os *algoritmos* internos através de uma interface de linguagem de programação complexa que é diferente para cada serviço. A ROA exibe os *dados* internos através de uma interface de processamento de documentos simples que é sempre igual. No capítulo 10, compararemos as duas arquiteturas e mostraremos como argumentar sobre a ROA.

Como reunir as webs

Os programadores vêm usando os sites web como serviços web por anos - não oficialmente, claro[§]. É difícil para um computador compreender as páginas web designadas para o consumo humano, mas isso nunca impediu que os hackers recuperassem as páginas com clientes automáticos e fragmentassem a tela com bits interessantes. Com o tempo, essa atividade foi purificada nas tecnologias amistosas do programador para exibir a funcionalidade de um site web de maneiras oficialmente aprovadas - RSS, XML-RPC e SOAP. Essas tecnologias formaram uma web programável, uma que estendeu a web humana para a conveniência com os programas de software.

Nosso objetivo fundamental neste livro é reunir a web programável à web humana. Prevemos uma única rede interconectada: uma World Wide Web que é executada em um conjunto de servidores, usa um conjunto de protocolos e obedece a um conjunto de princípios de desenvolvimento. Uma rede que você pode usar se estiver fornecendo dados para os seres humanos ou programas de computador.

A Internet e a Web não tinham que existir. Elas chegaram até nós como uma cortesia da defesa contra o dinheiro mal distribuído, de projetos de engenharia desprezíveis, de práticas de engenharia do tipo pior-é-melhor, da grande ciência, do idealismo liberal ingênuo, da política libertária excêntrica, do tecno-fetichismo, do trabalho duro e do capital de programadores e investidores que pensaram que encontrariam uma maneira fácil de enriquecer rapidamente.

O resultado é, surpreendentemente, uma plataforma simples, aberta (agora) e quase universal para as aplicações em rede. Esta plataforma abrange muito conhecimento humano e suporta a maioria das áreas de trabalho humano. Achamos que é hora de começar a aplicar seriamente suas regras na programação distribuída, de abrir essas informações e os processos para os clientes automáticos. Se você concorda, este livro mostrará como fazê-lo.

O que há neste livro?

Neste livro, concentramo-nos nas questões práticas: Como desenvolver e implementar os serviços web REST e os clientes para estes serviços. Nosso foco secundário está na teoria: o que significa ser REST e porque os serviços web devem ser mais REST, em vez de menos. Não discutiremos sobre tudo. Tentamos abordar os tópicos mais usuais atualmente e, como é o primeiro livro do tipo, voltamo-nos para a questão básica: como desenvolver um serviço REST.

Os três primeiros capítulos introduzem os serviços web da perspectiva do cliente e mostram o que há de especial nos serviços REST.

Capítulo 1, Web programável e seus habitantes

> Neste capítulo, apresentamos os serviços web em geral: os programas que ultrapassam a Web e pedem que um servidor externo forneça dados ou execute um algoritmo. Demonstramos as três arquiteturas do serviço web comuns: REST, estilo RPC e o híbrido REST-RPC. Mostramos as solicitações e as respostas HTTP de exemplo para cada arquitetura, junto com um código típico do cliente.

Capítulo 2, Como escrever clientes do serviço web

> Neste capítulo, mostramos como escrever os clientes para os serviços web existentes, usando uma biblioteca HTTP e um divisor gramatical XML. Apresentamos um serviço REST-RPC popular (o serviço web para o site de marcação de páginas del.icio.us social) e demonstramos os clientes escritos no Ruby, Python, Java, C# e PHP. Também damos recomendações da tecnologia para várias outras linguagens, sem de fato mostrar o código. O JavaScript e o Ajax são tratados separadamente no capítulo 11.

Capítulo 3, O que torna os serviços REST diferentes?

> Pegamos as lições no capítulo 2 e as aplicamos em um serviço REST puramente: o Simple Storage Service (S3) do Amazon. Enquanto desenvolvemos um cliente S3, mostramos alguns princípios importantes do REST: recursos, representações e interface uniforme.

Os próximos seis capítulos formam o centro do livro. Eles concentram-se no desenvolvimento e na implementação de seus próprios serviços REST.

Capítulo 4, Resource-Oriented Architecture

> Uma apresentação formal do REST; não em sua forma abstrata, mas no contexto de uma arquitetura específica para os serviços web. Nossa arquitetura é baseada em quatro conceitos REST importantes: recursos, seus nomes, suas representações e os links entre eles. Seus serviços devem ser julgados segundo quatro propriedades REST: endereçamento, falta de estado, encadeamento e interface uniforme.

Capítulo 5, Como desenvolver serviços orientados a recursos de somente leitura

> Apresentamos um procedimento para transformar uma idéia ou um conjunto de exigências em um conjunto de recursos REST. Estes recursos são somente de leitura: os clientes podem obter os dados em seu serviço, mas não podem enviar nenhum dado próprio. Mostramos o procedimento desenvolvendo um serviço web para fornecer mapas navegáveis, inspirados pela aplicação web Google Maps.

Capítulo 6, Como desenvolver serviços orientados a recursos de leitura/gravação

> Estendemos o procedimento do capítulo anterior para que os clientes possam criar, modificar e apagar os recursos. Demonstramos adicionando dois tipos novos de recurso ao serviço de mapa: contas do usuário e locais definidos pelo usuário.

Capítulo 7, Implementação do serviço

Remodelamos um serviço do tipo RPC (o híbrido REST-RPC del.icio.us para o qual escrevemos os clientes no capítulo 2) como um serviço puramente REST. Então, implementamos este serviço como um Ruby na aplicação Rails. Diversão para a família inteira!

Capitulo 8, Melhores práticas do REST e da ROA

Neste capítulo, reunimos nossas sugestões anteriores para o desenvolvimento do serviço em um local e adicionamos novas sugestões. Mostramos como os recursos padrões do HTTP podem ajudar nos problemas e otimizações comuns. Também fornecemos desenvolvimentos orientados a recursos para recursos difíceis como, por exemplo, as transações, que você poderia julgar impossíveis de fazer-se nos serviços web REST.

Capítulo 9, Blocos de desenvolvimento dos serviços

Aqui, descrevemos as tecnologias extras que trabalham sobre as três grandes linguagens do REST: HTTP, URI e XML. Algumas delas são formatos de arquivo para transmitir o estado, como, por exemplo, a XHTML e seus microformatos. Alguns são formatos de hipermídia para mostrar aos clientes os aproveitamentos do estado, como o WADL. Outros são conjuntos de regras para desenvolver os serviços web REST, como o Atom Publishing Protocol.

Os três últimos capítulos cobrem os tópicos especializados, cada um podendo ter um livro próprio agora:

Capítulo 10, Resource-Oriented Architecture versus Big Web Services

Comparamos nossa arquitetura e o REST em geral com outra marca líder. Achamos que os serviços web REST são mais simples, mais dimensionáveis e mais fáceis de usar-se, mais bem sintonizados com a filosofia da Web e mais capazes de lidar com uma grande variedade de clientes do que os serviços baseados no SOAP, WSDL e pilha WS-*.

Capítulo 11, Aplicações Ajax como clientes REST

Aqui, explicamos a arquitetura Ajax para as aplicações web em termos de serviços web: uma aplicação Ajax é apenas um cliente do serviço web que é executado dentro de seu navegador web. Isso torna esse capítulo uma extensão do capítulo 2. Mostramos como escrever os clientes para os servidores web REST usando o XMLHttpResquest e a biblioteca JavaScript padrão.

Capítulo 12 Estruturas para os serviços REST

No capítulo final, cobrimos três estruturas populares que facilitam a implementação dos serviços web REST: Ruby on Rails, Restlet (para o Java) e Django (para o Python).

Também temos três apêndices que esperamos lhe serem úteis:

Apêndice A, Alguns recursos para o REST e alguns recursos REST

A primeira parte lista padrões interessantes, tutoriais e comunidades relacionadas aos serviços web REST. A segunda parte lista alguns serviços web REST públicos que você pode usar e aprender.

Apêndice B, Os 42 principais códigos de resposta HTTP

Descreve todo código de resposta HTTP padrão (mais uma extensão) e explica quando você usaria um em um serviço web REST.

Faz o mesmo nos cabeçalhos HTTP. Cobre cada cabeçalho HTTP padrão e alguns cabeçalhos de extensão que são úteis para os serviços web.

Quais partes você deve ler?

Organizamos este livro para o leitor que está interessado nos serviços web em geral: alguém que aprende fazendo, mas que não tem muita experiência com os serviços web. Se isso descreve você, o caminho mais simples neste livro é o melhor. Você pode começar no início, ler até o capítulo 9 e então, prosseguir em diante indo para onde estiver interessado.

Se tiver mais experiência, poderá tomar um caminho diferente no livro. Se estiver preocupado apenas em escrever os clientes para os serviços existentes, provavelmente irá concentrar-se nos capítulos 1, 2, 3 e 11 – as seções sobre a do serviço não serão muito úteis para você. Se quiser criar seu próprio serviço web ou se estiver tentando descobrir o que REST realmente significa, poderá começar a ler a partir do capítulo 3. Se quiser comparar as tecnologias REST e WS-*, poderá começar a ler os capítulos 1, 3, 4 e 10.

Notas administrativas

Este livro tem dois autores (Leonard e Sam), mas no resto do livro mesclaremos nossas identidades em uma única autoria: "eu". No capítulo final (capítulo 12), o autor "eu" fica um pouco inflado, pois os desenvolvedores Django e Restlet reúnem-se para mostrar como suas estruturas permitem desenvolver os serviços REST.

Supomos que você seja um programador competente, mas não que tenha nenhuma experiência com a programação web em particular. O que dizemos neste livro não está ligado a nenhuma linguagem de programação e incluímos um código de amostra para os clientes e serviços REST em várias linguagens. Mas sempre que não estivermos demonstrando uma estrutura ou linguagem específica, usamos o Ruby (*http://www.ruby-lang.org/*) como nossa linguagem de implementação.

Escolhemos o Ruby porque é conciso e fácil de ler, mesmo para os programadores que não conhecem a linguagem. (E porque é bonito e confunde-se com o último nome de Sam.) A estrutura web padrão do Ruby, Ruby on Rails, também é uma das principais plataformas de implementação para os serviços web REST. Se você não conhecer o Ruby, não se preocupe: incluímos muitos comentários explicando seus idiomas específicos.

Os programas de exemplo neste livro estão disponíveis para o download no site web oficial deste livro (*http://www.oreilly.com/catalog/9780596529260*). Isso inclui a aplicação Rails inteira do capítulo 7 e as aplicações Restlet e Django correspondentes do capítulo 12. Também inclui as implementações Java de muitos clientes que apenas aparecem no livro como implementações Ruby. Esses programas clientes usam a biblioteca Restlet e foram escritos pelos desenvolvedores Restlet Jerome Louvel e Dave Pawson. Se você estiver mais familiarizado com o Java do que com o Ruby, essas implementações poderão ajudá-lo a compreender os conceitos sob o código. Em particular, há uma implementação Java completa do cliente S3 Amazon no capítulo 3 deste livro.

Convenções usadas neste livro

As seguintes convenções tipográficas são usadas neste livro:

Itálico

Indica termos novos, URLs, endereços de e-mail, nomes de arquivo e extensões de arquivo.

`Fonte Courier`

Usada para as listagens do programa, assim como dentro dos parágrafos para referir-se aos elementos do

programa, como: nomes da variável ou da função, bancos de dados, tipos de dados, variáveis de ambiente, instruções e palavras-chave.

`Fonte Courier Negrito`
Mostra os comandos ou outro texto que deve ser digitado literalmente pelo usuário.

`Fonte Courier Itálico`
Mostra o texto que deve ser substituído pelos valores fornecidos pelo usuário ou pelos valores determinados pelo contexto.

Este ícone significa uma dica, sugestão ou nota geral.

Este ícone indica uma advertência ou aviso.

Neste livro escolhemos mater o termo "Web Services" em português, "Serviços Web", porém, as duas formas, tanto em inglês como em português são utilizadas em nosso idioma.

Como usar os exemplos de código

Este livro está aqui para ajudá-lo a ter seu serviço pronto. Em geral, você pode usar o código neste livro em seus programas e documentação. Não precisa nos contatar para obter permissão, a menos que esteja reproduzindo uma parte significativa do código. Por exemplo, escrever um programa que use várias partes grandes de código deste livro não requer permissão. Vender ou distribuir um CD-ROM de exemplos dos livros requer permissão. Responder a uma pergunta mencionando este livro e citando o código de exemplo não requer permissão. Incorporar uma quantidade significativa do código de exemplo deste livro na documentação de seu produto requer permissão.

Apreciamos, mas não exigimos, uma atribuição, que, geralmente, inclui título, autor, editora e ISBN. Por exemplo: "*Serviços Web REST* por Leonard Richardson e Sam Ruby. Direitos reservados 2007 O'Reilly Media, Inc., 978-0-596-52926-0".

Se você achar que seu uso dos exemplos de código ficam fora do uso justo ou da permissão dada acima, sinta-se à vontade para nos contactar em *altabooks@altabooks.com.br*.

Como nos contactar

Enderece comentários e perguntas sobre este livro para a editora:

Editora Alta Books

Rua Viúva Cláudio, 291 - Jacarezinho

Rio de Janeiro - RJ CEP 20970-031

Telefones: (21) 32789-8069/8419 Fax: (21) 3277-1253

Temos uma página web para este livro, na qual listamos os erros, exemplos e qualquer informação adicional. Você pode acessar essa página em:

http://www.altabooks.com.br

Para comentar ou fazer perguntas sobre este livro, envie um e-mail para:

altabooks@altabooks.com.br

Agradecimentos

Estamos especialmente agradecidos às pessoas cujo trabalho mostrou-nos que podíamos programar diretamente com o HTTP. Para Sam, foi Rael Dornfest com seu blog Blosxom. A experiência de Leonard origina-se do desenvolvimento de aplicações de fragmentação de telas em meados dos anos 90. Seus agradecimentos vão para aqueles cuja web tornaram seus sites úteis como serviços web; a saber, o autor pseudônimo da história em quadrinhos on-line "Pokey the Penguin".

Compreendemos que Roy Fielding estava lá para fornecer mais informações. Sua opinião, seu trabalho, foi importante para delimitar, sustentar, possibilitar algo que para nós era apenas um pensamento, uma suposição. É sobre a base teórica de Roy que tentamos nos fundamentar.

Ao escrever este livro, tivemos uma grande ajuda da comunidade REST. Estamos agradecidos pelo retorno que tivemos de Benjamin Carlyle, David Gourley, Joe Gregorio, Marc Hadley, Chuck Hinson, Pete Lacey, Larray Liberto, Benjamin Pollack, Aron Roberts, Richard Walker e Yohei Yamamoto. As outras pessoas ajudaram, sem saber, através de seus trabalhos escritos: Mark Baker, Tim Berners-Lee, Alex Bunardzic, Duncan Cragg, David Heinemeier Hansson, Ian Hickson, Mark Nottingham, Koranteng Ofosu-Amaah, Uche Ogbuji, Mark Pilgrim, Paul Prescod, Clay Shirky, Brian Totty e Jon Udell. Naturalmente, todas as opiniões neste livro, quaisquer erros e omissões são nossos.

Nosso editor Michael Loukides foi útil e inteligente no desenvolvimento deste livro. Também gostaríamos de agradecer a Laurel Rume e a todos os outros na O'Reilly por sua produção.

Finalmente, Jerome Louvel, Dave Pawson e Jacob Kaplan-Moss merecem agradecimentos especiais. Seu conhecimento sobre o Restlet e o Django tornou o capítulo 12 possível.