RESTful Web APIs
可因时而变的服务

RESTful Web APIs

中文版

[美] *Leonard Richardson & Mike Amundsen* 著
Sam Ruby 序
赵震一 李哲 译

O'REILLY®
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

RESTful Web APIs中文版

RESTful Web APIs

[美] Leonard Richardson & Mike Amundsen 著

赵震一 李哲 译

電子工業出版社

Publishing House of Electronics Industry

北京 • BEIJING

内 容 简 介

本书是针对RESTful API的实用指南，通过展示各种用来创建高可用应用的强大工具，讲解REST的深层原理，以及介绍基于超媒体API的策略，使读者得以在将上述内容融会贯通后，设计出让客户高度满意的RESTful的web API。本书极具权威性与前瞻性，既代表了API领域的最前沿趋势，也覆盖了API领域的最重要实践。

本书适合所有从事Web开发和架构工作的读者阅读参考。

版权贸易合同登记号 图字：01-2014-2675

图书在版编目（CIP）数据

RESTful Web APIs 中文版 /（美）理查德森（Richardson,L.），（美）阿蒙森（Amundsen,M.）著；赵震一，李哲译 . —北京 ：电子工业出版社，2014.6
书名原文 ：RESTful Web APIs
ISBN 978-7-121-23115-5

Ⅰ . ① R… Ⅱ . ①理… ②阿… ③赵… ④李… Ⅲ . ①互联网络－网络服务器－程序设计 Ⅳ . ① TP368.5

中国版本图书馆 CIP 数据核字 (2014) 第 087092 号

策划编辑：张春雨
责任编辑：张春雨
印　　刷：北京丰源印刷厂
装　　订：三河市鹏成印业有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路173信箱　邮编：100036
开　　本：787×980　1/16　印张：26　字数：540.8千字
版　　次：2014年6月第1版
印　　次：2014年6月第1次印刷
定　　价：79.00元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至zlts@phei.com.cn，盗版侵权举报请发邮件至dbqq@phei.com.cn。

服务热线：（010）88258888。

O'Reilly Media, Inc.介绍

O'Reilly Media通过图书、杂志、在线服务、调查研究和会议等方式传播创新知识。自1978年开始,O'Reilly一直都是前沿发展的见证者和推动者。超级极客们正在开创着未来,而我们关注真正重要的技术趋势——通过放大那些“细微的信号”来刺激社会对新科技的应用。作为技术社区中活跃的参与者,O'Reilly的发展充满了对创新的倡导、创造和发扬光大。

O'Reilly为软件开发人员带来革命性的“动物书”;创建第一个商业网站(GNN);组织了影响深远的开放源代码峰会,以至于开源软件运动以此命名;创立了Make杂志,从而成为DIY革命的主要先锋;公司一如既往地通过多种形式缔结信息与人的纽带。O'Reilly的会议和峰会聚集了众多超级极客和高瞻远瞩的商业领袖,共同描绘出开创新产业的革命性思想。作为技术人士获取信息的选择,O'Reilly现在还将先锋专家的知识传递给普通的计算机用户。无论是通过书籍出版,在线服务或者面授课程,每一项O'Reilly的产品都反映了公司不可动摇的理念——信息是激发创新的力量。

业界评论

“O'Reilly Radar博客有口皆碑。”

——Wired

“O'Reilly凭借一系列(真希望当初我也想到了)非凡想法建立了数百万美元的业务。”

——Business 2.0

“O'Reilly Conference是聚集关键思想领袖的绝对典范。”

——CRN

“一本O'Reilly的书就代表一个有用、有前途、需要学习的主题。”

——Irish Times

“Tim是一位特立独行的商人,他不光放眼于最长远、最广阔的视野并且切实地按照Yogi Berra的建议去做了:‘如果你在路上遇到岔路口,走小路(岔路)。’回顾过去Tim似乎每一次都选择了小路,而且有几次都是一闪即逝的机会,尽管大路也不错。”

——Linux Journal

推荐序

“hypermedia as the engine of application state”

上面这段话看起来有些神秘，甚至不能算是一个完整的句子。它究竟是什么意思？它是一段咒语吗？它有什么神奇的魔力？

在我看来，如果把 Web 系统比作是电影《黑客帝国》（The Matrix）里面那座精密宏伟、无与伦比的 Matrix 系统，上面这段话就是那位华人钥匙匠制作的、能够到达 Matirix 系统后台部分的钥匙。REST 之父 Roy Fielding 无疑是设计建造这座 Matrix 系统的主架构师之一，此外还有 Tim-Berners Lee 这样大神级的人物。这座 Matrix 系统的架构师不是一个人，而是一个英雄的团队。

Roy Fielding 博士一贯反对 design by buzzword（按照时髦的词汇来做设计），在其 2000 年的博士论文中，他借用《建筑师讽刺剧》（The Architects Sketch）里面某人所构思的一座到处悬挂着屠宰刀的公寓楼设计，来辛辣地讽刺那些只会 design by buzzword 的人。然而极为讽刺的是，Fielding 在其博士论文中创造出来的“REST”（包括后来出现的“RESTful API”）这个缩写词现在已经成为了 Web 开发社区中最引人注目的一个 buzzword。很多人在尚未真正理解 REST 的一些核心概念的情况下，就到处公然宣称他们所设计的 API 是“RESTful API”。这样的情况太普遍了，以至于 Fielding 本人实在无法忍受，他在 2008 年 10 月写了一篇博客“REST APIs must be hypertext-driven”（RESTful API 必须是超文本驱动的）。这篇博客引起了 Web 开发社区广泛的讨论和反思，后来 Web 开发社区将 Fielding 在其博士论文中关于超媒体作用的论述，特别是本文开头的这句话，简化成了缩写词“HATEOAS”。“超文本驱动”和“HATEOAS”是完全相同的概念，只是表述方式不同，完全可以互换。

没错，HATEOAS 正是 REST 的灵魂，抛弃了 HATEOAS，REST 就失去了灵魂。虽然不支持 HATEOAS 的所谓“RESTful API”在很多场合仍然是很有用的，但是这样的 API 在松耦合和可伸缩性方面会受到一些损失。

Web 开发社区对于设计 RESTful API 最佳实践的探索，有点像 20 世纪初人类征服南极点或者征服更高山峰的竞赛，相关的图书反映出了竞赛的进展。从 2007 年第一本 REST 开发方面的图书《RESTful Web Services》，到今天为止这方面的图书已经不下 30 本。《RESTful Web APIs》这本新书，在对设计 RESTful API 的探索方面可谓是征服了一个新的高度。这本书并不是一本面向初学者的书，因此并没有重复其他 REST 开发入门图书中的一些基础知识的介绍，而是直接站在了前人的肩头（也包括三位作者本人以前写的两本 REST 开发图书《RESTful Web Services》和《Building Hypermedia APIs with HTML5 and Node》）。设计 RESTful API 的一些高级概念，例如对于超媒体的使用（HATEOAS）、各种为超媒体添加语义的技术、媒体类型的选择和设计、设计支持 HATEOAS 的 RESTful API 的流程等，在这本书里面讲的最为清楚。简单来说，这本书最核心的内容，就是如何设计支持 HATEOAS 的 RESTful API。

我在几年前翻译 Roy Fielding 博士论文的过程中，感觉论文虽然非常深刻、精彩，但是也比较抽象。REST 的这些核心概念，在具体的 RESTful API 设计和开发过程中如何实现，仍然是一个巨大的挑战。《RESTful Web APIs》这本书令人满意地解决了这些挑战，它把 REST 的抽象概念，忠实地具像化了，真的是非常棒的工作！

本书的中文版译者赵震一是我的一位朋友，我曾经和震一对于 REST 的一些核心概念做过深入探讨，震一对待技术问题的严谨态度让我印象非常深刻。震一和他的朋友李哲翻译的这本书，阅读起来非常流畅，体验很好，可以想见他们做过大量的推敲和润色。翻译技术图书，是一件辛苦的差事，需要大量的付出。我们在为原著作者的深厚功力而叹服的同时，也应该为好的译者而喝彩！

深入理解和学习 REST，就像是一种禅修的过程，可能需要持续几年时间。在这个过程中，有些时候会有顿悟，但是更多的还是渐悟。《RESTful Web APIs》这本书将会带领我们翻越一座新的山峰，过了那座山，后面还会有什么？

李锟

2014 年 5 月 22 日

对《RESTful Web APIs》一书的赞誉

“本书是学习 API 设计必备技艺的最佳起点。”

——Matt McLarty

API Academy 的联合创始人

“在阅读这本书的全部时间里，我一直在心里咒骂。我之所以这样做是因为在我逐个阅读了每一条解释之后，我不由地开始担心——它们写得实在太棒了，以至于在编写我自己的书时很难再找到一个更好的解释。你将再也无法找到另一部能将这个主题挖掘得如此彻底，解释得如此清晰的书了。请拿好这些工具，去创造一些奇妙的东西吧，并将它分享给世界上的其他人，好吗？”

——Steve Klabnik

《Designing Hypermedia APIs》一书的作者

“超媒体是最不容易被理解的 REST 要义，而本书对超媒体格式的解说精彩彻底。”

——Stefan Tilkov

REST 的布道者、作家及顾问

“超媒体 API 的最佳实用手册。人手必备。”

—— Ruben Verborgh

语义超媒体研究员

献给 Sienna、Dalton 和 Maggie。——Leonard

献给“主管”Milo，不管是在本次还是很多其他的项目中，
你自始至终是我不变和耐心的朋友！——Mike

序

渐进呈现是用户界面设计中的一个概念，它提倡只在用户需要的时候呈现用户所需的信息。从很多方面来讲，你正在阅读的这本书就是一个实践了该原则的实例。而事实上，仅仅是回到七年前，这本书中所提到的内容很可能还无法“工作”。

正如你所见，相比于编写《RESTful Web Services》（本书的前身）之时，编程世界如今已时过境迁。在那个时候，“REST”这个词还是很少有人使用的。即便在有人使用时也往往被误用，人们对该词也存在着广泛的误解。

在 20 世纪 90 年代中后期，REST 所基于的标准 HTTP 和 HTML 已经被开发出来，并成为了 IETF 和 W3C 的标准，大致已接近它们现今的形态。尽管如此，还是存在着上述这样的情况。Roy Fielding 在他 2000 年发布的论文中引入了 REST 一词，而本书便是基于这篇论文所编写的。

Leonard Richardson 和我曾打算纠正这一（对 REST 来说）不公正的现状。为此，我们主要专注于那些支撑 HTTP 的基础概念，并且就如何将这些概念应用到实际应用中提供了一些具有实践意义的指导。

我认为我们的努力起到了松动这一现状根基的作用，并自此引发了对 REST 支持的雪崩效应。REST 迅速有了它自己的生命，并逐渐成为了一个流行的词汇。事实上现在的情况差不多是：只要有新的 web 接口推出，几乎默认都会称它为 REST。在短短的几年里，我们确实走了很长一段路。

不可否认，REST 作为一个词已经被用过了头，而且通常都没有被正确地应用。但是综合所有的考虑，我还是非常欣慰的，资源和 URI 的概念已经成功地渗入到应用接口设计之中。Web 毕竟是一个颇具弹性的地方，这些新的接口尽管不完美，但是比起那些它们所替代的老的接口却有着飞跃式的进步。

但是我们还可以做得更好。

如果将此事比喻成构建一座大厦，那么现在用于构建大厦的材料已经具备，是时候回过头来去重新审视一下整个领域，并基于这些概念来构建我们的大厦。下一步应该做的就是去探究那些通用的媒体类型以及特定的超媒体格式。上一本书几乎完全专注于构建正确的 HTTP 应用，而现在是时候让我们去深入探究下超媒体类型（那些像 HTML 一样没有与单独的应用或者甚至是单个厂商紧密绑定的媒体类型）背后的概念了。

HTML 就是这样一个超媒体格式的典型例子，它在 web 架构中占据着特殊的位置。事实上，我个人的发现之旅已经深入到了 HTML 的 W3C 标准的发展，也就是现在的 HTML5。虽然 HTML 确实在这本新书中占据了一个非常显著的位置，但是超媒体主题还有更多的内容需要讨论。所以尽管我们保持着联系，Leonard 选择了一个能够胜任我原来角色的人——Mike Amundsen 来作为本书的联合作者。

非常高兴能看到本书的编写，而在本书的阅读过程中我学习到了大量我从未在任何其他来源接触过的媒体类型。更重要的是，本书展示了这些类型所具有的共同点，以及如何对它们进行区分，因为它们中的每一个都具有自己的特性。

希望本书所做的一切能起到跟它的前身（《RESTful Web Services》）相同的效果。谁知道在另一个七年内所有的事情会不会有机会再重来一次呢，并且对表述性状态移交（Representational State Transfer）[注1] 中那些仍然被忽视的其他方面做出强调。

——Sam Ruby

注 1　此处 Representational State Transfer 参考了李锟修订后的译法，译作“表述性状态移交”，此前曾被翻译为“表述性状态转移”——译者注。

前言

“大多数软件系统在创建时都有一个隐含的假设：整个系统处在一个实体的控制之下；或者至少参与到系统中的所有实体都向着一个共同目标行动，而不是有着各自不同的目标。当系统在互联网上开放地运行时，无法安全地满足这样的假设。”

——Roy Fielding

Architectural Styles and the Design of Network-based Software Architectures

“Discordia 信徒应该一直使用官方的 Discordian 文档编号系统。”

—— Malaclypse the Younger 和 Lord Omar Khayyam Ravenhurst

Principia Discordia[注1]

我要向你展示一种可以更好地进行分布式计算的方式，它使用了有史以来最成功的分布式系统，即万维网的根本思想。如果你已经决定（或者你的经理已经决定）需要为你的公司发布一个 web API 的话，我希望你能够读一下这本书。不管在你计划中的是一个公共的 API，还是一个纯粹的内部 API，抑或是一个只有受信伙伴可以访问的 API——它们都可以从 REST 的哲学中受益。

如果你想学习如何编写 API 客户端的话，那么这本书对你来说并不是必要的。这是因为大多数现有的 API 设计都基于一些有着数年之久的假设，而这些假设正是我想要摧毁的。

大部分今天的 API 都有着一个很大的问题：一旦部署，它们将无法改变。有一些大名鼎鼎的 API 会在一次部署后多年保持静态不变，即使围绕它们的行业发生着改变，这是因

注 1　Discordia 是罗马神话中专司纷争与混乱的女神，Principia Discordia 则是一部与该女神相关的宗教书籍。——译者注

为要改变它们非常困难。

但是 RESTful 架构是为掌控变化而设计的。万维网由数百万的网站组成，运行在数千种不同的服务器实现之上，并且经历着周期性的重新设计。这些网站被数十亿的用户访问着，而这些用户使用着几十种硬件平台之上的数百种不同的客户端实现。你的部署在一开始可能看上去不会如此混乱，但是当你的应用越发接近 web 的规模时，你将会看到越发相似的混乱景象。

要改变一个非常简单的系统通常都是很容易的。在规模很小时，一个 RESTful 系统比一个一键式的解决方案（push-button solution）需要花费更多预支的设计成本。但是当你的 API 逐渐成熟并开始发生变化时，你将会真正需要一些像 REST 这样的方式来应对变化。

- 一个商业上成功的 API 将保持连续多年的可用。一些 API 拥有数百甚至是数以千计的用户。就算问题域只是偶然地发生变化，对客户端带来的累积效应将是非常大的。
- 有一些 API 一直都在发生变化，新的数据元素和业务规则不断地被添加进来。
- 在某些 API 中，每个客户端都可以通过改变工作流来使其适合自己的需求。即使 API 自身从不变化，每个客户端对 API 的经历（鉴于经历不同的工作流）将会不同。
- 编写 API 客户端的人通常不会和编写服务器的人隶属于同一个团队。所有向公共开放的 API 都属于这一类。

如果你不知道外部的客户端是哪种类型的话，你需要在做出变化时格外小心——否则你就需要一个能够在发生变化时保证不会破坏所有客户端的设计。如果你为你的 API 复制了现有的设计，你将很可能只是在重复以往犯过的错误。不幸的是，大部分的改进发生在幕后，它们大都还处于实验阶段并需要经过漫长的标准流程。我将会在本书中讨论到数十种特定的技术，包括很多还仍然处于开发之中。但是我的主要目标是要教会你 REST 的基本原则。通过对这些内容的学习，你将可以对任何实验成果以及那些通过流程审核的标准善加利用。

这里有两个我想在本书中尝试解决的具体问题：重复的工作以及对超媒体的逃避。让我们来看看它们。

重复的工作

现今已发布的 API 都是根据托管它们的公司的名字进行命名的。我们谈论着“Twitter API”、“Facebook API”和“Google+ API”。这三套 API 做着相似的事情。它们都拥一些用户账户的概念，（在其他方面）它们都允许用户向自己的账户发布文本信息。但是每个 API 都具有完全不同的设计，学习一个 API 并不能帮助你学习下一个。当然，

Twitter、Facebook 和 Google 都是互相竞争的大型公司，它们并不想让你很容易地就学会它们竞争对手的 API。但是小公司和非营利性组织也在做着相同的事。它们重新设计着自己的 API，就好比从来没有人在这方面有过相似的想法一样，但是这干扰了它们想让人们实际使用它们的 API 的目标。

让我来向你展示一个例子吧。网站 ProgrammableWeb（*http://www.programmableweb.com/*）拥有着一个超过 8000 个 API 的目录。当我正在编写此书之时，它已经收录了 57 种微博 API——这些 API 的主要用途是向用户的账户发布文本信息[注2]。很不错，有 57 家公司在这个领域发布了 API，但是我们真的需要 57 种不同的设计吗？我们在这里讨论并不是那些复杂难懂的业务，例如保险政策或法规守则，我们讨论的只是向用户账号发布少量的文本信息。你想成为那个设计第 58 种微博 API 的人吗？

最显而易见的解决方案便是创建一个微博 API 的标准。但是我们已经有了一个可以很好工作的标准：Atom 发布协议（Atom Publishing Protocol）。它发布于 2005 年，然而几乎没有人使用它。有一些关于 API 的原因，使得每个人都想从头开始设计他们自己的 API，即使从业务的角度来看这并没有什么意义。

我不认为凭我一个人的力量就能结束这种无用功，但是我确实认为可以将问题分解成若干有意义的小块，然后提供一些方式来让新的 API 可以复用已经完成的这些工作。

超媒体很难

早在 2007 年，Leonard Richardson 和 Sam Ruby 编写了本书的前身，*RESTful Web Services*(O'Reilly)。那本书同样也尝试于解决两个大的问题。其中一个问题已经被解决，而另一个却没有任何进展[注3]。

第一个问题是：在 2007 年，在 API 设计的多个阵营中，REST 学派正在与使用基于 SOAP 等重量级技术的对手学派进行对峙，他们忙于应对来自对方的对 REST 学派合理性的质疑。*RESTful Web Services* 一书的出现打破了这种对峙的僵局，有效地为 RESTful 设计原则防御了来自 SOAP 学派的进攻。

很好，这场对峙已经结束，而 REST 赢得了胜利。SOAP API 仍在被使用着，不过仅限于那些起初支持 SOAP 学派的大公司。几乎所有面向公众的 API 口头上都说自己遵守了 RESTful 原则[注4]。

注2　具有微博(microblogging)标签的ProgrammableWeb API的完整列表提供了有关列表中每个API的信息。

注3　*RESTful Web Services* 一书现在是 O'Reilly 开放图书项目（*http://oreilly.com/openbook/*）中的一部分。你可以从该图书的页面下载到它的 PDF 版本。

注4　如果你想知道，这便是我们为什么更改了本书书名的原因。“web service”这个词与 SOAP 耦合得过于紧密，当 SOAP 没落之时，将会带着“web service”一词一起日暮西山。这些日子，每个人都开始改为谈论 API 了。

这又将我们带到了第二个问题：REST 并不只是一个技术词汇——它同样还是一个营销术语。在很长一段时间里，REST 成了一个口号，它象征着任何站在 SOAP 学派对立面的势力。任何没有使用 SOAP 的 API 都将自己标榜为 REST，即使它的设计与 REST 毫无关系甚至是违背了 REST 的基本原则。这样做是错误的，是令人困惑的，它给 REST 这个技术词汇带来了一个坏名声。

这种情况自 2007 年便有了较大的改善。每当我审视那些新的 API，我看到了开发者们的工作，可以看得出，这些开发人员是理解那些我将在本书前几章中解释的概念的。今天大部分举着 REST 大旗的开发者都理解资源和表述，理解如何使用 URL 来为资源命名，以及如何正确地使用 HTTP 方法。因此本书的前三章将不需要做过多的事情，只需让新的开发者加速赶上我们即可。

但是在 REST 中，还有一个方面令大部分的开发人员仍然无法理解，即：超媒体。我们都理解 Web 环境中的超媒体。它只是作为代表链接的一个华丽的词汇。网页经过互相的链接，随即产生了万维网，万维网便是由超媒体驱动的。但是，貌似只要在 web API 中涉及到超媒体，我们便有了心理障碍。这是一个大问题，因为超媒体是一项能让 web API 优雅处理变化的特性。

从 *RESTful Web APIs* 一书的第 4 章开始，我的首要目标便是教会你超媒体的工作原理。如果你从未听到过这个词，我将会结合其他重要的 REST 概念向你讲授该词。如果你听到过超媒体，但是这个概念吓到了你，我将会尽我所能来为你建立勇气。如果你无法将超媒体装进你的大脑，我将会以各种我所能想到的方式来向你展示超媒体，直到你记住并理解它。

RESTful Web Services 一书也涉及到了超媒体，但是这并不是该书的重心所在。就算跳过该书的超媒体部分也可以照样设计出一个功能性的 API。相比之下，*RESTful Web APIs* 则是一本真正有关超媒体的书。

我之所以这样做是因为超媒体是 REST 最重要的一个方面，也是最不被理解的一个方面。在我们完全理解超媒体之前，REST 将会被继续视为一个营销术语，而不是对处理分布式计算复杂性的一次认真的尝试。

这本书讲了什么？

前 4 个章节介绍了 REST 背后的概念，并将其应用于 web API。

第 1 章，网上冲浪

这一章通过一个你已经熟悉的 RESTful 系统：网站，来对基本的术语进行了说明。

第 2 章，一个简单的 API

这一章将我们在 Web 上的经验转换到了一个可编程 API 中，该 API 与第 1 章中所讨论的网站具有相同功能。

第 3 章，资源和表述

资源是 HTTP 中的基本概念，而表述则是 REST 中的基本概念。本章对它们之间是如何进行关联的进行了说明。

第 4 章，超媒体

超媒体是一种缺失的材料，可以将它和表述一起整合进一个一致的 API。本章展示了超媒体可以做些什么，并大都使用了你已经熟悉的超媒体数据格式：HTML。

接下去的 4 个章节描述了用于设计超媒体 API 的不同策略：

第 5 章，领域特定设计

显而易见的一个策略是设计一个全新的标准来处理你的具体问题。我使用了 Maze+XML 标准来举例说明。

第 6 章，集合模式（Collection Pattern）

很特别的一个模式：集合模式，在 API 设计中出现了一次又一次。在这一章中，我展示了两个不同的标准：Collection+JSON 和 AtomPub，它们都实现了这种模式。

第 7 章，纯 - 超媒体设计

当集合模式无法满足你的需求时，你可以使用一种通用的超媒体格式来传达任意的表述。这一章使用了 3 种通用超媒体格式（HTML、HAL 和 Siren）作为实例来展示上述方式是如何工作的。这一章同样还介绍了 HTML 微格式和微数据，并以此引出了下一章的内容。

第 8 章，Profile

Profile 可以用于填平某种数据格式（可以被多种不同的 API 使用）与某个特定 API 实现之间的鸿沟。我所推荐的 profile 格式是 ALPS，但是我同样也提到了 XMDP 和 JSON-LD。

在这一章中，我给出的建议开始超越了编写本书时的艺术状态（outstrip the state of the art）。因此我不得不为本书开发了 ALPS 格式，因为当时还没有其他可以完成这项工作的选择。如果你已经对基于超媒体的设计非常熟悉，那么你可以跳过前面的部分直接来到第 8 章，但是我不认为你应该跳过第 8 章。

第 9 章到第 13 章涉及到了例如选择正确的超媒体格式以及如何充分利用 HTTP 协议这些实用主题。

第 9 章，API 设计流程

这一章将本书到目前为止讨论过的所有内容整合在了一起，并给出了一个用于设计 RESTful API 的按部就班的指引。

第 10 章，超媒体动物园

为了展示超媒体的能力，本章讨论了近 20 种标准化的超媒体数据格式，它们中的大部分并没有在本书的其他章节中有所涉及。

第 11 章，API 中的 HTTP

本章给出了在 API 实现中使用 HTTP 的一些最佳实践。我同样也讨论了一些 HTTP 的扩展，包括即将到来的 HTTP 2.0 协议。

第 12 章，资源描述和 Linked Data

Linked Data 是语义网社区用以实现 REST 的方法。JSON-LD 毫无疑问是最重要的 Linked Data 标准。我们曾在第 8 章谈到过它，而我在本章中对它进行了重温。本章同样讨论了 RDF 数据模型和一些我在第 10 章没能谈到的基于 RDF 的超媒体格式。

第 13 章，CoAP: 嵌入式系统的 REST

本章通过对 CoAP 的讨论结束了本书的核心部分，CoAP 是一个完全没有使用 HTTP 的 RESTful 协议。

附录 A，状态法典

作为对第 11 章的扩展，本附录对 HTTP 规范中定义的 41 个标准状态码以及一些作为扩展定义的有用的状态码进行了深入的考察。

附录 B，HTTP 报头法典

与附录 A 相似，本附录也是对第 11 章的扩展。它为 HTTP 规范中定义的 46 个请求和响应报头，以及一些扩展提供了详细的概述。

附录 C，为 API 设计者准备的 Fielding 论文导读

本附录包括了一个围绕 REST 的基础文档（即 Fielding 论文）展开的深度讨论，用以说明该文档在 API 设计中的意义。

词汇表

该词汇表包含了你在 *RESTful Web APIs* 一书中会经常遇到的一些术语的定义。如果

你想要熟悉基本概念，或者是需要一个快速的、用于浏览特定概念定义的提示工具，那么这将是一个很好的去处。

这本书没讲什么

RESTful Web Services 是最早有关 REST 的一部书籍，并且涉及了很多的背景知识。幸运的是，现在已经有着超过一打的关于 REST 的各个方面的书，而这样便让 *RESTful Web APIs* 可以腾出精力来专注于核心的概念。

为了保持本书的专注度，我去除了部分你可能会希望我覆盖到的主题。我想要告诉你这本书没讲什么，这样你便可以选择不购买本书，从而不会为此感到失望：

- 本书没有涉及到客户端编程。编写客户端来消费基于超媒体的 API 是一种新的挑战。眼下，对一个通用的 API 客户端来说，我们可以拥有的就是一个能够发送 HTTP 请求的代码库。这在 2007 年如此，时至今日仍然如此。因为问题存在于服务器端。

  当你要为一个现有的 API 编写客户端时，你将只能仰仗 API 设计者。我无法给你任何通用的建议，因为现在并不存在跨 API 的一致性。这就是为什么我在本书中鼓动大家保持对服务器端一致性的积极性的原因。当 API 之间变得更加相似，我们也就可以编写更为精细的客户端工具。

  第 5 章包含了一些示例的客户端实现，并尝试对不同类型的客户端进行归类，但是如果你想要一本全部关于 API 客户端的书，那么这本书并不适合你。我不认为目前市场上存在着你想要的书。
- 世界上部署最广泛的 API 客户端应当属 JavaScript 的 XMLHttpRequest 代码库。在每一个浏览器中都拥有一份该代码的副本，而当今大部分的网站也都构建于那些专门设计供 XMLHttpRequest 消费的 API 之上。对于本书来讲，这个领域过于庞大而无法完全涉及。市场上有着全篇专门讲述 JavaScript 代码库的书籍。
- 我花费了相当多的时间来讨论 HTTP 的机制（第 11 章、附录 A 和附录 B），但是我没有覆盖到任何具有深度的特定 HTTP 主题，有一些主题，尤其是关于缓存和代理这样的 HTTP 中间组件的相关主题，我只是简单地在本书中有所涉及。
- *RESTful Web Services* 重点专注于将你的业务需求拆分成一组互相关联的资源。根据我 2007 年以来的经验，我深信将 API 设计作为一种资源设计来思考可以有效地避免考虑超媒体。但本书却采用了一种截然不同的方式，它专注于表述和状态转换，而非资源。

  也就是说，资源设计的方式无疑也是有效的。如果想听听在该方向发展的建议，我推荐由 Subbu Allamaraju 编著的 *RESTful Web Services Cookbook*(O'Reilly)。