O'REILLY®

Broadview®
www.broadview.com.cn

RESTful Web Clients

基于超媒体的可复用客户端

RESTful Web Clients

[美] Mike Amundsen 著
曾著 徐必涛 译

中国工信出版集团

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

O'REILLY®

RESTful Web Clients
基于超媒体的可复用客户端

RESTful Web Clients

[美]Mike Amundsen 著

曾著 徐必涛 译

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

Web开发领域的REST运动已经进行了很多年了，在REST的Richardson成熟度模型提出后，第3级——HATEOAS的应用——仍然没有得到广泛应用。事实上，其中一个难点在于客户端如何支持HATEOAS。之前很多REST相关书籍聚焦于如何打造服务端的RESTful API，本书则着重研究RESTful客户端，介绍了如何把一个针对服务端规约硬编码的定制客户端重构为一个支持HATEOAS的通用客户端，并提供了多格式支持、超媒体类型、版本化、微服务等相关问题的全面指导。本书附有所有样例代码的GitHub地址，方便读者快速理解和实践。

本书适合Web应用开发者，尤其适合希望Web应用程序的服务端与客户端能够独立演化的Web架构师。

版权贸易合同登记号 图字：01-2017-3325

图书在版编目（CIP）数据

RESTful Web Clients：基于超媒体的可复用客户端 /（美）迈克·阿蒙森（Mike Amundsen）著；曾著，徐必涛译. —北京：电子工业出版社，2018.8
书名原文：RESTful Web Clients
ISBN 978-7-121-33758-1

Ⅰ. ①R… Ⅱ. ①迈… ②曾… ③徐… Ⅲ. ①超文本标记语言－程序设计 Ⅳ. ①TP312

中国版本图书馆CIP数据核字(2018)第036154号

策划编辑：张春雨
责任编辑：牛　勇
封面设计：Randy Comer　张　健
印　　刷：三河市华成印务有限公司
装　　订：三河市华成印务有限公司
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：22.25　字数：488千字
版　　次：2018年8月第1版
印　　次：2018年8月第1次印刷
定　　价：99.00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）88254888，88258888。

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。

本书咨询联系方式：010-51260888-819，faq@phei.com.cn。

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的会议和峰会集聚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

我多次承诺会写作这样一本书，感谢那些忍受我这么多次承诺的人们。
我诚恳地向各位致歉——为了曾经的诺言和本书有限的成果。

推荐序：一场与超媒体的未了情缘

在 2007 年，当我初次翻译 Roy Fielding 关于 REST 的博士论文（中文名为《架构风格与基于网络应用软件的架构设计》）时，自己对 Web 的整体架构是毫无概念的。无知者无畏，当时我仅仅是出于求知欲就开始了翻译工作。后来我发现这个挑战严重超出了我的能力范围，Fielding 的博士论文是我翻译过的专业技术著作中难度最高的。后来我在 2013 年重新翻译了这篇博士论文，力求把初次翻译时的大量问题和留下的遗憾都弥补上，而不至于误人子弟。

这篇博士论文是一座内涵丰富的宝藏，可以说是 2000 年前 Web 架构相关论文中的集大成者，其重要性不亚于 Tim Berners-Lee 爵士的论文。文中系统地阐述了 HTTP 1.1 协议背后的设计原理——REST。绝大多数不理解 REST 思想的 Web 开发者都会陷入两个误区之中：一个是将 HTTP 简单看作是一种传输协议（而不是应用协议），另一个是完全忽视超媒体的重要性。在 2005 年 REST 思想随着 Web 2.0 的兴起和迅速发展而逐渐普及之后，脱离第一个误区的 Web 开发者越来越多，但是能脱离第二个误区的 Web 开发者还是极少数。Fielding 博士其实从最初就认为 REST 和超媒体是不可分割的，而 HTTP 1.1 最重要的设计目标就是更好地支持超媒体。在当时的语境中，这个超媒体自然就是 HTML。因为关于超媒体的误解如此之多，所以 Fielding 博士在 2008 年写了一篇博客文章 *REST APIs Must be Hypertext-driven*（《REST API 必须是超文本驱动的》）。

这篇博客引起了 Web 开发社区的广泛讨论和反思，随后 *RESTful Web Services* 一书的作者 Leonard Richardson 提出了一个 Richardson 成熟度模型。这个模型将 RESTful API 划分成了 3 个等级，把能够熟练应用超媒体的 RESTful API 列为了最高等级——第 3 级。然而很多年过去了，在普通 Web 开发者看来，这似乎仍然是一个乌托邦式的目标。有些 RPC 铁杆粉丝总是会拿这个来讥讽 REST 爱好者，把他们说成是中看不中用的 API 设计美学爱好者。这些传统的观点过于强大，以至于 REST 爱好者对于在 API 设计中使用超媒体也视为畏途。

当然，我们都是工程派，而不是学院派，随便给别人乱扣学院派的大帽子是一种不尊重人的行为，也是不求甚解的体现。包括 REST 思想的创造者 Roy Fielding 也是“如假包换”的工程派。他早年为很多开源项目贡献过大量代码，还指导了大量 HTTP 客户端 / 服务器端开发团队，他的实战能力肯定在绝大多数开发者之上。REST 当然并不是一种 API 设计美学，它更像是中庸之道，包含了设计 Web 应用的各种架构权衡。虽然 Fielding 在设计 HTTP 1.1 和撰写博士论文时，超媒体主要指的是 HTML，但是 REST 是通用的理论，能支持的超媒体类别远不止 HTML 一种。在 Fielding 发表 2008 年博客文章之后，Web 开发社区与超媒体相关的研究和开发实战非常活跃，出现了大量新型的超媒体，而且越新出现的超媒体，越是基于 JSON 而非 XML 的。一直很支持 REST 设计开发的 O'Reilly 出版社，也不失时机地出版了很多 REST 开发相关的图书，这些图书目前已经形成了一个强大的系列，包括：

- *RESTful Web Services*
- *RESTful Web Services Cookbook*（中文版为《RESTful Web Services Cookbook 中文版》）
- *REST in Practice*（中文版为《REST 实战》）
- *RESTful Web APIs*（中文版为《RESTful Web APIs 中文版》）
- *RESTful Web Clients*（中文版为《RESTful Web Clients：基于超媒体的可复用客户端》，即本书）

另外还有很多针对某个开发平台的 REST 开发图书，例如：

- *RESTful .NET*
- *Building Hypermedia APIs with HTML5 and Node*（中文版为《使用 HTML5 和 Node 构建超媒体 API》）
- *PHP Web Services: APIs for the Modern Web*

在这些优秀的 REST 开发图书之中，涉及了超媒体的图书中最有代表性的是如下 3 本：

- *REST in Practice*（中文版为《REST 实战》）
- *RESTful Web APIs*（中文版为《RESTful Web APIs 中文版》）
- *RESTful Web Clients*（中文版为《RESTful Web Clients：基于超媒体的可复用客户端》，即本书）

我有幸作为 *REST in Practice* 一书的翻译负责人，赵震一负责翻译 *RESTful Web APIs*，曾著负责翻译 *RESTful Web Clients*。我们就像是一个接力赛的团队一样，在 7 年的时间里，把接力棒传递下去，希望能通过这 3 本书向国内的 Web 开发者全面展示 RESTful API 和超媒体的独特魅力。

这 3 本书由浅入深，逐步揭开了支持超媒体的第 3 级 RESTful API（也就是所谓的 Web API）的神秘面纱。尤其是第 3 本书 *RESTful Web Clients*，令人信服地展示了使用超媒体之后，对于 API 客户端代码的复用性和松耦合起到的巨大作用。除此之外，这本书最重要的贡献是让超媒体变得如此平易近人，要达到这个目标其实是非常困难的。作者 Mike Amundsen 虽然功力深厚，但为了保证本书的质量，原著足足推迟了一年时间才上市。结果不负众望，Mike 为读者奉献了一本高质量的图书。连 Roy Fielding 在 Twitter 上也承认 Mike Amundsen 对于超媒体如何使用的理解，与 REST 博士论文是最接近的。相信认真阅读完本书的 Web 开发者对在 API 设计中适当使用超媒体不再会那么犹豫。其实在 API 设计中适当使用超媒体，就应该像我们平时写代码时要写单元测试一样习以为常，并且熟能生巧，不断探索在 API 设计中使用超媒体的乐趣。

另外我还建议读者在读完 *RESTful Web Clients* 之后，再去认真读一下 Mike Amundsen 的上一本书 *RESTful Web APIs*，因为这两本书有很强的关联性。*RESTful Web APIs* 虽然很棒，但是偏重于概念阐述，实战性不足，*RESTful Web Clients* 弥补了 *RESTful Web APIs* 在实战性方面的缺憾。思想的发展是有传承性的，没有几年前的 *RESTful Web APIs* 就不会有 *RESTful Web Clients*。对于 API 来说，服务器端是皮，客户端是毛，皮之不存，毛将焉附？

顺便说一下，虽然现在 HTTP 1.1 已经升级到了 HTTP 2，不过 REST 和超媒体的思想是完全适用的。如果熟悉 HTTP 2，你会发现，其实 HTTP 2 的设计还是为了更好地支持超媒体（特别是 HTML5），HTTP 2 仍然是与超媒体紧密相关的。这再次证实了 Fielding 在 REST 博士论文中所阐述的思想，REST 是 Web 自身的架构风格，REST 就是一切优秀 Web 应用的灵魂，而 REST 自身的灵魂就是超媒体。超媒体是计算机软件领域最伟大的思想之一，它是 Web 应用取之不竭的力量源泉。

开卷有益，最后我代表国内的 Web 开发者感谢本书的两位译者曾著、徐必涛的辛勤工作。也感谢博文视点张春雨老师，他 10 年来坚持不懈地支持 REST 开发图书的出版。

Web 架构师　李锟

2018 年 3 月 4 日于上海

译者序

2017 年年末，就职小米的一位前同事送了我一枚 F 码，我用它抢购到一枚小爱音箱。我满怀期待地装上“小爱同学”，希望能够通过她用语音控制所有小米产品。但我失望地发现，早先我购买的 YeeLight 床头灯并不能接受“小爱同学”的指挥——YeeLight 床头灯的客户端只能适配其出厂时的服务端所提供的功能，在服务端扩展新的功能，比如接入智能语音控制之后，已经发布的客户端却无法跟上，无法获得新功能。

随着互联网尤其是物联网的发展，跨企业、跨行业的互联需求越来越多，上述遗憾会频繁发生，无疑会造成巨大的浪费，也阻碍了物联网的发展。我们迫切需要服务端和客户端相互独立演化的能力，这正是 RESTful 服务端与客户端的用武之地，也是本书的主要目的。

本质上，服务端和客户端独立演化的能力取决于双方的实现细节的解耦程度。首先，如果服务端与客户端紧密耦合，服务端变更就很可能造成客户端失效。例如，服务端和客户端通过 RPC 的 IDL 来约定服务接口，那么当服务端的变更引起过程名、参数或返回值的变更时，客户端就很难不受其影响。又例如，服务端的 Web Service 遵循一套 URL template 规约，客户端根据服务端提供的规约文档编码，如果服务端改变了资源名称或资源之间的关联关系，也会导致客户端失效。其次，与服务端紧密耦合的客户端显然也无法正常接纳服务端推出的新功能。

REST 解决的思路就是将可能变化的部分抽象出来，融入服务端响应的消息体中，如果客户端拥有解读这些变化的能力，也就使双方都获得了独立演化的能力。

在过去我们非常熟悉的“通用浏览器 + 服务端渲染逻辑（CGI、JSP、ASP、PHP 等）+HTML”方案中，服务端输出了整个 HTML 文档，自然也包含了服务端所有可能变化的部分。从理论上说，客户端和服务端的确获得了相互独立演化的能力——浏览器是通

用的，不需要应用程序员编写任何客户端代码，也并不因为服务端改变了 JSP 脚本或增加一个功能，就需要升级浏览器。这是一个极为流行的方案，成功地挤掉了 C/S 结构的份额，成为当时互联网应用的主流方案。

然而，这个服务器渲染的方案并不够好。因为服务端响应是完整的 HTML 文档，不仅有业务数据，而且包括渲染的每一个细节，客户端要么全要，要么全不要。如果客户端希望对业务数据再加工，或者局部渲染，就不得不剥离易变的其他元素，例如 <p/> 这个渲染相关的元素。写过爬虫的程序员应该深有体会，好不容易把有用数据从 HTML 大杂烩里提取出来，结果目标网站稍微改动一些风格布局，原先的爬虫程序就失效了。由于难以局部渲染，人们不得不忍受 HTML 页面跳转的糟糕体验。AJAX 和 JS 客户端渲染脚本的兴起成为压倒骆驼的最后一根稻草，传统的服务器渲染方案被逐步取代了。

拥有 JS 渲染脚本和静态 HTML 模板的富客户端，将服务端渲染方案中 view 的职责从服务端抢了过来，服务端只输出数据，不必带上各种 HTML 标签。职责的分离大大促进了复用，一个服务端可以拥有多个不同展现形式的客户端，一个客户端可以 mashup（mashup 的定义请参考 *https://en.wikipedia.org/wiki/Mashup_(web_application_hybrid)*）多个服务端，跨企业的应用互联变得很流行。但这样一来，过去 C/S 结构的老问题——服务端与客户端紧耦合的毛病又出现了，如果客户端团队与服务端团队不是同一拨人，独立演化造成不兼容就会成为大问题。

客户端渲染和客户端服务端独立演化能否兼得？本书给出了答案。

本书首先讲到，如果服务端响应中包含下一步操作的所有可能路径，客户端应该能够正确理解和处理这些路径代表的操作，人作为交互模型的主动角色，通过操控客户端选择路径，形成“探索地图”式的人机交互形式。其中客户端是不需要事先知道服务端提供的操作路径的，这就形成了服务端和客户端对于操作路径的独立演化的能力，换言之，服务端可以修改、增加、删除这些操作路径而不会引起客户端失效。

然后，作者将类似“操作路径”这样的客户端需要识别的变化部分归纳并抽象为 OBJECT、ADDRESS、ACTION（简称 OAA）三个重要元素，客户端需要理解响应中携带的 OAA 的元数据，才能有效处理 OAA 元素的变化。这些元数据如下。

- OBJECT 元数据：服务端返回业务数据的元数据，例如对象中字段的类型、长度甚至语义等，这些元数据能够帮助客户端正确地渲染展现数据。
- ADDRESS 元数据：响应中业务数据的关联操作列表，包括名字、URL、展示名称等，能够指示客户端展现当前可操作的最新功能。
- ACTION 元数据：对 ADDRESS 列表每一项操作的详细描述，包括参数的元数据（与 OBJECT 字段元数据类似）、名字、URL、HTTP 方法、展示名称等，能够指示客户

端引导用户正确地发起这个操作。

接着，作者在第 3 章讲述了如何在服务器响应中包含 OAA 这些可变因素，在第 4 章、第 6 章、第 8 章讲述了如何通过一些表述格式让客户端理解和正确处理 OAA。难得的是，作者并没有假设理想化从头开始的场景，而是让虚拟主人公 Bob 和 Carol 从老团队和传统的设计中，将传统的个性化定制客户端一步一步重构和扩展为可复用的支持多种表述格式的通用客户端。

最后，作者还讲述了处理版本化和微服务需要注意的地方。

RESTful Web 客户端是整个 REST 风格应用（服务端和客户端）的一部分，服务端已经有不少著作，《RESTful Web Clients：基于超媒体的可复用客户端》则是 HATEOAS 的 RESTful 应用缺失的最后一块拼图，对于实践 RESTful 服务和客户端有重要的指导意义。

作为译者，特别提醒读者在阅读过程中留意几点，第一，本书是如何将系统中稳定不变的部分、较稳定不易变的部分和经常变化的部分相互分离，分别在服务端、客户端、通信报文中描述，以期获得最大复用的同时促进服务端和客户端的独立演化的。第二，从服务端渲染方案的缺陷我们可以得出结论，客户端并不是越通用越好。当缩小复用范围、聚焦某个行业领域时，保留客户端与服务端之间对该行业领域语义的耦合和隐式的规约，反而能够增强在领域内的复用程度而不影响行业领域内的复用。展望未来，RESTful 运动可能会产生若干包含协议语义的通用格式与具体应用领域 DSL 相结合的应用模式。最后，本书留了一个待解决的问题，即当客户端不是由人类，而是由另一程序操控时，地图将退化为规则操控的路径，成为自动化的客户端；当这个规则是 AI 操控时，这个客户端将进化为智能客户端，这时，YeeLight 将不仅能接受“小爱同学”的指挥，而且能胜任更多预先并不知道的工作。

在翻译的过程中，我得到了李锟、赵震一等翻译 REST 系列服务端著作的前辈的帮助，和 REST 实战讨论组里的李念辉等学友的讨论也让我受益匪浅。另外，出版社的张春雨老师对译文的规范性提出了不少意见和建议，在此一并表示感谢。由于水平有限，错漏难免。如读者发现错漏，请不吝赐教，我的邮箱是 *zengzhu.cn@gmail.com*。

原书推荐序

在本书的先驱之作 *RESTful Web APIs* 的前言中，Sam Ruby 说道：

> 希望 *RESTful Web APIs* 这颗鹅卵石可以像它的先驱 *RESTful Web Services* 那样具有影响力。但谁也说不好，也许七年之后这一切又将重演，从而强调表述性状态迁移（Representational State Transfer）仍然没有受到应有重视的方面。

现在虽不是此前的七年之约，但 *RESTful Web Clients* 已经如期而至。Mike 在 API 领域精耕细作多年，通过本书的文字，我们可以领略他在写作和思想上一贯清晰的风格，以及那部分被忽略的 Web API。

REST 一词出自 Roy Fielding 的博士论文《架构风格与基于网络应用软件的架构设计》。在论文的前面部分，Fielding 介绍了用于描述 REST 的七个基本架构约束，第一个是客户 – 服务器（Client-Server，CS），其中描述如下：

> 分离关注点是在客户端 – 服务器约束背后的原则。功能的适当分离会简化服务器组件，从而提高可伸缩性。这种简化所采用的形式通常是将所有的用户界面功能（user interface functionality）移到客户组件中。只要通信的接口不发生改变，这种分离允许两种类型的组件独立地进化。

显然，通过这种方式设计系统时，服务端和客户端同等重要。然而，那些基于 Fielding 论文所完成的工作中普遍存在一种偏见：几乎所有的工作都聚焦在服务端，却很少讨论客户端。这样便遗漏了一个非常重要的部分，RESTful 架构的一些好处只能通过良好设计的客户端来实现了。虽然有许多客户端 – 服务器架构风格的系统，但都没有遵循 Web 中的一些原则。如果我们必须采取某种方式让服务端更接近 Web 的运行方式，那么我们也必须改变客户端的编写方式。

事实上，在我们构建 Web API 时，这是向 Web 思维方式转变的一个难点所在。对此，我与热衷于部署超媒体技术的组织有过诸多讨论，他们不懈地将这些努力付诸实践。我相信这其中的很多困难都来自超媒体的生产，因为并没有考虑到 API 消费方式的变化。如果当 API 设计人员期望构建一个 Web API 客户端时，可以采用消费其他 API 的原则来消费 RESTful API，那么结果将会令人沮丧。不过，这一切都是可以理解的，毕竟那些倡导这种架构风格的人员几乎都集中在 API 的生产方。

事后来看，这种缺陷其实很明显：如果我们只关注方程的一边，我们该如何设计一个好的系统呢？我确信这种情况是现如今人们对 API 认识所导致的一个二次效应：我的组织关注的是如何为我的产品提供一个 API，但使用这个 API 的是你的组织，所以如何处理它是你分内的事。过去的五到十年中，这种关于 API 的潜在情绪有所扩大：转向“简单的 API，我们不需要复杂的 SDK 来完成工作”。同时，提供 API 客户端的组织也常常被投以怀疑的目光。不过，我们现在可以看到事情已经开始反转，开发人员已经厌倦每次为新的 API 重新部署一次客户端。官方提供的 SDK 正被逐渐正视，通过这些 SDK，API 的消费者可以更多地将关注点放在自己的应用上，而不是 API 的集成上。

此外，Web 也越来越被看作是应用平台，而不是用于分享文本文档的一种简单方式。随着专属平台的兴起，特别是在移动领域，那些热爱 Web 自由和开放的人们，正在动员更多的人来显著扩展 Web 平台的能力。为了利用这一优势，随着他们的产品愈发雄心勃勃，应用也随之萌芽了越来越多的 JavaScript 代码。而这些应用正是另外一种客户端，随着时间的推移，它们也将日趋重要。

随着格局的转变，组织需要再次考虑方程两边的平衡问题。从广义上讲，我认为这会产生更好的 API，但它们不会都是“玫瑰”。正如我之前所提到的，你可以买到无数的书籍来帮助你了解方程的服务端一边，但是在客户端方面却完全没有一个类似的指导手册——至今仍是如此。

自从 Mike 告诉我他开始着手这一工作以来，我一直在期待着这本书，而他也确实没有让我失望。对于 Web API 方程式中缺失的那部分，本书堪称一部梦幻般的指南，我相信它的作用和影响在未来数年内将得到关注。希望你也可以像我一样享受这本书的阅读过程。

——*Steve Klabnik*

前言

开始阶段是工作中最重要的部分。

——柏拉图（Plato）

基于 Web 的 REST 和超媒体服务正日趋普遍，但却鲜有客户端可以利用这些 API 的强大功能。这主要是因为缺乏创建成功的超媒体客户端的技术和模式——长期以来它们一直被忽视。然而，如果可以采取恰当的方式，基于超媒体的客户端应用将会比典型的一次性客户端具有更强的稳定性和灵活性。

本书旨在为读者提供一个坚实的背景知识和一些可工作的示例源码，这些示例为如何处理超媒体 API 提供了明确的建议。本书的主要思想之一是客户端应用程序应该依赖于 *Request*、*Parse* 和 *Wait* 所构成的循环，我简称其为 RPW 循环。这是所有电脑游戏采用的实现方式，也是所有事件驱动接口从窗口式工作站（windowing-style workstation）到响应式机器接口（reactive machine interface）的工作方式。

有人告诉我，一些前端的开发人员可能会认为 RPW 模式比较生僻，甚至也有人认为我的建议比较“激进”。对于这些观点，我都能理解。当下，众多的前端开发库和实践都在致力于设计用于专门构建的一次性用户界面，这些界面难以修改，并且很难在运行时对服务提供的新信息做出反应。不过，在阅读本书的例子后，我希望前端开发人员（他们中的大多数都比我的经验更丰富）能够在目前初步工作的基础上创造出更丰富的最佳实践、工具和可重用的库，为越来越多的超媒体 API 打造高质量的用户体验，力争在不需要频繁升级的情况下满足高质量的用户体验需求，以及不断发展的服务接口自适应客户端的需求。

本书讲了什么

本书将带领读者开启一段从个性化定制实现到强大的通用客户端应用的旅程，同时向你

展示如何利用那些支撑万维网良好运行的基本原则。本书包括以代码为中心的章节信息和对相关重要主题的探索，比如表述器模式、人机交互模型和 Web API 在版本控制上的挑战等。当然，其中也少不了大量的代码（我为本书创建了 20 多个 GitHub 仓库），包括一些小的代码片段。不过，这些片段单独看可能不易理解。因此，我会为读者指出完整的在线代码库的位置，你可以在相关代码库中找到本书所涵盖的全部功能实例。

各章内容

本书探讨了通用超媒体风格客户端的世界，包括它们的外观、与典型 JSON 对象客户端的区别，以及客户端和服务端开发者如何构建更易于支持和适应的系统。本书包含了一些专项讨论所选格式（如 HTML、纯 JSON、HAL、Siren 和 Collection+JSON）的章节，以及所有 Web 开发人员需要熟悉的理论和实践的章节，包括服务端对消息格式的支持、人机交互模型、版本控制，以及一个在同时与多个独立的后端服务进行交互时可以支持多种超媒体格式的解决方案。

本书的大多数章节都可以独立成章，在阅读时可以不讲究先后顺序。不过，为了更充分地理解和吸收本书的内容，我鼓励你将本书看成一次单程旅行，按照顺序从头读到尾。这里简单介绍一下本次旅程的路径。

第 1 章，从 HTML 到简单 Web API

本章向我们介绍了一个经典的纯 HTML 客户端。我们将通过它来了解浏览器的基本工作原理，以及它们是如何影响人们看待 Web 中那些超媒体格式的。此外，本章还介绍了将纯 HTML 服务转换为一个原始的纯 JSON 服务的过程。该服务将是我们为本书构建的所有其他客户端应用程序的基础。

第 2 章，JSON 客户端

大多数的客户端 Web 开发人员都会构建 JSON 客户端，它会记住所有的 URL，处理静态对象，并通过固定的工作路径来导航。构建 JSON 客户端是一个很好的开始，不过时间却证明它是一个糟糕的工作方式。在本章中，我们将讨论如何克服在维护纯 JSON 风格客户端应用程序时的挑战。

第 3 章，表述器模式

表述器模式是处理服务器输出的一种简单而重要的方式，也是将内部对象模型转换为外部消息模型的过程。我们将在本章中审视该模式（及其源头），并介绍如何使用服务器上的 Web 服务迁移语言（WeSTL）和浏览器客户端上的 HTML DOM 将该模式应用于 API 提供方。

第 4 章，HAL 客户端

HAL 媒体类型是目前较为流行的超媒体格式之一。比如在 Amazon 的 Web 服务团队中，至少有两个 API 使用了 HAL。对于所有 Web 客户端都需要处理的三个重要元素，HAL 负责处理其中之一：ADDRESS 挑战。我们将看到如何将 HAL 作为消息格式来构建一个通用客户端，以及如何通过 HAL-FORMS 扩展来提高 HAL 的处理能力。

第 5 章，可重用客户端应用的挑战

你会注意到，我们所构建的多数客户端看起来都很类似。从本质上讲，我们正在建立能够在探索周围世界（以某种有限的方式）的过程中发展自己的“探险者”。这些客户端都采用了 Request—Parse—Wait 循环，即 RPW 模式。该模式基于我们如何与世界交互这一命题下的几个经典概念，我们将在本章中探讨它们。

第 6 章，Siren 客户端

Siren 内容类型是另一种强大的超媒体类型。Siren 目前被用作 Zetta IoT 平台的一部分，旨在处理 Web 客户端的三个关键任务中的两个：ADDRESS 和 ACTION。我们将看到使用 Siren 作为消息格式来构建一个通用客户端，也会通过探索 Siren 拓展（Profile for Object Display，即 POD）来增强它在 UI 上显示元数据的能力。

第 7 章，版本化与 Web

当你开始将超媒体类型作为客户端 Web 应用程序的基础时，API 版本的概念会发生什么变化？本章将讨论管理随时间变化的各种尝试，以及如何依赖基于消息的超媒体风格的 API 来减少当接口特性发生变更时也要变更接口契约的问题。

第 8 章，Collection+JSON 客户端

在本章中我们将探讨另外一种超媒体格式：Collection+JSON，或称为 Cj。Cj 能够处理 Web 客户端中的所有元素：OBJECT、ADDRESS 和 ACTION。我们将看到如何将 Collection+JSON 作为消息格式来构建一个通用客户端，并学习如何扩展 Cj 的数据显示和验证程序。

第 9 章，超媒体与微服务

创建一个可以无缝调用多个服务的通用超媒体客户端需要什么？如果这些服务使用的超媒体类型各不相同，又需要什么呢？当我们谈论消息格式时，该怎样制作一个可以“说”多种语言的客户端应用？这一切将在最后一章中揭晓。

对话

本书的所有章节都以简短的小插曲或对话开始和结束。我们通过这些虚构的对话来说明，组织在思考如何在互联网上创建可扩展且稳定的项目时所遇到的挑战。这些对话既是各章主题的铺垫，也是主题的总结。

此外，对话也是一种提示，让读者有机会思考所提出的挑战，以及如何应对这些挑战并给出解决方案。每章从对话开始读起，读完后花几分钟时间来描绘（也可以写下来）一下你对于如何解决问题的想法。这种方式可以帮助你对书中提供的解决方案形成更深刻的认识，并对自己的问题解决能力有额外的洞察。

最后，这些对话还可以帮助那些只想跳跃阅读书中部分内容的读者收获一些感悟。你可以单独阅读对话，并收集相关的基础信息。各章内容也正是对对话中的细节和微妙之处进行了深入讨论。你可能会发现，至少在某些章节中，阅读对话可以获取当前主题或挑战相关的所有信息，而且还很不错。

艺术作品

本书包含了一些才华横溢之辈的艺术作品和图表。Dana Amundsen，路易斯维尔（Louisville）KY 的一位成功的艺术家，和我一起创作了本书中出现的人物 Bob 和 Carol。此外，她还设计了示例程序中所使用的 BigCo 徽标。事实上，Dana 创作的作品比本书所包含的要多得多，希望在不久的将来可以通过其他方式将这些艺术作品分享出去。

本书中的图表均由我的好友 Diogo Lucas 创作，他是一位资深的软件工程师、架构师、演说家和老师。我第一次遇见 Diogo 是在前往巴西（他的居住地）的旅途中，当谈论关于本书的一些想法时，他向我展示了令人惊叹的草图。我抓住这个机会邀请他创作插图，很幸运，Diogo 欣然同意贡献他的艺术才华。

本书所使用的 Dana 和 Diogo 的艺术作品，均是根据 Creative Commons - Attribution-NoDerivatives 4.0 International(CC BY-ND 4.0) 获得授权的。

Dana 和 Diogo 为本书增加了很特别的内容，我非常感谢他们。

本书没有讲什么

本书在相对有限的篇幅中涵盖了较广的内容范围。为了做到这一点，我需要删减一些有