Web开发经典丛书

RESTful API开发实战

使用REST JSON XML和JAX-RS构建微服务
大数据和Web服务应用

Pro RESTful APIs: Design, Build and Integrate with REST, JSON, XML and JAX-RS, First Edition

[美] Sanjay Patni 著
郭理勇 译

Apress®
清华大学出版社

Web 开发经典丛书

RESTful API 开发实战

使用 REST *JSON* XML 和 JAX-RS 构建微服务
大数据和 Web 服务应用

[美] Sanjay Patni 著

郭理勇 译

清华大学出版社

北 京

Sanjay Patni
Pro RESTful APIs: Design, Build and Integrate with REST, JSON, XML and JAX-RS, First Edition
EISBN：978-1-4842-2664-3

北京市版权局著作权合同登记号　图字：01-2017-5755

图书在版编目(CIP)数据

RESTful API 开发实战　使用 REST JSON XML 和 JAX-RS 构建微服务 大数据和 Web 服务应用/ (美) 桑杰 • 帕特尼(Sanjay Patni) 著；郭理勇译. —北京：清华大学出版社，2018
(Web 开发经典丛书)

书名原文：Pro RESTful APIs: Design, Build and Integrate with REST, JSON, XML and JAX-RS, First Edition

ISBN 978-7-302-49211-5

Ⅰ. ①R… Ⅱ. ①桑… ②郭… Ⅲ. ①互联网络－网络服务器－程序设计 Ⅳ. ①TP368.5

中国版本图书馆 CIP 数据核字(2017)第 331724 号

责任编辑：王　军　韩宏志
封面设计：牛艳敏
版式设计：思创景点
责任校对：曹　阳
责任印制：杨　艳

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　邮　　编：100084
社 总 机：010-62770175　　邮　　购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：三河市金元印装有限公司
经　　销：全国新华书店
开　　本：148mm×210mm　　**印　　张**：4.625　　**字　　数**：125 千字
版　　次：2018 年 2 月第 1 版　　**印　　次**：2018 年 2 月第 1 次印刷
印　　数：1～3200
定　　价：48.00 元

产品编号：076424-01

译者序

每个互联网从业人员都有这样一种感觉：RESTful API 概念既熟悉又陌生。熟悉的是我们能在大量开放平台或开源项目中看到它的身影，如常见的 OpenStack、Kubernetes API 等，众多企业基于开放的 RESTful API 实现了业务扩展和资产获利；而陌生的是我们似乎很难找到 RESTful API 的准确定义，也不了解如何真正在实际项目中使用它。如 REST 和 SOAP 协议到底有何区别？RESTful API 和 HTTP 协议到底存在什么关系？REST 的安全性如何保证？令人欣慰的是，拜读 Sanjay Patni 先生这本著作后，这些问题都将迎刃而解。

REST 一词是由 Roy Fielding 博士于 2000 年在他的博士论文 *Architectural Styles and the Design of Network-based Software Architecture* 中提出的，实际指一种有助于创建和组织分布式系统的架构风格。它并不是一个标准或准则，而是一种基于资源的架构风格。基于 REST 风格构建的 API 应该满足 CS 模式交互、统一的资源接口、透明的分层系统以及支持无状态和缓存等条件约束，另外需要保证 API 的安全性等。以 REST 风格构建的系统将在性能、可扩展性、可移植性、可靠性等多个方面得到提升和优化。

本书作者 Sanjay Patni 拥有 15 年以上的企业级软件开发经验，尤其在构建 RESTful API 方面有深厚的理论研究和技术实践背景。本书主要

从 RESTful API 的架构、设计和编码三个方面进行深入介绍。首先阐述 REST 的基本原理和准确定义，并对 REST 与 SAOP 协议的差异，以及 XML/JSON 等数据交换格式等进行全方位比较。其次在 API 设计和建模方面讨论 API 的完整生命周期和 RESTful API 设计的最佳实践，着重介绍 API 组合和框架如何实现 API 的一致性和可重用性，以及 API 平台管理、API 安全性和缓存机制等。另外通过一个实际案例(播客 podcast)演示如何通过 RAML 建模工具实现 API 接口的完整定义，并基于 JAX-RS 规范实际编写了一个 REST 应用，包括 API 的外观层、数据访问层以及服务层的实现等，为我们提供了可应用于企业实际场景的标准应用示范，使得 RESTful API 不再是虚无的概念描述，而是可真正用于企业实践的架构风格实现。

书中很多内容都给译者留下了深刻印象，首先在构建 RESTful API 的基础 URL 时，我们需要从原有的“动词+名词”的设计向“名词+HTTP 动词”的观念转变，使得基础 URL 简单而又直观。其次在 API 组合的管理中，我们需要解决稳定高效的 API 服务提供与企业创新(和实验)之间的矛盾，因此我们需要通盘考虑 API 的一致性、可重用、版本和变更管理等。此外，企业级应用需要关注 JavaScript 跨域访问解决方案、OAuth 2 协议的安全性保证以及多种缓存机制等。

如果希望真正透彻掌握 RESTful API 的设计理念和实际应用，译者建议大家主动完成每章的程序练习，代码其实是最好的老师。每章最后一节都包含详细丰富的环境设置和代码，使得我们更容易理解和掌握 RESTful API 的精髓。

最后要感谢清华大学出版社能给我这次宝贵的翻译机会，把自己关于 RESTful API 的一些学习和理解在本书中与大家分享。尤其是要感谢本书的编辑，在本书的翻译过程中付出了很多的心血和努力，非常感谢他们的帮助和鼓励。另外感谢妻子以及我的家人，感谢你们一直以来对我的包容和理解；本书也献给我未来的孩子，希望你会喜欢老爸的这份礼物。

翻译英文书籍涉及多个概念的中文释义，我每次都力求与维基百

科、搜狗百科、百度百科等保持一致。鉴于译者水平有限，错误或瑕疵在所难免，如果对书中有任何意见或建议，也欢迎大家批评指正，我将感激不尽！本书全部章节由郭理勇翻译并整理，参与翻译的还有杨小琼、贺珊珊、王永胜等。

希望这本书能有力推动业界对 RESTful API 的统一定义和应用，更好地构建统一、高效、可扩展的分布式系统应用。

作者简介

Sanjay Patni 是一位注重实际成果的技术专家，在创新技术方案与业务实际需求的协调上具有丰富的经验，长期致力于企业业务流程的优化和运营效率的提升。

在过去五年中，他一直在 Oracle 公司的 Fusion Apps 产品研发团队任职，在那里他发现了对 Fusion Apps 代码管理实现自动化的机会，其中不仅涉及 GA 版本的交付发行，还包括正在进行的演示、开发和测试代码。他提出并开发了自助服务 UX 用于代码请求和审核，减少了 80%的手工步骤。他还发起了 12 次代码快速迭代，通过使用工作流和 RESTful API 等自动化技术与其他子系统进行集成，使得大约 100 多个手工步骤实现了自动化。

在加盟 Oracle 前，他已经在软件行业工作了 15 年以上，为不同的行业提供关键技术解决方案。他的职责包括对基于 Web 的企业级产品和解决方案提供技术创新、需求理解和分析，技术架构设计，以及推进软件敏捷开发等。他率先创新使用 Java 来构建业务应用，不断推动和完善用于企业级业务应用构建的 Java API，并获得 Sun Microsystems 公司颁发的奖项。

Sanjay 曾担任 RESTful API 设计和集成培训或课程的客座讲师、技术导师。他拥有强大的计算机科学教育背景，硕士毕业于印度理工学院(IIT)。

技术审稿人简介

Massimo Nardone 拥有超过 22 年的安全、Web/移动开发、云计算和 IT 设施等领域的丰富经验，对网络安全和 Android 有着狂热的技术激情。

在过去 20 多年，他一直致力于编程开发和教学，包括 Android、Perl、PHP、Java、VB、Python、C/C++、MySQL 等，拥有意大利萨莱诺大学计算机科学专业的硕士学位。

他曾经担任项目经理、软件工程师、研究员、首席安全架构师以及信息安全经理等，同时作为 PCI(国际安全标准)/SCADA(数据采集与监视控制系统)审计员，还是 IT 安全/云/SCADA 高级架构师等。

他的技术栈包括安全、Android、Cloud、Java、MySQL、Drupal、Cobol、Perl、Web 和移动开发、MongoDB、D3、Joomla、Couchbase、C/C++、WebGL、Python、Pro Rails、Django CMS、Jekyll、Scratch 等。

他目前担任 Cargotec Oyj 公司的首席信息安全官，曾任赫尔辛基理工大学(Aalto University)网络实验室的访问讲师和主管，拥有四项国际专利(PKI、SIP、SAML 和 Proxy 领域)。

Massimo 已为不同的出版公司审阅了 40 多种 IT 类图书，同时是 *Pro Android Games* 的作者之一(Apress, 2015)

前　言

众所周知，数据库、网站以及业务应用之间都需要数据交换。这通过定义标准的数据格式、传输协议或 Web 服务来实现，常见的数据格式有 XML(Extensible Markup Language，可扩展标记语言)、JSON(JavaScript Object Notation，JavaScript 对象表示法)等，常见的传输协议或 Web 服务包括 SOAP(Simple Object Access Protocol，简单对象访问协议)，以及目前更受欢迎的 REST(Representational State Transfer，表述性状态传递)等。开发人员通常需要设计自身应用的 API 接口，使得应用能集成特定的业务逻辑并运行在操作系统或服务器上。本书涵盖以上数据交换概念和通用的数据格式，并重点阐述如何构建 REST 风格的 API。

对于 Web 系统的交换来说，你将学习 HTTP 协议，包括如何使用 XML。另外本书还比较了 SOAP 和 REST，介绍无状态转移的概念。同时介绍软件 API 设计和最佳实践等。本书后半部分将重点讨论遵循 JAX-RS 标准的 RESTful API 的设计和实现，以及通过 Java API 构建 RESTful Web 服务。你将学习如何使用 JSON 和 XML 构建和使用 JAX-RS 服务，并通过实际案例使用 RESTful API 将众多不同的数据源集成在一起(包括关系型数据库和 NoSQL 数据库等)。你将应用这些最佳实践完成一个小型软件系统 API 的设计与实现，并以 RESTful API 的方式公开可用的 API 服务。

本书适用于那些在实际项目中使用数据交换的软件开发人员，对那

些希望了解数据交换方法以及如何与业务应用交互的数据专家同样有所帮助。书中的案例练习要求读者具有 Java 编程经验。

本书的主题包括：

- 数据交换和 Web 服务
- SOAP 与 REST，有状态与无状态
- XML 与 JSON
- API 设计简介：REST 和 JAX-RS
- API 设计实践
- 设计 RESTful API
- 构建 RESTful API
- 与 RDBMS(MySQL)进行交互
- 使用 RESTful API(比如 JSON、XML)
- API 安全性-OAuth
- API 缓存

源代码下载

读者可访问 www.apress.com/9781484226643 下载源代码，也可扫描本书封底的二维码直接下载。

目　　录

第 1 章

RESTful API 的基本原理

API 已经不是新兴事物了。近几十年间，API 一直作为接口使得应用之间可以相互通信。但 API 的作用在过去几年中发生了巨大变化。越来越多的创新型公司发现，通过为业务伙伴提供 API 接口可利用数字资产获利，此外，还可通过为合作伙伴提供更多功能来扩展价值主张，以及通过多种终端和设备来实现与客户的连接。创建 API 意味着：允许所在组织内或组织外的其他人，利用你的服务或产品来创建新应用、吸引客户或拓展业务。

其中内部 API 可通过在新应用中最大化重用性和增强一致性，来提升开发团队的生产效率。公共 API 可通过允许第三方开发人员增强你的服务或带来新客户从而使你的业务增值。一旦开发人员能利用你的服务和数据开发出新应用，就会形成一种网络效应，从而对底层业务产生显著影响。例如，Expedia 通过 API 向合作伙伴开放旅行预订服务来建立 Expedia 联盟网络，给公司带来了新的收入增长点，目前每年能产生 20 亿美元的收益。再如，Salesforce 通过发布 API 使得合作伙伴能够扩展其平台的功能，这些基于 SOAP(JAX-WS)以及最近的 RESTful(JAX-RS)的 API 收益已经贡献了公司年收入的半壁江山。

最初使用的 SOAP Web 服务依赖许多技术(如 UDDI、WSDL、SOAP、HTTP)和协议，用于在服务提供者和消费者之间传输和转换数据。可使用 JAX-WS 创建 SOAP Web 服务。

后来，Roy Fielding 于 2000 年发表了他的博士论文 *Architectural Styles and the Design of Network-based Software Architecture*。他创造了 REST 一词，一种分布式超媒体系统的架构风格。简而言之，REST(表述性状态传递，Representational State Transfer 的缩写)是一种有助于创建和组织分布式系统的架构风格。这个定义中的关键词应该是“风格”，因为 REST 很重要的一个方面(也是撰写本书的一个主要原因)，在于 REST 是一种架构风格，而不是一个准则、一个标准，也不是为最终形成 RESTful 架构而需要遵循的一系列硬性规则。

本章详细介绍 REST 基本原理、SOAP 与 REST 的比较以及 Web 架构风格。

总之，以 REST 风格组织的分布式系统将在以下几个方面得到改进：

- **性能**：REST 提出的通信方式是高效简单的，可在采用它的系统上实现性能提升；
- **组件交互的可扩展性**：任何分布式系统都应能很好地处理这方面的问题，而 REST 提出的简单交互极大地满足了这一点；
- **接口的简单性**：简单接口允许简化系统之间的交互，反过来又可提供上述优点；
- **组件的可变性**：系统的分布性质以及 REST 提出的关注点分离概念(稍后再次讨论)，可使组件以最低成本和风险彼此独立地进行修改；
- **可移植性**：REST 与技术和语言无关，这意味着可通过任何类型的技术来实现和使用它(存在一些限制，稍后将谈到，但不涉及具体技术)；
- **可靠性**：REST 提出的无状态(stateless)约束(见稍后的讨论)使得系统在发生故障后更容易恢复；
- **可见性**：重述一次，REST 提出的无状态(stateless)约束增加了所述请求的完整状态(稍后会谈到这些约束，到时大家就明白了)。从这个列表中可推断出 REST 的一些直接优点。以组件为中心的设计使得系统容错性非常好。单个组件的故障不影响系统的整体稳定性，这对于任何系统来说都是一个很大的优点。另外

互连组件是非常容易的，它可最大限度地减少添加新功能或系统弹性伸缩时的风险。由于 REST 的可移植性(如前所述)，以 REST 为基础设计的系统将更受大众的欢迎，具有通用接口的系统可被更广泛的开发人员使用。为实现这些属性和优点，REST 添加了一组约束以帮助定义统一连接的接口。但如果需要在客户端和服务器之间执行严格的交互协议或者执行涉及多个调用的事务，不建议使用 REST。

1.1　SOAP 和 REST 的比较

表 1-1 针对 SOAP 和 REST 各自支持的使用场景进行了全方位比较。

表 1-1　SOAP 和 REST 的比较

主题	SOAP	REST
起源	SOAP(简单对象访问协议)是 1998 年由 Dave Winer 等人与微软合作时提出的 该协议由大型软件公司开发，旨在满足企业级市场服务需求	REST(表述性状态传递)是由加州大学尔湾分校的 Roy Fielding 于 2000 年提出的 这个概念诞生于学术环境，涵盖了开放式网络的理念
基本概念	使数据可用作服务(动词+名词)，例如 getUser 或 PayInvoice	使数据可用作资源(名词)，例如 user 或 invoice
优点	SOAP 遵循正式的企业规范； 可在任何通信协议上运行，甚至是异步方式； 关于对象的信息需要传递给客户； 安全性和授权也属于 SOAP 协议的一部分； 可使用 WSDL 语言进行完全描述	REST 遵循开放式网络理念； 容易实施和维护； 明确分离客户端和服务器的实现； 通信不由单个实体控制； 客户端可存储信息，以防多次调用； REST 可用多种格式返回数据(如 *JSON*、*XML* 等)
缺点	SOAP 花费大量带宽来传递元数据； SOAP 实现较困难，在 Web 和移动应用开发人员中并不受欢迎	REST 仅在 HTTP 协议之上运行； 很难仅在 REST 之上执行授权和安全性

(续表)

主题	SOAP	REST
适用场景	客户端需要访问服务器上可用的对象； 在客户端和服务器之间执行正式的协议	客户端和服务器在 Web 环境中运行； 关于对象的信息不需要传递到客户端
不适用的场景	如果广大开发人员希望能够轻松使用 API，SOAP 不太容易满足； SOAP 在带宽非常有限的场景中也不太适用	需要在客户端和服务器之间执行严格的协议时，REST 不适用； 执行涉及多个调用的事务时，REST 也不适用
使用实例	金融服务； 支付网关； 电信业务	社交媒体服务； 社交网络； 网络聊天服务； 移动服务
示例	https://www.salesforce.com/developer/docs/api/-Salesforce SOAP API https://developer.paypal.com/docs/classic/api/PayPalSOAPAPIArchitecture/Paypal SOAP API	https://dev.twitter.com/ https://developer.linkedin.com/apis
小结	如果正处理事务性操作，并且已经有对 SOAP 技术满意的受众群体，可使用 SOAP	如果专注于大规模采用 API 或你的 API 针对移动应用，请使用 REST

1.2 Web 架构风格

根据 Fielding 博士所述，可采用以下两种方式来定义系统：

- 第一种方式是从一个空白的白板开始。采用这种方式时，直到系统需求均被满足时，才能对正在构建的系统有初步了解或熟悉组件的使用；
- 第二种方式是从系统的全套需求开始。为各个组件添加约束，直至影响系统的各部分能彼此协调地进行交互。

REST 采用第二种方式。为定义 REST 架构，最初会定义一个空状态，这个空状态是一个没有任何约束条件和组件区分的系统，之后会逐

一对其添加约束条件。下面将介绍 Web 应用架构风格的约束条件。每个约束条件定义了应如何构建和设计 REST API 框架。当我们向最终用户交付 RESTful API 时，另一个需要单独考虑的方面是安全性。安全性也是该框架的一部分。

1.2.1　CS 模式

关注点分离是 Web 客户端-服务器模式(CS 模式)约束条件的核心主题。Web 是基于 CS 模式的系统，客户端和服务器在其中发挥着不同的作用。

只要客户端和服务器符合 Web 的统一接口，它们就可以使用任何语言或技术独立地实现和部署。

1.2.2　统一资源接口

Web 组件之间的交互依赖它们接口的一致性。Web 组件包括客户端、服务器和基于网络的中间件等。

Web 组件使用统一接口实现一致的交互操作，建立在 Fielding 博士定义的以下四个约束条件之上：

- **源标识的唯一性**：每个资源的资源标识可用来唯一地标明该资源；
- **资源的自描述性**：一个 REST 服务所返回的资源需要能够描述自身，并提供用于操作该资源的足够信息；
- **消息的自描述性**：每条消息都包含足够的信息来描述如何处理该消息；
- **超媒体驱动性(HATEOAS)**：一个典型的 REST 服务不需要额外的文档来标识通过哪些 URL 访问特定类型的资源，而是通过服务端返回的响应来标识到底能在该资源上执行什么操作。

1.2.3　分层系统

一般来说，基于网络的中间件会为了某些具体目的拦截客户端和服务器端之间的通信，通常用于安全性增强、缓存响应和负载平衡等。

分层系统的约束条件使得基于网络的中间件(如代理和网关)可通过使用 Web 的统一接口，透明地部署在客户端和服务器之间。

1.2.4 缓存机制

缓存机制是 Web 架构最重要的约束条件之一。缓存约束指 Web 服务器需要具备对每个响应数据的缓存能力。

缓存响应数据有助于减少客户端感知的延迟，提高应用的总体可用性和可靠性，以及控制 Web 服务器的负载。总之，缓存机制降低了 Web 的总体成本。

1.2.5 无状态

无状态约束条件规定 Web 服务器不需要记住其客户端应用的状态。因此，在每次与 Web 服务器进行交互时，客户端必须包含其认为与该次交互相关的所有上下文信息。

Web 服务器要求客户端去管理应用状态通信的复杂性，从而使得 Web 服务器能为更多客户端提供服务。这种权衡实际上是 Web 架构风格具有高可扩展性的一个关键因素。

1.2.6 按需编码

Web 大量使用按需编码。按需编码这一约束条件使得 Web 服务器可暂时将可执行程序(如脚本或插件)转移到客户端。

按需编码试图建立 Web 服务器和客户端之间的技术耦合，因为客户端必须能够理解并执行从服务器按需下载的代码。出于这个原因，按需编码是 Web 架构风格的唯一非必选的约束。

1.2.7 HATEOAS

REST 的最后一条原则是使用超媒体驱动性(Hypermedia As The Engine Of Applications State，HATEOAS)。当通过使用 HATEOAS 开发 CS 模式的解决方案时，服务器端的逻辑改变可独立于客户端。

超媒体是一种以文档为中心的方法，它支持在文档格式中嵌入其他服务和信息的链接。超媒体和超链接的用途之一是将不同来源的信息合成复杂的信息集。这些信息可来自公司私有云或不同来源的公有云。

例如：

```
<podcast id="111">
    <customer>http://customers.myintranet.com/customers/1
    </customers>
    <link>http://podcast.com/myfirstpodcast</link>
    <description> This is my first podcast </description>
</podcast>
```

每种 Web 架构风格都为 Web 系统增添了有益的特性。通过采用这些约束条件，开发团队可建立简单、可见、可用、可访问、可演化、灵活、可维护、可靠、可扩展和高性能的系统，如表 1-2 所示。

表 1-2　约束条件和系统特性

约束条件	系统特性
C/S 模式交互	简单、可演化、可扩展
无状态通信	简单、可见、可维护、可演化、可靠
缓存数据	可见、可扩展、高性能
统一接口	简单、可用、可见、可访问、可演化、可靠
分层系统	灵活、可扩展、可靠、高性能
按需编码	可演化

1.3　安全性

本章没有把安全性作为 REST 基本原理的一部分，但安全性对于交付 RESTful API 是非常重要的。本书将用完整一章来详细阐述 RESTful API 的安全性，包括如何使用 OAuth 协议确保 RESTful API 安全性的最佳实践的相关细节，OAuth 目前已成为 RESTful API 安全性的标准。

1.4 什么是 REST?

前一节简要介绍了 REST 与 REST API 的基本原理。本节将介绍有关 REST 概念的更多详细信息。

REST 是由 Roy Fielding 博士在博士论文中提出的，用于描述实现网络系统的设计模式。REST 的字面解释是表述性状态传递(REpresentational State Transfer)，是一种设计分布式系统的架构风格。它不是一个标准，而是一组约束条件。它不强行绑定于 HTTP，但通常都与 HTTP 相关联。

1.4.1 REST 基础知识

与 SOAP 和 XML-RPC 不同，REST 并不需要新的消息格式。我们知道 HTTP API 包括 CRUD(创建、检索、更新、删除)等。

- GET =“给我一些信息”(检索)；
- POST =“这是一些更新信息”(更新)；
- PUT =“这是一些新信息”(创建)；
- DELETE =“删除一些信息”(删除)；
- 还有更多……；
- PATCH = PATCH 方法可用于更新部分资源。例如，当只需要更新资源的一个字段时，PUT 完整的资源这种表述可能会很麻烦而且会占用更多带宽；
- HEAD = **HEAD** 方法与 GET 方法相同，不同之处在于服务器并不会在响应中返回消息体。HEAD 方法常用于测试超文本链接的有效性、可访问性和最近的修改情况；
- OPTIONS = OPTIONS 方法允许客户端确定与资源相关的选项或需求，或者服务器的容量，但这并不意味着资源操作或者初始化资源检索；
- “幂等性”概念——当向系统发送 GET、DELETE 或 PUT 方法时，不论该方法发送多少次，执行效果应该是一样的，但 POST 方法在集合中创建了实体，因此不是幂等的。

1.4.2　REST 基本原理

据 ProgrammableWeb.com 网站 2016 年的数据，大约有 8356 个 API 是用 REST 写的。REST 基于资源架构，资源可通过基于 HTTP 标准方法的通用接口访问。REST 要求开发人员显式使用 HTTP 方法，并以符合协议定义的方式使用。每个资源都有一个 URL 标识，且都应该支持 HTTP 的通用操作，同时 REST 允许该资源有不同的表现形式，例如文本、XML、JSON 等。REST 客户端可通过 HTTP 协议(内容协商)请求特定的表现形式。表 1-3 描述了 REST 中使用的数据元素。

表 1-3　REST 结构

数据元素	描述
资源	超文本引用的概念目标，例如 customer/order
资源标识符	统一资源定位符(URL)或统一资源名称(URN)标识特定的资源，例如 http://myrest.com/customer/3435
资源元数据	描述资源信息，如标签、作者、源链接、替代位置、别名
表现形式	资源内容——JSON 消息、HTML 文档、JPEG 图片
表现元数据	描述如何处理表现的信息，如媒介类型、最后修改时间
控制数据	描述如何优化响应处理的信息，例如 if-modified-since、cache-control-expiry

让我们看一些例子：

1. 资源

首先，GET 播客(podcast)列表的 REST 资源：

```
http://prorest/podcasts
```

其次，获取 podcast id= 1 的 REST 资源的详细信息：

```
http://prorest/podcasts/1
```

2. 表现形式

以下是通过 id 获取客户响应信息的 XML 表现形式：

```
<Customer
>     <id>123</id>
>     <name>John</name>
</Customer>
```

以下是通过 id 获取客户响应信息的 JSON 表现形式：

```
{"Customer":{"id":"123","name":"John"}}
```

3. 内容协商

HTTP 天然支持基于 HTTP 协议头来告知服务器你所期望的内容和所能处理的内容。基于此机制，服务器将以正确格式返回相应内容，见图 1-1。

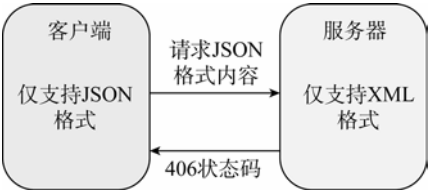

图 1-1　内容协商

如果服务器不支持请求的格式，那么根据相关规范，服务器将返回 406 状态码(不接受)以通知提出请求的客户端(即“请求的资源只能根据请求中包含的 Accept 头部生成内容，若不能满足，则返回表示不接受的状态码(406)”)。

1.5　小结

REST 指出了 Web 普及和可扩展的关键架构原则。Web 推广和教育的下一步工作是将这些原则应用到语义 Web 和 Web 服务领域中。REST 提供了一种简单、可互操作、灵活的方式来编写 Web 服务，这种方式与 WS - *等众多曾使用的方式迥异。下一章将更详细地介绍这些概念。