PEARSON

SOA with REST

Principles, Patterns & Constraints for Building Enterprise Solutions with REST

“SOA可以通过许多不同的方法实现，而REST则是潜在的实现框架中最闪亮的新方法。本书向架构师和开发人员介绍了实现RESTful SOA所需的知识，而最重要的是，它告诉人们一种通过REST实现SOA的风格：其核心是设计服务生态系统，在其中向客户提供简单的使用资源的方式，并通过服务将资源连接起来。本书无疑将有助于使SOA从Web架构的主要价值主张（分散、松耦合、连通性、自描述服务、独立于实现的服务接口）中获益。”

——Erik Wilde博士，EMC公司架构师

“面向服务和REST这两种架构风格都是现代应用程序和云计算的基石。它们都致力于交付可伸缩的、可互操作的解决方案，但是它们的不同根基使得它们并不能天然地互相配合。本书阐述了如何在企业环境中使二者协调工作。书中讨论了一组设计流程，它们使服务集合在满足SOA目标的同时又符合现有的REST约束。此外，为使REST风格满足企业级需求，它还务实地在必要之处放松了约束。”

——Christoph Schittko，微软公司云战略总监

SOA与REST

用REST构建企业级SOA解决方案

〔美〕Thomas Erl　Benjamin Carlyle
Cesare Pautasso　Raj Balasubramanian　著
马国耀　申健　刘蕊　译

人民邮电出版社
POSTS & TELECOM PRESS

PEARSON

SOA与REST

用REST构建企业级SOA解决方案

SOA with REST

Principles, Patterns & Constraints
for Building Enterprise Solutions with REST

Thomas Erl　Benjamin Carlyle
〔美〕Cesare Pautasso　Raj Balasubramanian　著
马国耀　申健　刘蕊　译

人民邮电出版社
北京

图书在版编目（CIP）数据

SOA与REST : 用REST构建企业级SOA解决方案 / (美)埃尔 (Erl,T.) 等著 ; 马国耀, 申健, 刘蕊译. -- 北京 : 人民邮电出版社, 2014.1
书名原文: SOA with REST: Principles, patterns & constraints for building enterprise solutions with REST
ISBN 978-7-115-33194-6

Ⅰ. ①S… Ⅱ. ①埃… ②马… ③申… ④刘… Ⅲ. ①互联网络－网络服务器－研究 Ⅳ. ①TP368.5

中国版本图书馆CIP数据核字(2013)第225519号

内 容 提 要

SOA 与 REST 是当前两种流行的技术架构风格。然而，二者却站在不同的层次看架构，SOA 的角度偏向于战略；而 REST 的角度则偏向于战术。SOA 给出了一组架构原则实现其战略目标，而 REST 则通过一系列约束实现其战术目标。

本书深入介绍了 SOA 与 REST 的原理、术语及特性；深入阐述了二者之间的差异及合作点；重点阐述了如何将 REST 作为媒介来实现 SOA 的战略目标，通过对 REST 服务的建模流程和专为 REST 服务定制的面向服务的分析和设计流程的详细讲解，逐步向读者展开了一幅 REST 与 SOA 在企业级解决方案中完美“联姻”的画卷。此外，本书还通过完整的案例研究示例展示了 REST 与 SOA 在实践中的结合。

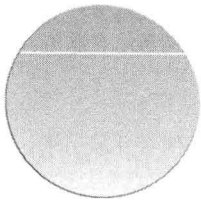
本书适合于考虑实施面向服务架构的开发人员、架构师或项目经理阅读参考，尤其适合任何 SOA 实践者或任何计划发起一个 SOA 项目的专业人员。

◆ 著 [美] Thomas Erl Benjamin Carlyle
Cesare Pautasso Raj Balasubramanian
译 马国耀 申 健 刘 蕊
责任编辑 杨海玲
责任印制 程彦红 杨林杰
◆ 人民邮电出版社出版发行 北京市丰台区成寿寺路 11 号
邮编 100164 电子邮件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
北京艺辉印刷有限公司印刷
◆ 开本：800×1000 1/16
印张：23.75
字数：523 千字 2014 年 1 月第 1 版
印数：1－2 500 册 2014 年 1 月北京第 1 次印刷

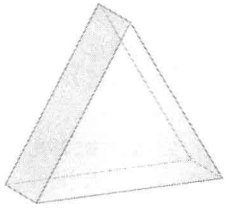
著作权合同登记号 图字：01-2013-1132 号

定价：89.00 元

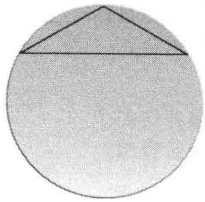
读者服务热线：(010)81055410 印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京崇工商广字第 0021 号



服务契约
（带弦的圆符号）

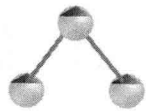


统一契约



通过统一契约
（带弦的圆符号）
访问的服务契约

REST服务
（带标签的）



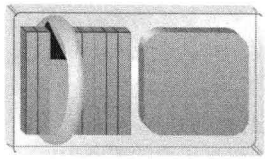
REST服务组合

组件或程序

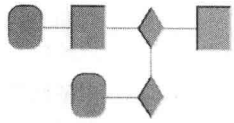
解耦的服务
契约

通过统一契约
访问的解耦的
服务契约

服务
代理



使用统一契约
的REST服务



处理逻辑

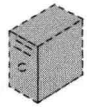
WSDL
定义

XML模式
定义

一般的机器可
处理的文档

人可读的文档
或内容

物理
服务器

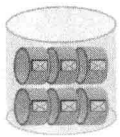


虚拟
服务器

防火墙

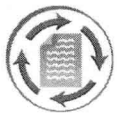
消息

安全元素或
加锁的资源



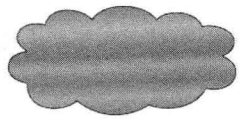
消息队列

仓库或注册

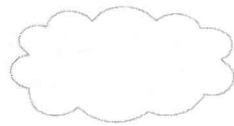


主动处理

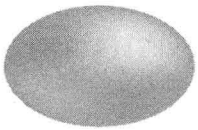
处理过程或项目/
生命周期阶段



互联网



云



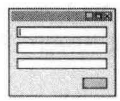
区域或地带

冲突的图标

转移箭头

人或角色

客户端
工作站

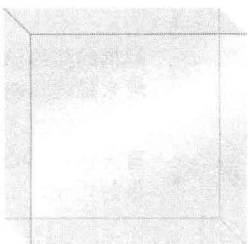


用户界面

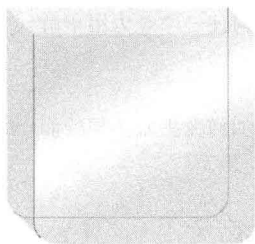
移动设备

产品或
系统

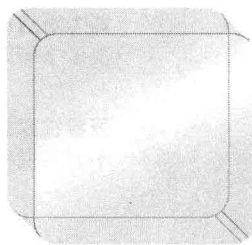
概念性关系图中
使用的图标



一般性物理边界



服务目录边界



服务边界

无关性能力[292] 如何使服务于多目标的服务逻辑得到有效地消费和组合？

无关性上下文[293] 如何将服务与多个目标的服务逻辑作为有效的企业资源？

无关性子控制器[293] 如何对无关性的、跨实体的组和逻辑进行拆分、重用和单独管理？

异步队列[293] 服务及其消费者之间如何独立处理失败，并且避免不必要的资源锁？

原子服务事务[294] 如何将事务回滚的能力延伸到基于消息传输的服务中？

中介认证[294] 当消费者与服务间不互信时，或当消费者需要访问多个服务时，服务如何有效地核实消费者的凭证？

规范表达[294] 如何使服务契约得到一致地理解与解析？

规范协议[295] 如何设计服务以避免协议桥接？

规范资源[295] 如何避免基础设施资源不必要的不一致？

规范模式[295] 如何设计服务以避免数据模型转换？

规范模式总线[296]

规范版本控制[296] 如何以最小的影响对同一服务目录中的服务进行版本控制？

能力组合[297] 当服务能力需要借助于服务边界之外的逻辑以解决问题时，怎么办？

能力重组[297] 如何使一个能力用于解决多个问题？

兼容性变更[297] 如何变更服务契约而不影响现有消费者？

补偿服务事务[298] 在服务无需锁住资源的情况下，如何一致地调解组合服务的运行时异常？

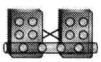
组合自治[298] 如何实现组合并最小化自治损失？

并发契约[298] 如何使服务同时支持多个消费者的耦合需求及抽象关注？

内容协商[299] 如何使服务能力适应于携带不同数据格式或表述需求的服务消费者？

契约集中化[299] 如何避免消费者对于服务实现的直接耦合？

契约去规范化[299] 服务契约如何支持消费者程序的不同数据交换需求？



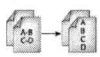
跨域实用功能层[300] 如何避免在跨领域的服务目录中出现冗余的公用功能逻辑？

数据保密性[300] 如何保护消息中的数据，使之在传输过程不会泄露给非目标接收者？

数据格式转换[300] 服务如何与带有不同数据格式的程序交互？

数据模型转换[301] 当服务为同一数据类型使用不同的数据模型时，如何进行互操作？

数据来源认证[301] 服务如何核实消息来源于一个已知发送者，并且在传输过程中未被篡改？



分解的能力[301] 如何设计服务，使其能力逻辑拆解的可能性最小化？

解耦的契约[302] 如何独立于服务的实现来描述其能力？

直接认证[302] 服务如何核实消费者提供的凭证？

此为试读，需要完整PDF请访问：www.ertongbook.com

分布式能力[302] 服务如何在保持其功能上下文的同时完成特殊的能力处理需求？

域目录[303] 在不可能确定企业范围标准的情况下，如何交付服务以实现重组最大化？

双重协议[303] 服务目录如何在保持标准化的同时克服规范协议的限制？

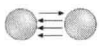
端点重定向[303] 当特定服务端点变更或移除时，消费者如何适应？

企业目录[304] 如何交付服务以最大限度地支持重组？

企业服务总线[304]

实体抽象[305] 如何对无关性业务逻辑进行独立地划分、重用及治理？

实体链接[305] 服务如何暴露其业务实体间的内在关系以支持松耦合组合？



事件驱动消息机制[306] 如何自动向服务消费者通知运行时的服务事件？

异常防护[306] 当异常发生时，如何防止服务内部实现信息的泄露？

联邦端点层[307]

文件网关[307] 对于仅能通过文件交换来共享信息的遗留系统，服务逻辑如何与之交互？

功能分解[308] 在不建立独立的解决方案逻辑的前提下，如何解决大型业务问题？

幂等能力[308] 服务能力如何安全地接受相同消息的多份拷贝，从而处理失败的通信？

仲裁路由[308] 动态的运行时的因素如何影响消息的路径？

目录端点[309] 如何在向外部消费者提供服务能力的同时，保护服务目录免受外部访问？

遗留系统包装器[309] 如何防止带有非标准契约的包装器服务去传播消费者与服务实现的间接耦合？

轻量级端点[310] 如何将轻量级业务逻辑单元变成有效的、可重用的企业资源？

逻辑集中化[310] 如何避免冗余服务逻辑的误用？

消息筛查[310] 如何保护服务免受变形输入或恶意输入的侵害？

消息机制元数据[311] 如何设计服务使之在运行时处理特定于活动的数据？

元数据集中化[311] 如何集中发布和治理服务元数据？

多通道端点[312] 如何将因为交付渠道不同所造成的分散而重复的遗留逻辑集中并统一起来？

关联性上下文[312] 如何将单一目标的服务逻辑当作有效的企业资源？

正式端点[312]

编排[313]

部分状态延迟[313] 如何设计服务，在保留有状态的情况下优化资源消耗？

部分校验[314] 如何避免不必要的数据校验？

策略集中化[314] 如何规范策略，并使之跨多个服务一致地执行？

流程抽象[314] 如何拆分并独立治理关联性流程逻辑？

流程集中化[315] 如何集中治理抽象的业务流程逻辑？

协议桥接[315] 服务如何与使用不同通信协议的消费者交换数据？

代理能力[315] 当服务面临分解时，如何继续支持受此分解影响的消费者？

冗余实现[316] 如何提高服务的可靠性和可用性？

可靠消息机制[316] 当服务在不可靠的环境中实现时，如果保证其通信可靠性？

可重用契约[316] 服务消费者在进行服务组合时，如何使自己不与特定于服务的契约相耦合？

规则集中化[317] 如何抽象业务规则并集中治理它们？

模式集中化[317] 如何设计服务契约以避免冗余的数据表述？

服务代理[318] 如何拆分并单独治理事件驱动的逻辑？

服务中介[318]

服务回调[319] 服务如何与其消费者进行异步通信？

服务数据复制[319] 当服务需要访问共享的数据源时，如何保持服务自治？

服务分解[319] 在服务实现之后，如何细化服务粒度？

服务封装[320] 如何将解决方案逻辑作为企业资源提供出来？

服务门面[320] 服务如何在允许其核心服务逻辑独立演化的同时，适应契约或实现的变更？

服务网格[320] 如何分散延迟的服务状态数据并保持容错性？

服务实例路由[321] 如何使消费者与多个服务实例在无需私有处理逻辑的情况下进行联系和交互？

服务分层[321] 如何根据功能通用性组织目录中的服务？

服务消息机制[321] 服务如何在互操作的同时无需形成持久、紧耦合的连接？

服务规范化[322] 如何避免服务目录产生冗余的服务逻辑？

服务周边防护[322] 如何将运行在私有网络中的服务提供给外部消费者，同时不暴露内部资源？

服务重构[322] 如何在不影响现有消费者的情况下进行服务演化？

状态消息机制[323] 如何使参与有状态的交互的服务保持无状态性？

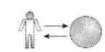
状态仓库[323] 如何在不消耗运行时资源的情况下，将服务的状态数据保持更长时间？

有状态服务[323] 如何在不消耗运行时资源的情况下，对服务的状态数据进行持久化和管理？

终止通知[324] 如何将计划中的服务契约到期通知到消费者程序？

三层目录[324]

受信子系统[324] 如何阻止消费者绕开服务而直接访问服务资源？



UI调解人[325] 面向服务的解决方案如何提供一致的、互动式的用户体验？

统一契约[325]

公用功能抽象[326] 如何对通用的非以业务为中心的逻辑进行划分、重用和独立治理？

检验抽象[326] 如何设计服务契约，使之更易于适应校验逻辑的变化？

版本标识[326] 如何使消费者感知到服务契约的版本信息？

版 权 声 明

Authorized translation from the English language edition, entitled: *SOA with REST: Principles, Patterns & Constraints for Building Enterprise Solutions with REST*, 9780137012510 by Thomas Erl, Benjamin Carlyle, Cesare Pautasso, and Raj Balasubramanian, published by Pearson Education, Inc., publishing as Prentice Hall, Copyright © 2013 Pearson Education, Inc.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc.

CHINESE SIMPLIFIED language edition published by PEARSON EDUCATION ASIA LTD. and POSTS & TELECOM PRESS Copyright © 2013.

本书中文简体字版由 Pearson Education Asia Ltd.授权人民邮电出版社独家出版。未经出版者书面许可，不得以任何方式复制或抄袭本书内容。

本书封面贴有 Pearson Education（培生教育出版集团）激光防伪标签，无标签者不得销售。

版权所有，侵权必究。

谨以此书献给 ChristophSchittko，他毫无顾忌的审阅评注使我们在本书书稿已经提交印刷之后仍然决定改变内容结构，而后续的变化极大地提升了书稿的质量。

——Thomas Erl

献给亲爱的妻子 Michelle，献给我的父母 Rob 和 Sue，献给我的孩子 Genevieve 和 Matthew。感谢你们多年来的支持与鼓励！

——Benjamin Carlyle

献给我的家人及我的 Esperanza。

——Cesare Pautasso

向父母的牺牲与支持致敬！

——Raj Balasubramanian

对本书的赞誉

“本书通过具体且实用的方式，阐释了 SOA 和 REST 领域之间的联系，简明地将其运用到日常遇到的架构挑战上。太棒了！”

——Ryan Frazier，技术战略师，微软公司

“SOA 可以通过许多不同的方法实现，而 REST 则是潜在的实现框架中最闪亮的新方法。本书向架构师和开发人员介绍了实现 RESTful SOA 所需的知识，而最重要的是，它告诉人们一种通过 REST 实现 SOA 的风格：其核心是设计服务生态系统，在其中向客户提供简单的使用资源的方式，并通过服务将资源连接起来。本书无疑将有助于使 SOA 从 Web 架构的主要价值主张（分散、松耦合、连通性、自描述服务、独立于实现的服务接口）中获益。”

——Erik Wilde 博士，架构师，EMC 公司

“这是一部杰作，它将 REST 原则优雅地运用到该丛书中的工业标准 SOA 框架上。书中为实践者提供了有用的指导，并且在形式和精神上都与 Roy Fielding 论文中定义的 REST 约束保持一致。有关 RESTful 契约设计的章节使本书物有所值。本书对于任何开发 REST 服务的人来说都是必读的。”

——Dave Slotnick，企业架构师，Rackspace Hosting

“面向服务模式的精彩大作，它将有效地解决现实世界里的问题。REST 观点和原则将完全地覆盖现代 Web 2.0 风格的方法。强烈推荐。”

——Sid Sanyal，IT 架构师，苏黎世金融服务公司

“REST 不仅仅是接口的另一种实现方法。本书为我们展示了服务组合生态系统如何随着服务组合架构设计的新机遇而变化。对于任何正在考虑 REST 风格服务构建应用程序架构的认真的 IT 架构师来说，这都是一本全面指南和必读之作。”

——Roger Stoffers，解决方案架构师，惠普公司

“面向服务和 REST 这两种架构风格都是现代应用程序和云计算的基石。它们都致力于交付可伸缩的、可互操作的解决方案，但是它们的不同根基使得它们并不能天然地互相配合。本书阐述了如何在企业环境中使二者协调工作。书中讨论了一组设计流程，它们使服务集合在满足 SOA 目标的同时又符合现有的 REST 约束。此外，为使 REST 风格满足企业级需求，它还务实

地在必要之处放松了约束。”

——Christoph Schittko，云战略总监，微软公司

“这是一本鼓舞人心的书，它为下一代基于 REST 的面向服务的系统的设计与开发带来了深刻见解。本书务实地阐述了 SOA 与 REST 的融合，解决了工作中常见的实际问题。对于软件设计师、架构师和顾问来说，是必要的‘现代服务实现的工具’和‘强大的知识体系’。”

——Pethuru Raj 博士，企业架构师顾问，Wipro 顾问服务公司

“Thomas Erl 的服务技术丛书一贯地使用简单的例子说明复杂的概念。在该丛书中的最新著作《SOA 与 REST》中，作者通过常见的 SOA 语言来讨论 REST。《SOA 与 REST》对企业架构师和开发人员来说都是极好的资源！”

——Kevin P. Davis，博士，软件架构师

“不同于其他相似内容的书籍，《SOA 与 REST》一书中的叙述做到了完善、易读，包含了现实世界的案例研究，可同时满足开发人员和分析师的需要。对于 SOA 实践者及任何计划启动 SOA 项目的执行者来说，这都是不可或缺的资料。”

——Theodore T. Morrison，认证的 SOA 分析师，CSM，Geocent，LLC

“任何将 REST 应用程序构建为面向服务架构的 IT 架构师或软件工程师，要想深入理解其中原则、模式和实现概念的话，就都需要读一读这本书。它不仅包含了基本的话题，还探讨了 REST 与各种特定的 SOA 原则及模式之间的关系。”

——Sanjay Singh，认证的 SOA 分析师，开发经理，NorthgateArinso

“一本面向企业架构师、分析师、开发人员的权威的上乘参考书。本书不仅展示了 REST 的优雅、简单性和通用性，还使我们清楚地理解了 REST 是如何增强 SOA 和面向服务的，REST 如何能够影响 SOA 设计目标，我们如何来设计和开发 REST 服务，我们如何解决 REST 集成到面向服务时所面临的独特挑战。任何以 REST 来构建面向服务架构的人，想要掌握这门技术，都有必要阅读本书。”

——Philip Wik，MSS Technology

“这是一本理解如何在面向服务架构中采用 REST 的基础而全面的书。对于任何对面向服务感兴趣的实践者来说，书中提供的许多示例和模式将是非常宝贵的资源。”

——Gustavo Alonso，计算机科学系，苏黎世联邦理工学院

“SOA 和 REST 是分布式计算中两种非常重要的架构风格。SOA 成功地在大多数企业中得到采用，而研究者和工业用户越来越多地关注 REST 风格。《SOA 与 REST》一书介绍了一种新

此为试读，需要完整PDF请访问：www.ertongbook.com

的架构风格，巧妙地结合了 SOA 和 REST 风格，清晰地揭示了两者的协同工作，通过 REST 来产生成功的企业 SOA 策略，以及对架构设计决策提出指导。本书是使用 REST 来设计和实现 SOA 架构的最佳实践的圣经。这是一本 IT 实践者和研究人员的必读书籍。”

——Longji Tang，联邦快递 IT 高级技术顾问，CSSE 博士

“REST 和 SOA 是过去十年间在软件工业中被误解最多的两个术语。然而 REST 架构风格加上现代 RESTful 框架实现，提供了可伸缩和可靠的 SOA 方式。本书涵盖了关于如何将 REST 原则应用到小型和大型 SOA 开发中的全面阐述。如果你已经熟悉 REST 并在考虑 SOA，那么你需要本书。如果你还没有在你的 SOA 工作中考虑 REST，那么本书同样适合你。它囊括了 REST 和 SOA 的概念，还包含了设计模式与使用的时机，本书是架构师和工程师的精彩指南和优秀工具。”

——Mark Little 博士，JBoss 首席技术官，红帽公司

“本书精彩地介绍了如何将 SOA 方法论与 RESTful 架构风格的服务结合起来。对于 SOA 架构师如何更好地理解将 REST 集成到面向服务架构流程的含义和要求，Thomas Erl 及其合著者们提供了很大的帮助。”

——Gerald Beuchelt，MITRE

译者序

如今，在企业级解决方案领域，SOA 这一词汇已不再陌生。几十年来，为了打通不同时期使用不同技术建设的信息孤岛之间的连接，人们一直在改进软件工程方法，也尝试过很多不同的技术手段，比如点对点（P2P）的集成、基于消息的集成（Messaging）、企业应用集成（EAI）等，它们在各自的历史舞台上发挥了重要的作用。然而，直到面向服务思想和面向服务架构的诞生，这一问题才真正得到有效解决。

与 SOA 不同，REST 是在互联网环境中成长起来的，根据 Fielding 博士的论文，REST 是为了解决互联网规模的软件架构所面临的问题而诞生的。尤其，当 REST 与 HTTP 标准结合起来，通过 RESTful Web 服务本身的优势，如无状态性和幂等性，为提高系统的可伸缩性起到重要作用。REST 以其简洁性、更低的资源占用，统一接口抽象、代理服务器支持、缓存服务器支持等诸多方面的优势深受众多架构师和开发者的喜爱。

作为两种优秀的架构风格，它们都不缺乏推崇者。自然地，二者之间的论战逐渐成了技术圈的热点话题。网络上关于 SOA 与 REST 的论战非常之多，但是，它们大多聚焦于二者之间的差异，有的 SOA 的拥护者称 REST 不适合企业级应用，而 REST 的拥护者则把对 SOAP 协议的抱怨转嫁到 SOA 的身上。

Thomas Erl 站了出来。他说，两种架构风格的目标有诸多相同之处，即便在企业级解决方案中，SOA 和 REST 也能够相得益彰，通过“战术性”的 REST 媒介实现“战略性”的 SOA 的战略目标。Thomas Erl 是 SOA 领域的大师级人物，他在 SOA 方面的诸多著作，尤其是其“SOA 系列著作”，为架构师和开发者们提供了丰富的理论知识和实践指引。

本书全面介绍了面向服务概念、REST 约束及目标，并且针对企业应用所面临的问题提出了用 REST 来实现 SOA 的设计目标和架构原则。作者提出了在分析和建模过程中的注意事项，并启发出一些新的 SOA 设计模式。

作为本书的译者，我们为能够翻译这样一本优秀的著作而感到荣幸。尽管我们各自都有许多工作和事情要处理，但是我们尽量利用下班后和周末的时间，兢兢业业地翻译这本书，不敢有一丝疏忽和怠慢，生怕由于自己的失误而未能准确地表达作者们的原意。

我们从 2012 年 9 月开始翻译此书，翻译分工如下。

- 马国耀负责翻译第 4、6、7、14、15、16 章及附录 E，这些章节的内容包括：SOA 术语和概念，REST 服务契约，REST 与面向服务的关系，受 REST 启发的设计模式，REST 服务版本控制，统一契约介绍，SOA 设计模式概要等。
- 申健负责翻译第 2、5、11、12、13 章及部分附录，这些章节的内容包括：案例研究背

景，REST 约束与目标，REST 服务组合，REST 高级服务组合及案例研究等。

- 刘蕊负责翻译第 1、3、8、9、10 章及部分附录，这些章节的内容包括：内容导读，服务基础知识，主流 SOA 方法论及 REST 技术，REST 服务的分析与建模，REST 面向服务的设计等。

在翻译过程中，我们采用迭代方法，每人各翻译完一个章节之后就进行一次审校迭代。我们采取 A 审校 B，B 审校 C，C 审校 A（或相反顺序）的方式进行审校，所有意见不一致的地方由定期会议仲裁解决。全书最后由马国耀进行统一修订和审校。

由于译者的能力水平有限，难免存在疏忽与差错，在此恳请读者见谅，并给予批评指正。

翻译是个痛苦而辛苦的过程，翻译《云计算与 SOA》的经历让我感触很深。另外，由于自己工作上进入一个新的平台以及刚出生的女儿的缘故，根本没有太多空余时间来翻译。所以，我本不打算翻译这本书的。但是，在人民邮电出版社杨海玲女士的邀请和李锟先生的鼓励之下，我快速阅读完了本书，在此之后我就再无丝毫犹豫。因为，不将本书呈现给国内的读者和广大 SOA 从业者，我觉得是一个遗憾。于是，就有了连续一年翻译历程。在这个过程中，我结识了申健、刘蕊两位认真而专业的译者，这一年来的沟通与交流是难忘而深刻的，感谢你们。最后，感谢家人对我一贯的支持，特别感谢我的妻子和女儿，因为翻译这本书所用的时间正是原本应该用来陪伴你们的时间。

——马国耀

回首过去忙碌的一年，看到自己在企业内完成了手机银行解决方案的重要演进，辅导了更多的敏捷团队，在企业外结识了更多的朋友，锻炼了更多的技能，并在而立之年迈出了职业生涯中新的一步。同时，看到天津软件社区变得热闹起来，南京大学天津校友会愈发壮大，女儿的轮滑也练得有模有样了。最重要的，就是有幸与马国耀、刘蕊两位战友共同完成了这本书的翻译。三个人素未谋面，仅凭着互相信任和彼此认同工作在一起，每个人都投入了大量业余时间，字斟句酌，反复推敲。还记得每天下班后虽然已经很累，却仍在夜深人静时翻译几页；还记得每次迭代互相审校之后，即使事先邮件沟通过多次，大家还是会在 Skype 上讨论到深夜；还记得百会工作表中数百条待定意见逐一关掉后获得的满足感。当然，我们应该可以做得更好，比如在每次迭代后都尽快拿到出版社的反馈意见，尽早做出翻译风格的调整。无论如何，这是一段存在我深深的脑海里的难忘经历，因为能与志同道合的朋友共事的机会是奢侈的。我也从两位合作者的精神和专业知识中学到了很多，感谢有你们。另外，也要感谢家人的支持和照顾，使我能够安心地投入工作。

——申健

2012 年对我而言是特殊的一年，在大洋彼岸开始了一段新的生活。因为怀孕，准备生老二，赋闲在家，怕自己跟技术脱节，作为一个 SOA 实践者，我对 REST 结合 SOA 这一论题很感兴

趣，于是加入了这段长达一年的翻译历程。我同另外两位译者并不熟悉，与申健之前不认识，国耀虽然是以前的老同事，但也仅限于邮件打过交道。一开始我对这种远程合作的模式的想象是非常松散的。但是实际上，国耀很快把翻译的工作组织的井井有条，利用百会表格交互分工安排，词汇勘误；相互进行审校；使用 Skype 进行定期交流，讨论所有翻译中有争议的部分；利用 REST 技术 QQ 群与潜在读者进行相关技术讨论。这种工作模式让我在忙碌的家庭琐事之余感觉到自己是个有追求的 IT Professional！终于，书籍翻译完成时，我家的 Max 呱呱坠地，定稿前，我刚好开始了在澳洲的第一份工作，有幸参与一个大型的 SOA 项目，而我也有机会将阅读本书过程中得到的新知识加以运用，尤其是服务的筛选，事务的处理等相关知识。感谢本书，充实了我在澳洲的第一年，让我认识了两位朋友，拓展了 REST 相关的 SOA 知识，这段翻译历程，将会成为我的一段珍贵的回忆。

——刘慈

序

2002 年初我第一次听到 REST 时，我就坚信新兴的 Web 服务规范和标准即将带来的价值。起初，我只是对这一方法充满好奇，尤其“统一接口”的理念，但是很快我便得出结论，虽然 REST 很有吸引力，但是它却几乎无法适用于企业环境。

一两年之后，我便开始欣赏 REST 风格的优雅和简洁性了。而且，我还认为在某些场景下 REST 是更好的选择，虽然一些高级的场景仍然需要使用 SOAP、WSDL 和 WS-*。又过了一年之后，我开始发现自己在大多数情况下都更倾向于 REST 风格的 HTTP，而非 SOAP 风格的 Web Service，并且我已经说服了自己，并决定自称为“REST 盲从者”。我开始坚信，现在仍然坚信，遵守 REST 架构风格的约束不仅能使公共 Web 上的系统变得更好，而且还适用于各种企业环境。

今天，REST 已经成为主流——连同它对别的技术所产生的正面及负面的影响。我们现在向人们介绍 REST 方法时是相当轻松的，即便在大企业里也不例外，人们不再对此不屑一顾了。此外，让我惊讶的是，很多时候人们甚至将其视为默认选择。

所以，REST 社区应该感到高兴和满足，并对我们在公共 Web 上及企业内构建出各种优秀的系统开发及集成充满期待。但是，这里仍然存留着两个问题：一是，并非所有声称 RESTful 的架构都是名符其实的；二是，SOA 常常被看成由 WS-*风格的 Web Service 组成的架构，因此与 REST 不相容。

“REST”名称的误用

遵循 REST 原则，应用 Web 技术（如 HTTP 和 URI），使用超媒体格式（如 HTML），就能使系统变得可演进、动态且稳定，并且可以通过新的未预见的方式连接系统。可是，真正称作“RESTful”的构建系统或集成系统的方式需要一种全新的设计方法，它不同于大多数开发人员过去所使用的方法。若选择 REST 方法而非分布式对象、RPC 或 WS-*时，需要通过资源、媒体类型、超媒体及统一接口来表述整个接口域。你将发现，对这些概念理解得越深入，就越能轻松地坚持 REST 的内在约束，越容易挖掘 REST 的价值。

因为 WSDL/SOAP/WS-*架构的一个根本设计决定是“传输独立性”，所以通过它们构建的系统很难成为 RESTful 的，几乎无人反对这一说法。但是，即便没有出现 XML 或 SOAP，Web 的底层理念也很容易遭到违反。互联网上充斥这样的例子：通过 HTTP GET 或 POST 实现的管道（Tunnel）方法调用，忽视缓存和超媒体，将构成 URI 的字符串当作开发用的 API。使用

JSON、HTTP 和“简洁”的 URI 不一定意味着系统就是 RESTful 的。不要想当然地把任何宣称为“RESTful API”的东西都当作典范。

评判 SOA

“SOA”这个标签已经被完全炒作殆尽了。在我看来，这主要由于很多人将它与 Web Service 技术（最初它是支持 SOA 的流行方法）紧密关联的缘故，而且此现象仍然在继续。网络上许多有关 SOA 与 REST 的比较大多将它们强行地变成类似于苹果与橘子之间的比较。所以，我们有必要回过头来看看 SOA 最初的动机。

在任何公司里，整体 IT 环境都是由许多单独的系统构成的。这些系统运行在不同技术之上，最初通过不同的工具建设而成。有些来自与商业厂商，有些是企业内建的。一段时间后，你会想把它们连接起来，因为这么做的价值是显而易见的。你可以采用点对点的集成方式，通过文件、共享数据库或其他集成解决方案，从某个系统中导出一些数据，定期地导入到另一个系统中。这么做的结果是，你将得到一个脆弱的、不易于管理的 IT 环境。

因为系统集成的工作非常困难，所以你（或为你服务的厂商）会使其变得越来越臃肿，直到你不得不付出巨大的代价来改变。尽管引入集中式的集成中间件起初看起来非常诱人，但它却不是个好办法。你将依赖于一家厂商，它有可能被另一家收购，或者破产，或在你与竞争对手（它采用的是另一家产品，而且更时新）合并之后变成遗留系统。

那么，你该怎么做呢？你应该合理地解决这一问题，通过在接口层限制不同技术的数量来减少集成系统所需的工作。为实现这一目标，你可以选择一种较好的接口抽象方法，将大型应用系统分解成更细粒度的功能模块。这样，你就无需依赖于单个厂商，不论它是中间件厂商还是应用系统厂商，你期望功能便于使用（及重用）。应该不会有人否认这是一个合理的方法。

对我而言，这就是 SOA 的本质：它是一个运用于企业内一组应用而非单个应用的系统软件架构；它重点关注网络和系统间的接口，而非实现；它进行一切必要的标准化，确保流畅的交互和偶尔的重用；别无其他。几年前，我们发表 SOA 宣言时，重点关注的是内在互操作性，我认为，这才是真正象征着 SOA 和面向服务的特性，也正是它促使了 SOA 与 REST 原则的对齐。

在我看来，面向服务是非常值得追求的一个目标，而 RESTful HTTP 是当前看到的实现这一目标的最佳方法。任何其他架构都不具备如此广泛的支持，便利地支持重用，以及相同的对持续演变的支持。然而，在 SOA 环境中使用 REST 仍然处于初级阶段，到目前为止，尚很难找到成熟的最佳实践与模式。

此为试读，需要完整PDF请访问：www.ertongbook.com