小 型 微 型 计 算 机 系 统
Journal of Chinese Computer Systems

2010 年 7 月 第 7 期
Vol. 31 No. 7 2010

# REST 和 RPC:两种 Web 服务架构风格比较分析

冯新扬,沈建京

(信息工程大学 理学院, 河南 郑州 450001)
E-mail:hermit2005@sina.com

**摘 要**: 目前包括 SOA 在内的大量 Web 服务架构均采用 RPC 风格构建,在 Web 级的大规模应用中 RPC 风格的架构在扩展性、性能等方面存在着瓶颈. 由于 REST 的架构属性更加符合 Web 的设计理念,REST 成为除 RPC 之外 Web 服务架构风格的另一种选择. 可扩展性、耦合性、安全性等多个方面对 RPC 和 REST 进行了比较分析,并探讨了 Web 服务架构技术的发展方向.

**关 键 词**: RPC;REST;Web 服务;架构风格
**中图分类号**: TP393 **文献标识码**:A **文 章 编 号**:1000-1220(2010)07-1393-03

# REST and RPC: Comparative Analysis of Two Web Services Architectural Styles

FENG Xin-yang, SHEN Jian-jing
(*Science Institute of Information Engineering University, Zhengzhou* 450001, *China*)

**Abstract**: At present, a large number of Web services architectures are RPC-styled, SOA, for example, which have scalability and performance bottlenecks in large-scale Web applications. Due to the adaptability to Web, REST becomes an alternative to RPC for Web services architecture. A comparative analysis between RPC and REST is made in many ways, such as scalability, coupling, and security. In the end, the direction of Web services architectural development is discussed.

**Key words**: RPC; REST; web services; architectural styles

## 1 引 言

作为新一代的分布式技术,Web 服务越来越多的被用于解决异构平台互操作和企业应用集成问题. 目前,多数 Web 服务基于 WS-* 协议栈(SOAP、WSDL、WS-Addressing、WS-ReliableMessaging、WS-Security 等)构建,这种类型的 Web 服务采用 RPC(Remote Procedure Call)风格,这些服务能够满足局部范围内的互操作需求. 随着应用规模的扩大,RPC 风格的 Web 服务逐渐显露出可扩展性不足、复杂性大、性能不高等问题,而且受其架构风格的制约这些问题难以克服. 为了能够解决这些问题,REST(REpresentational State Transfer)架构风格的 Web 服务逐渐进入人们的视野.

## 2 RPC 概述

RPC 最早起源于 10 年前的分布式对象运动,该运动旨在用跨网络的对象消息来替代内存中的对象消息[1]. RPC 架构风格将服务器看作是由一些过程组成,客户端调用这些过程来执行特定的任务. 过程是动词性的,因此 RPC 建模是以动词为中心的,这种方式实际上是一种事务脚本[2].

SOA 就是一种典型的 RPC 风格的架构. SOA 的目标是将业务规则和策略抽象成可共享的分布式服务来避免出现孤立的应用. 蕴含在 SOA 背后的概念非常有价值:从应用中抽象出业务服务,定义服务的契约描述(WSDL),并将其注册在服务容器中以便客户端查询和共享(UDDI),客户端依据服务契约利用 SOAP 消息远程调用服务中的过程(操作),这样可以产生一个易于创建、维护和扩展的整体系统. 这看上去似乎很完美,但这种完美只是存在于理想中. 由于采用 RPC 风格的 SOA 抽象出的服务是由若干个离散的过程(操作)组成,它无形中就已经限定了服务使用者对业务数据的使用方式,而在实际的应用系统中有着无数的业务规则,要想使服务中的过程覆盖应用中所有的业务规则,所花费的成本是巨大的,而且随着系统规模的增大会带来一系列扩展性、性能方面的问题,使系统难以保持良好的设计. 由于 RPC 风格服务的高复杂性和对大型平台的依赖性,一些学者将该风格的 Web 服务戏称为"大 Web 服务"[3].

## 3 REST 概述

REST 是一种架构风格,它是由 Roy T. Fielding 在自己的博士论文中首先提出来的. Fielding 曾从事 HTTP1.0 的开发,并担任 HTTP1.1 的首席架构师,他还参与了创建统一资源描述符(URI)的工作. Fielding 将 REST 视为一种有助于传达蕴含于 Web 中的基本概念的方式. 要理解 REST,就必须澄清资源(Resource)、表示(Representation)和状态(State)三个概念. 资源可以是任何事物,它可以是一个实物,也可以是一个抽象的概念. 只要有被引用的必要,就可以将其抽象为一个资源. 通常一个资源是某个可以存放在计算机上并体现为比特流的事物. 表示是一个资源当前状态的有用信息,对于给定的资源,可以有多个不同的表示. REST中的状态分为资源状

收稿日期:2009-03-24 作者简介:冯新扬 男,1980 年生, 博士研究生,研究方向为网络系统软件; 沈建京,男,1961 年生, 教授,博士生导师,研究方向为分布式智能系统.

态和应用状态:资源状态是关于资源的信息,保存在服务端;应用状态是客户端在应用中所处状态的信息,由各个客户端自己维护. REST 提供了一组架构约束,当作为一个整体来应用时,强调组件交互的可伸缩性、接口的通用性、组件的独立部署、以及用来减少交互延迟、增强安全性、封装遗留系统的中间组件[4]. REST 的一些设计约束:

1. 将所有事物都抽象为资源
2. 为每个资源定义唯一的资源标识符 URI;
3. 使用统一的接口对资源进行操作;
4. 通过资源的表示来处理资源;
5. 消息具有自描述性;
6. 所有的交互都是无状态的;
7. 将超媒体作为应用状态的引擎.

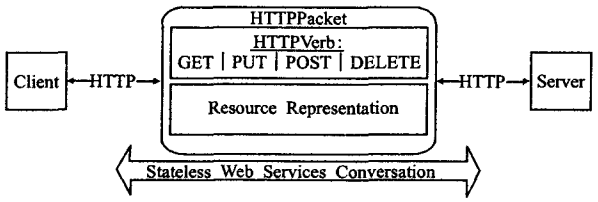


图 1 REST 风格的 Web 服务架构

Fig. 1 REST of the web services architecture

REST 与 RPC 在抽象原则方面存在着巨大不同,采用 REST 的架构将服务器抽象为一组离散资源的集合,资源是一个名词性的概念,因此 REST 的抽象建模是以名词为中心的,是一种领域模型[2]. 图 1 是 REST 风格的 Web 服务架构示意图. 客户端通过统一的接口与服务器进行无状态的交互,在交互过程中服务器与客户端交换的是资源表示.

## 4 二者的比较分析

作为架构风格 REST 提出了一组相互协作的架构约束,这些约束限制了架构元素的角色和功能,以及在任何一个遵循该风格的架构中允许存在的元素之间的关系. 上文中每一种约束都为 REST 架构带来了某些架构属性,而这些架构属性恰恰是 Web 赖以成功的关键,如易扩展、低耦合、可寻址等. 在下面的章节里,我们将结合这些属性对 REST 和 RPC 进行比较分析.

### 4.1 可扩展性

在 RPC 风格的架构中,如 SOA,服务由细粒度的自定义操作组成,不同的服务具有不同的专有接口. 每个接口具有自己的语义和操作参数,服务的接口契约对服务的定义非常关键. 客户端要想与 SOA 服务正确的互操作,它必须理解每个服务接口契约的语义. 这种方式在相对封闭的应用环境中不会出现问题. 然而由于操作的数量是没有限制的,在 Web 这个开放的分布式环境中会产生紧密耦合和接口复杂等问题,难以达到 Web 级规模的可扩展性要求. 试想一下,以目前网络中 Web 页面的数量,如果每个页面都定义了自己的专有接口,要求浏览器必须下载或编写插件来适应这些接口,否则就无法了解接口语义进行交互,那么客户端的浏览器将不得不安装数百万的不同插件,这种情况显然是不能接受的.

在接口问题上 REST 采取了与 RPC 不同的方式,统一接口约束(约束 3)要求 REST 架构要为所有资源提供统一的操作接口. 最著名的 REST 实现就是 HTTP 协议,REST 架构理念正是基于真正理解 HTTP 协议而形成的. 对于 REST 而言,HTTP 不仅是简单的数据传输协议,还是状态转移协议. 它不但能够对于互联网资源进行唯一定位,而且也作为直接数据处理的工具. REST 主张使用 HTTP 标准的 GET、POST、PUT 以及 DELETE 来进行请求和响应. GET 用于获取资源的表示,PUT 创建一个资源,POST 为已存在的资源创建从属资源,DELETE 删除一个资源. 每个操作都具有明晰的语义,且 GET 操作不会导致资源状态的改变,具备安全性. GET、PUT、DELETE 三种操作对资源使用多次和使用一次的效果相同,具备幂等性[3]. 统一接口成为了接口定义的巴别塔,它使得客户端和服务器端的实现解耦,可以独立进化. 只要接口不变,就不会影响对方的使用. REST 的统一接口约束,将接口的变化转移到了资源表示上,这种变化性的转移,使我们的关注点更贴近最终的目标 - 数据.

REST 的无状态交互约束(约束 6),也使得 REST 架构的扩展性提高. 无状态交互要求客户端的每个请求都应包含该请求所需的全部信息,这些状态信息不应保存在服务器上,也不应蕴含于之前的请求之中. 无状态交互约束的引入,降低了提升系统规模所需的代价. 由于会话都是无状态的,当系统需要扩展时,只需增加负载均衡服务器即可,不必在各台服务器之间做大量的协调工作. 而 RPC 中的有状态会话在面临相同问题时,不得不采用数据复制、共享内存等辅助技术手段才能达到同一效果.

### 4.2 耦合性

RPC 风格的架构中,在服务契约中定义的不仅仅是接口,还包括交互所涉及的数据格式. CORBA 中使用 IDL 来定义数据,SOA 架构在 WSDL 中普遍使用 XML Schema 来定义数据. 这种将接口与数据契约绑定在一起的方式在设计之初是为了便于代码生成. 常用的面向对象的编程语言一般采取这种方式,如 Java、C + + 都是将方法与数据类型定义在一起. 在这些语言的使用中,如果数据格式要发生变化,所有该数据格式的使用者都要重新编译来使新的定义生效. 但是这种方式违背了接口与实现分离的原则,使客户端与服务器间的耦合度大大增加. 也许对于本地系统这种高耦合的负面效果并不明显,但是在 Web 级的分布式系统中这种高耦合必须避免.

在 REST 中,鉴于统一接口的约束,数据类型与接口是正交的. 资源的状态都是通过表示来处理(约束 4),且 REST 要求消息是自描述的(约束 5),表示中的数据格式是可以变化的,它取决于客户端与服务器内容协商的结果. 不同的消息可以通过 HTTP 的 content-type 和 accept 头中指定不同的格式 - 前者指明消息的数据负载格式,后者在请求中指明调用者期望在响应中接受的数据格式[5]. REST 对数据格式的处理方式减轻了客户端与服务器间的数据耦合,允许服务处理多种数据格式意味着客户端和服务可以为不同类型的数据选取

合适的数据格式,如图片、文本或电子表格. 这些媒体类型是 REST 表示元数据的具体形式. 在消息使用这些元数据也使得客户端可以请求它们喜欢的数据格式. 在 HTTP 消息中,可以使用 IANA 的 MIME 来标识数据格式,MIME 是一个通行的国际标准可以减少数据格式中存在的歧义[6].

### 4.3 安全性

SOAP 是目前 RPC 风格的 Web 服务中使用最广泛的协议之一. 客户端与服务器通过交换 SOAP 数据包来实现交互. SOAP 数据包通常被放入 HTTP 文档中,利用 HTTP 的 POST 方法来传递,借助 HTTP 协议的 80 端口可以很方便的穿越防火墙. 但这种将 HTTP 当作管道的方式存在安全隐患,因为每个请求的真实意图包装在 SOAP 文档中,当 SOAP 文档解析时才被服务器了解,如果 SOAP 携带的是一些危险的请求(如非法删除、恶意修改),那么这些请求将不能被防火墙所阻挡. 为了消除这些潜在的危险,防火墙不得不执行额外的协议过滤.

相对于 RPC,REST 的安全模型更加简单有效. REST 中所有事物都被抽象成了资源(约束 1),每个资源都有唯一的 URI(约束 2),有了这两个约束,如果要隐藏某个资源,不发布它的 URI 即可. REST 使用 HTTP 的统一接口,每个接口的语义清晰,且不含有 SOAP 式的嵌套. 对资源的 4 种操作分别设置权限,就可以形成不同的安全策略,如果某个资源是只读的,那么只开放它的 GET 操作,如果资源允许删除,就开放它 DELETE 操作,且这些设定全部可以由 HTTP 防火墙来完成,大大降低了实现安全策略的难度.

### 4.4 可寻址性

可寻址性是 Web 应用的重要特点,有了可寻址性用户就能够在 Web 中简明准确的描述自己要访问的信息. RPC 风格不排斥可寻址性,SOA 支持命名资源,但是它的命名方式缺乏固定的标准不具备一致性,服务的命名与标识完全取决于该系统的设计者与实现者. 而且它的标识符也并非指向数据资源而是指向包含一组操作的服务契约,这更像是分布式的网络句柄,往往数量很少甚至只有一个. REST 要求为每个资源定义基于 URI 标准的资源标识符(约束 2),也可以为资源的每个表示设计标识符. URI 标识符能够封装寻址所需的信息且可读性好,允许用超级链接的方式传播. 在 REST 中的 URI 的数量是无限多的,这与传统的 Web 页面的命名机制是一致的.

### 4.5 连通性

连通性以可寻址为基础,它的存在使 Web 可以形成一个相互关联的整体,用户可以在自己感兴趣的页面间导航. REST 要求通过资源的表示来处理资源(约束 4),并将超媒体作为应用状态的引擎(约束 7),服务器可以通过在表示里给出表单和链接来引导应用状态的迁移,链接起到了连接资源的作用. 设计良好的 REST 服务将具备出色的连通性,服务调用者只要跟随链接并提交表单就能推进应用状态了. 相对而言,RPC 架构在连通性方面就不那么出色. Web 中的连通性指的是相关数据信息间的连接,而 RPC 架构中充满的都是一个个操作,即使将服务契约的标识符通过某种技术手段连接在一起,也只是组成了一个动作序列,并不能达到信息导航的目的.

### 4.6 交互性能

交互性能是 REST 的优势之一. 首先,REST 性能优势来自于它与生俱来的简单性. REST 建立在已经广泛使用的 Web 标准之上,不需要额外的附加标准,从而避免了对大型专用平台的依赖,减少对系统资源的占用. 比如在数据传输方面,REST 的交互直接使用 HTTP 协议,客户端和服务器都免除了解析和封装 SOAP 数据包的性能消耗,也降低了传输的负载. 其次,REST 主张客户端或媒介缓存那些服务器标识为允许缓存的应答来消除一些不必要的交互,以便提高性能. 在实际应用中,Amazon 同时提供了基于 REST 的和基于 SOAP 的 Web 服务,据 Amazon 宣称 REST 风格的 Web 服务比基于 SOAP 的快 6 倍[7].

## 5 结束语

REST 采用了一种新的思维方式来抽象服务,让人们真正理解网络协议 HTTP 的本来面貌,更充分利用现行 Web 的特性,使服务更贴近 Web 而不是背离它. 与 RPC 风格的 Web 服务架构相比,采用 REST 的 Web 服务架构在扩展性、安全性、数据耦合性等方面存在着优势. 随着支持 REST 开发工具的涌现,REST 会被越来越多的采用,并逐渐成为 Web 服务的主流技术.

**References:**


[1] Steve Vinoski. REST eye for the SOA guy[EB/OL]. http://www2. computer. org/portal/web/csdl/abs/html/mags/ic/2007/01/w1082. htm,2008.

[2] Martin Fowler. Patterns of enterprise application architecture[M]. Addison-Wesley. 2002.

[3] Leonard Richardson, Sam Ruby. RESTful web services[M]. O'Reilly Media. 2007.

[4] Roy Thomas Fielding. Architectural styles and the design of network-based software architectures[D]. Doctorial Dissertation, Dept. of Computer Science, Univ. of California, Irvine. 2000.

[5] Steve Vinoski. Demystifying RESTful data coupling[EB/OL]. http://ieeexplore. ieee. org/stamp/stamp. jsp? arnumber = 04463391. 2008.

[6] IANA. MIME media types[EB/OL]. http://www. iana. org/assignments/media-types/. 2007.

[7] Adam Trachtenberg. PHP web services without SOAP[EB/OL]. http://www. onlamp. com/pub/a/php/2003/10/30/amazon_rest. html. 2003.