

# 基于 REST 架构风格的 Web2.0 实现<sup>①</sup>

## Implementation of Web2.0 Based on REST-Style Architecture

戴亚娥<sup>1</sup> 俞成海<sup>2</sup> 尧飘海<sup>2</sup> 李艳芳<sup>2</sup> (1.浙江工商职业技术学院 信息工程学院

浙江 宁波 315012; 2.浙江理工大学 信电学院 浙江 杭州 310018)

**摘要:** 分析了现有 Web 系统框架的不足, 缺乏统一的软件体系结构的概念, 各系统是紧耦合的, 系统的小部分功能需求改变有可能牵动整个技术架构的调整, 提出了基于用户状态转移的原则的轻量级模型, 通过分析此模型的特点, 然后以图书服务的例子实现了其架构风格, REST 对于资源型服务接口来说很合适, 将服务器上共享的任何信息看作是资源, 客户端通过资源标识符去操作资源, 获得资源的表示, 其风格的开发方式会使系统结构更加清晰。

**关键词:** Web Web 模型 REST HTTP

### 1 引言

随着企业级应用的发展, 存在各种不同类型的系统框架。特别是在 Web 应用中, 在系统架构分析时, 如何应用这些已有框架为系统量身定做一个合适的架构, 对 Web 设计开发者提出了挑战。传统的 Web 架构为了简化应用程序开发、部署, 降低开发的风险, 提高程序的可维护性, 引入轻量级 Web 架构模型。所谓轻量级是相对“重”而言的, 众多的中小型系统中需要的是一种“可选择”的服务容器, 轻量级容器正是在这种需求驱动下提出的<sup>[1]</sup>。

Web 服务作为 Web 应用程序的分支, 是目前比较热门的企业级技术。它借助 XML 元标记语言, 发现和定义服务等方面都采用了标准的规范, 如 SOAP<sup>[1]</sup> 协议、WSDL<sup>[1]</sup> 规范等; 它采用 HTTP 协议进行传输, 因此可以跨越防火墙的限制; 是自包含、自描述、模块化的应用; 由于它是基于标准互联网的协议, 只要将服务功能发布和部署在 Internet 和 Intranet, 其他任何平台和应用程序可以通过协议查找并调用部署的服务, 处理各种操作和请求。因此 Web 服务具有异构性、跨平台性及松散耦合性, 同时也支持分布式系统的集成应用, 被认为是当前基于 Internet 环境下的构件编程, 具有组件的集成和重用等特性。目前在 Web 服务中, 常用的标准主要包括 WSDL、SOAP 和 UDDI。

SOAP 是 Web 服务中的基础协议, 它采用的交互模型是基于 RPC 机制的。SOAP 设计的初衷之一就是使在 Internet 上的 DCOM、CORBA 和 EJB 等中间件之间相互沟通, 而这些中间件在实现企业逻辑时基本都是通过基于 RPC 方法的, 所以采用 RPC 作为 Web 服务的交互模型是理所当然的。在一个相对封闭的环境中, 即 RPC 模型取得了较大成功。因为在封闭环境中, 所有用户都是已知的, 可以共享一个数据模型, 用户直接调用发布接口的 API 来获得服务, 服务器端的开发者也通过接口获得了很大的灵活性。然而, 在 Web 环境, 尤其是应用程序达到了 Web 级的规模可伸缩性, 就会出现一些问题。首先, 使用即 RPC 的方法会导致调用者和服务之间的紧密耦合。举例来说, 在一个订单处理系统中, 可能会使用信用卡授权服务, 在客户端使用即 RPC 方法调用了服务后, 它必须等待, 直到服务器端处理完成后, 才能进行订单处理的动作。整个订单服务将会持续很长时间, 而 RPC 不支持长时间的连接活动。其次, RPC 系统在 Internet 的规模下产生接口复杂性。接口与实现它的对象或服务之间都有一定的约定, 如方法调用的顺序、参数类型等, 这样, 每个接口都具有自己的语义。随着接口个数的增加, 接口的语义有可能以接口个数平方增长, 这在 Web 级的规模上会产生接口复杂。

<sup>①</sup> 收稿时间:2008-12-16

在 SOA<sup>[2]</sup>的基础技术实现方式中 Web Service 占据了很重要的地位, 通常我们提到 Web Service 第一想法就是 SOAP 消息在各种传输协议上交互。当前基于 RPC 机制的 SOAP 协议存在一些不足, 主要表现在:

①安全性。RPC 交互模型下是通过调用方法来访问服务的, 所以要访问的服务对象名称藏在方法的参数中, 无法在代理服务器一级进行有效的控制。而安全问题是企业计算环境中需要面对的关键问题, 也是 Web 服务达到大规模商业应用的一个障碍。②代理和缓存。由于采用即 RPC 模型, 服务调用者要访问的资源和方法被封装在 SOAP 消息中, 单从 URI 和 HTTP 上无法得到有用的信息, 因此在 Web 环境下, SOAP 在支持代理和缓存服务器方面有一定的困难。③使用复杂性。由于在 SOAP 中可以任意定义自己的方法集合, 因此要有一个描述和发现机制, 使服务调用者获得服务及它提供的方法的语义后, 才能根据这些信息进行调用。当前, 基于远程过程调用(RPC)的交互模型, 在相对封闭的、小的应用环境中取得了较大成功。然而, 在 Web 这个开放、分布的环境中会产生紧密耦合和接口复杂等问题, 难以达到 Web 级的规模可伸缩性。

REST<sup>[3]</sup>(Representational State Transfer)是对当前 Web 体系结构潜在设计原则的一种描述, 也是对 Web 最成功要素的总结。采用基于 REST 的交互模型的 Web 服务将克服基于 RPC 的交互模型的诸多不足, 适应 Web 级的规模可伸缩性, 促使 Web 服务得到普遍应用和进一步发展。

## 2 实现原理

REST 不是一种协议, 而是一种体系结构风格, 是对当前 Web 体系结构设计原则的一种抽象和描述。它是一组协作的架构约束, 它试图使数据延迟和网络通信最小化, 同时使组件实现的独立性和伸缩性最大化。REST 通过将约束放置在连接器的语义上来实现, 而其他的架构风格则聚集于组件的语义。REST 提供缓存的交互和重用、动态替换组件以及中间组件对于动作的处理, 因此满足了 Internet 规模的分布式超媒体系统的需求。

在分布式系统中, REST 是基础架构实体的一种抽象, 它由数据元素、组件和连接器组成。其中数据元素指资源、资源标识及资源表示, 例如: HTML 文档、图像及 XML 文档等; 组件指服务、网关、代理及用户

代理等; 连接器指客户端、服务端和缓存等。在一个简单的企业系统中, REST 风格如下图 1 所示:

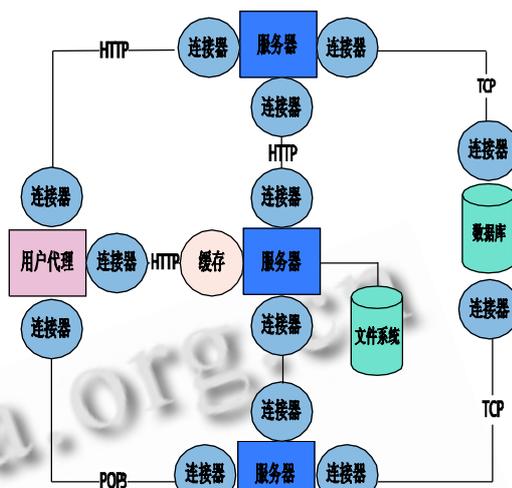


图 1 REST 架构风格

REST 的目标是:

- (1) 组件相互作用的规模性;
- (2) 接口的一般性;
- (3) 组件的独立分发;
- (4) 支持中间媒介。

REST 系统中的组件必须遵守下列约束:

- (1) 网络上的所有事物都被抽象为资源(Resource);
- (2) 每个资源对应一个唯一的资源标识(Resource identifier);
- (3) 通过通用的连接器接口(Generic connector interface)对资源进行操作;
- (4) 对资源的各种操作不会改变资源标识;
- (5) 所有的操作都是无状态的(Stateless)。

REST 是对当今 Web 体系结构设计原则<sup>[4]</sup>的一种描述, 所以 REST 的目标和原则是对当今 Web 中已成功应用的要素的总结。从我们正在应用的 Web 上讲, Web 上的每个资源通过一个 URI 来标识, 可以通过简洁通用的接口(如 HTTP 的 GET 和 POST)来操作 Web 上的资源。资源使用者与资源之间有代理服务器、缓存服务器来解决安全及性能等问题。REST 系统中的组件必须是自描述的, 这样, 客户可根据这些自描述信息来维护自己的程序状态。

在 REST 系统中, 所有资源都有一个 URI, 包括 Web 服务也可以用 URI 来标识。用以资源标识的 URI

最好是逻辑 URI，而不是物理 URI，和文件系统对应的相对路径和绝对路径相似。使用逻辑 URI 的好处是对服务器端的资源修改不影响客户的使用。目前，在标准的 HTTP 协议中，一般支持 GET、POST、HEAD 等几个常用的动作，而 REST 使用 HTTP 的 GET、POST、PUT 和 DELETE 四个动作作为资源的通用接口，用户通过它们访问资源，类似于数据库的查询、新增、修改、删除等操作功能。在此，HTTP 作为一个程序协议来使用，而不是只作为传送一个 SOAP 消息的传输协议来使用。

REST 在对请求的响应数据中包含链接<sup>[5]</sup>，这一点是非常重要的。在响应数据中保持对其他资源的链接可使客户程序实现自推进，因为响应信息中就包含了程序的下一步动作，这样就可以使客户程序维护自己的状态，而且以资源为中心的 Web 服务在本质上是易于集成的，其在应用程序的架构如下图 2 所示：

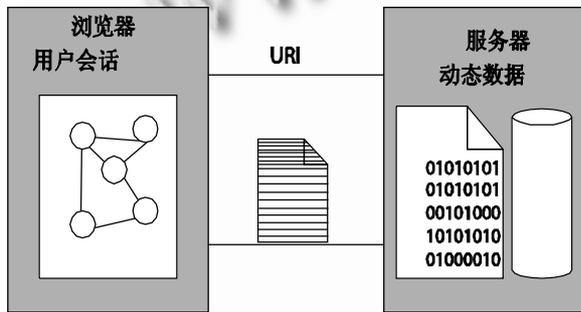


图 2 遵守 REST 准则的应用程序可伸缩架构

### 3 实现举例

由于 Web 服务可以通过一个 URL 来访问，因此调用服务十分简单。以图书列表服务为例来说明。例如：需要得到服务的图书列表，假如有一个 Web 服务 books 实现此功能，那么使用 http://localhost:3001/books/ 即可得到图书列表的 HTML 表示形式，如图 3 所示。客户只需这样调用，而至于服务器端的实现，对客户来说是透明的。返回响应数据中包含对各个图书的链接，客户通过选择合适的链接，对该链接指向的资源的描述迁移到客户，实现客户状态的自维护，这也是 REST 的关键特征。如果想以 XML 的形式返回，只需在 URL 后面加上参数，如图 4 所示，另外还可以采用 RSS, JS 等表现形式。如果要新增或删除图书，只需在客户端创建一个符合图书格式的实例，通过 HTTP 方法提交，而返回的响应数据中包含对此

图书的 URL。其服务列表的 Ruby 程序代码段分别为以下的 @index 和 @create。



图 3 EST 图书列表



图 4 EST 图书列表的 XML 表示

```
def index
  @books = Book.search(params[:query],
                        params[:page])
  respond_to do |format|
    format.html # index.html.erb
    format.xml { render :xml => @books }
    format.js
    format.rss
  end
end

def create
```

```
@book = Book.new(params[:book])
respond_to do |format|
  if @book.save
    flash[:notice] = '新增成功'
    format.html { redirect_to(@book) }
    format.xml { render :xml =>
      @book, :location => @book }
  else
    format.html { render :action => "new" }
    format.xml { render :xml =>
      @book, :location => @book }
  end
end
end
end
```

下列表 1 是图书服务所有对应的请求资源映射，客户只要按图 4 的 XML 的格式提交对应的资源到对应的地址上，即能实现对图书资源的新增、修改、删除和查询等操作。

表 1 EST 资源映射

| http 动作 | Rest URL | 操作 | 映射地址                     |
|---------|----------|----|--------------------------|
| GET     | /books/1 | 详细 | GET<br>/books/show/18    |
| DELETE  | /books/1 | 删除 | GET<br>/books/destroy/18 |
| PUT     | /books/1 | 修改 | POST<br>/books/update/18 |
| POST    | /books   | 新增 | POST<br>/books/create    |

REST 是一种针对网络应用的设计和开发方式<sup>[6]</sup>，可以降低开发的复杂性，提高系统的可伸缩性。它通过引入架构约束，把 HTTP 操作限定在 GET、POST、PUT 和 DELETE 四个操作，并把它们作为对数据库的 CRUD 的实现，使得开发人员把系统的实现采用数据库的操作模式进行。另外，它强制所有的操作都是无状态的，这样所有的操作都没有上下文约束，对于分布式运算、集群方便。同时它可以使使用连接池来缓存

部分系统，分配服务端和客户端的任务，Server 端只需要提供资源以及操作资源的服务，而 Client 要根据资源中的数据和表示进行相关的渲染，减少了服务器的开销。

#### 4 结论

现在软件过程中常常缺乏统一的软件体系结构<sup>[4]</sup>的概念；各系统是紧耦合的，系统的小部分功能需求改变有可能牵动整个技术架构的调整；单一的系统结构、单一的语言、单一的平台等。一方面造成传统的遗留系统无法集成到新系统中，另一方面对以后软件的升级、维护也将带来极大的不便。REST 对于资源型服务接口来说很合适，特别适合对于效率要求很高。它提供了简明的 Url 定位一个资源，传回给客户端不同格式的内容，同时其代码编写更少，直接在 HTTP 协议上支持 Create,Retrieve,Update,Delete 等操作。总之，REST 风格的开发方式会使系统结构更加清晰，开发人员可直接在该框架基础上进行应用开发，集中精力实现应用的业务逻辑，而不必在技巧性要求较高的、复杂的基础框架上浪费时间和精力。

#### 参考文献

- 1 姜芳苑.Web2.0 的探讨.计算机工程与设计, 2007, 28(8):1818-1819,1823.
- 2 REST 与 SOAP 之比较.http://www.duduwolf.com/wiki/2007/261.htm.
- 3 许卓明,栗明,董逸生.基于 RPC 和基于 REST 的 web 服务交互模型比较分析.计算机工程, 2003,29(20):6-8.
- 4 Fielding RT. Architectural Styles and the Design of Network-based Software Architectures. UNIVERSITY OF CALIFORNIA, IRVINE, 2000, 5:41-84.
- 5 Richardson L, Ruby S, Hansson DH. RESTful Web Services. O'Reilly Media, Inc, 2007,3:96-102.
- 6 黄宁海.基于 REST 的轻量级 J2EE 架构实现[硕士学位论文].杭州:浙江大学,2008:34-36.