

REST 风格和基于 SOAP 的 Web Services 的比较与结合

王建斌¹ 胡小生² 李康君³ 赵 靓⁴
¹(肇庆学院教育技术与计算机中心 广东 肇庆 526061)
²(佛山科学技术学院信息与教育技术中心 广东 佛山 528000)
³(广东质量技术监督局 广东 广州 520000)
⁴(肇庆医学高等专科学校 广东 肇庆 526061)

摘 要 对 REST 风格的 Web Services 进行介绍,接着对 REST (Representational State Transfer) 风格和基于 SOAP 的两种 Web Services 的耦合度进行比较,进而提出两者是可结合的。这样在构建 Web Services 的时候能做出合理的选择。

关键词 REST SOAP Web Services

COMPARISON AND COMBINATION OF WEB SERVICES IN REST STYLE
AND Web Services BASED ON SOAP

Wang Jianbin¹ Hu Xiaosheng² Li Kangjun³ Zhao Jing⁴
¹ (Education Technology and Computer Center, Zhaoqing University, Zhaoqing 526061, Guangdong, China)
² (Information and Education Technology Center, Foshan University, Foshan 528000, Guangdong, China)
³ (Administration of Quality and Technology Supervision of Guangdong Province, Guangzhou 520000, Guangdong, China)
⁴ (Zhaoqing Medical College, Zhaoqing 526061, Guangdong, China)

Abstract First, REST style Web Services was introduced, then the coupling degree of REST style Web Services and SOAP based Web Services was compared, and then we point out that they can be combined together. In this way, reasonable decision could be made in the process of building Web Services.

Keywords REST SOAP Web Services

0 引 言

网络已经遍及整个世界, Web 也成了全人类使用最多的网络应用。Web 上的资源可谓应有尽有, 各行各业都可以建立自己的 Web 站点, 发布他们的信息和资源。第一代 Web 对于所有人来说, 已几乎没有了技术门槛, 所以 Web 站点遍地开花。

“Web 2.0”概念^[1]的提出虽然引发了不少争议, 但人们已开始期待下一代 Web 的出现了, 然而要迎来 Web 2.0 的鼎盛时代, 可能需要 5 年、10 年甚至更长。在 Web 2.0 提出之前, Web Services 技术雏形在 1999 年已出现^[2]。Web Services 是一项被寄予厚望的技术, Web 2.0 不能少了它。目前提供 Web 资源的站点数不胜数, 但是同时提供 Web Services 的却很少。由于基于 SOAP 的 Web Services (SOAP Web Services) 技术上的复杂性和不成熟等原因, 降低了开发人员的热情, 阻碍了 Web Services 的推广和应用。而 REST 风格的 Web Services (RESTful Web Services) 技术的出现, 给推广和应用 Web Services 带来了新的活力。

1 RESTful Web Services 的发展

REST 是 Roy Thomas Fielding 在他的博士论文中的第五章提出的一种架构风格 (主要用于 Web)。在 REST 中, 很重要的一个概念是 Resource, Resource 可以是: 文档、图片、一个短暂的服务, 甚至可以是真实的人等^[3]。

2005 年 11 月 Restlet API 的首发, 引起开发人员对 REST 架构的关注。面对逐渐增多的 REST API, Roy Thomas Fielding 于 2008-10-20 在博客中撰文《REST APIs must be hypertext driven》, 表示他的看法。他认为那些号称 REST 的 API 只是基于 HTTP 的 API 并非真正的 REST API, 并对构建真正符合 REST 风格的网站 (及 API) 定义了六条规则:

- (1) REST API 不应依赖于某个特定的通信协议;
- (2) REST API 不应该去修改通信协议 (除了填充或修正标准协议中被指明的位);
- (3) REST API 应将几乎所有的精力用于定义媒体类型, 媒体类型是用于表示资源和驱动应用状态的;
- (4) REST API 一定不能定义固定的资源名称或层次;
- (5) REST API 定义资源的类型不应对客户终端有特别含义 (媒体类型除外);
- (6) REST API 应该只需知道初始 URI (书签) 和一套适合

于预期用户 (即可被任何使用该 API 的客户端所理解) 的标准媒体类型^[4]。


目前所有的 API 都不符合上述定义, Roy Thomas Fielding 本人也没有指出或设计出符合上述规则的 REST API。Roy 只给出了抽象规则, 所以我们很难知道如何去实现。并且有些规则是不太现实的, 如定义标准媒体类型, 这个超出 REST API 应考虑的范围。HTML 作为 Web 页面的格式, 最终成为让所有人接受的标准经历了艰难的过程, 如果让媒体类型都要变成标准的媒体类型, 那要付出的努力不可想象, 而且也不是那么迫切。

严格的讲, RESTful Web Services 不能称为 RESTful 但是如果大家都接受这个很酷的名称, 要改变不是件容易的事。RESTful Web Services 把 Web Services 看成 Resource 每个 Resource 都由一个 URL 来标识。我们知道, 普通的资源是通过 HTTP 协议中的 GET 和 POST 两个方法来访问和操作的, RESTful Web Services 还使用了 PUT 和 DELETE 两个方法。通过这四个方法, RESTful Web Services 实现类似数据库的 CRUD (Create, Read, Update, Delete) 操作^[5]。

目前有三种 URL 格式用来标识 Resource, 用得较多的一种是以“/”分割参数的方式^[6], 假如中国期刊网实现了 RESTful Web Services 则可以这样去访问一个资源 (一个 Web Service):

访问某个编号 (可变) 的文章: `http://dlib.cnki.net/kns50/papers/{paper id}`;

访问《计算机科学》2008 年 6 月的文章: `http://dlib.cnki.net/kns50/papers/jjxk/2008/06`

最好有一些通用性的图标, 告知用户可以进行何种操作。RSS/Atom 用  图标来表示 feed, 方便我们的收藏。RESTful Web Services 也可以借鉴该思路, 在页面中添加如下的图标, 来区分 Services 资源和一般性资源:

- R 图标 RESTful Web Services 标志, 默认 GET 操作;
- 绿色的 R 图标 对资源的 Post 操作 (创建);
- 红色的 R 图标 对资源的 Update 操作;
- R 图标 + 垃圾桶图标 对资源的 Delete 操作。

尽管 RESTful Web Services 的 API 不是 Roy Thomas Fielding 所说的 REST API, 但有些目标是一致的。Roy 谈到的 Resource 可以是文档、图片以及服务, 其中的隐含意思就包括服务和其它资源可以很好地混合在一起, 而 RESTful Web Services 也正有这样的目的, 它可以简化 Web 服务的开发和部署 (甚至不需要配置)。这样 Web Services 看上去和网页、图片、多媒体等资源是类似的, 用文件系统和超链接的组织方式就可以了。

2 REST 和 SOAP 两种 Web Services 的比较

SOAP 协议从 1998 年提出到现在的 SOAP 1.2 已经走过十年的时间, 相比其他的技术, SOAP Web Services 的发展并不算快, 当然和 SOAP Web Services 相关发布了很多的协议, 俗称 WS* 协议, 但是应用的情况不令人乐观。REST 的出现, 确实可以帮助快速开发 Web Services 但并不会取代 SOAP。下面主要从耦合度去比较 REST 和 SOAP 系统。

- 打个比喻, 分别用 REST 和 SOAP 去实现 119 火警系统。
- REST 路人甲救火只需要打 119, 消防队就会派人派车去现场处理, 下一次再想救火, 也只需要拨打 119。
 - SOAP 路人乙救火, 不仅需要打 119 还要具体找到某个负责人, 负责人再派人派车去处理, 而如果负责人发生了变动,

而路人乙不知, 则无法报告火情。

通过这个比喻, 我们能更好地对系统进行耦合度的分析。为了分析全面一些, 把 119 火警、REST、SOAP、RMI 四种风格系统的耦合度进行了比较, 如表 1 所示。

表 1 四种风格系统的耦合度分析

系统的构成	119火警	REST	SOAP	RMI
协议	电信 (程控交换)	主要是 HTTP	HTTP / SMTP	TCP / UDP
服务定位	119号码	URL	URI	Server IP
数据 (消息)	语音 (报告起火地理位置)	文本	XML (SOAP 信封)	Java 数据类型
接口依赖 (找负责人)	—	—	运行期	编译期

这里没有考虑系统的返回结果。REST 系统无论在服务定位、数据还是接口依赖方面, 耦合度都是很低的, 甚至没有。表中所示的 119 火警系统, 完全可以看作一个 REST 风格系统, 公众身边还有很多这样的便利系统。当然, 其它类型风格的系统也是存在的, 因为需要更加复杂和严格的规程。

也可以看出, 公众与 REST 系统的耦合度是非常小的, 只要记住 119 号码并报告起火位置, 系统就能发生作用了。而 SOAP 系统则增加了耦合度, 如果 WSDL 文件变更了, 客户端需要构建新结构的 SOAP 消息。对于 RMI (属于 RPC) 系统, 如果服务的 Remote 接口变更了, 则服务端和客户端要修改代码并通过程序的编译才能继续运行, 否则系统可能就失效了, 这几乎是完全耦合了。

虽然 RPC 看上去有很多弊端, 但 RPC 在小范围使用却是非常高效的。这就好比, 要 100 个人行动一致是比较容易的, 但要让 10000 个人行动一致, 就很困难了, 而且很自然, 我们会把 10000 个人分成 100 个组, 每组 100 个人。所以, 松散耦合是需要的, 但紧密耦合也是需要的。

很明显 REST 不是 RPC, 他们之间的差异很大。但是 SOAP 是不是 RPC (看上去比较接近), 争议还是挺大的。在 SOAP 文档中, `<soap binding>` 元素有一个 style 属性, 它的值可以是 rpc 或 document^[7], 着实又增加更多的混乱。其实 SOAP 包含的数据类型是中性的, 独立于任何程序语言, 并且对于接口的依赖主要在程序运行阶段, 因此和 RPC 还是有所区别的, 但 SOAP 也能提供 RPC 功能。

REST 和 SOAP 两类型的 Web Services 的重要特性, 汇总如表 2 所示, 比较的目的并不是为了分出孰优孰劣, 而是更透彻地去理解他们的特性。

表 2 REST 和 SOAP 系统的特性比较

特性	REST	SOAP
复杂度	简单	复杂
耦合度	低	高
应用规模	很大	中小
地址模式 ^[8]	URL 和服务一一对应	URL 和服务一对多
通用接口	4 个 HTTP 方法	没有, 自定义
交互类型	人机交互	机机协作
缓存支持	是	很难
服务发生变更	人的自我学习和适应	SOA 系统设计者人工干预

3 REST 和 SOAP 两种 Web 服务的结合

拥护 REST 和拥护 SOAP 的人都很多, 你应该加入哪个阵

营? 这个问题就好比该用 Java 还是用 .net, 其实真的很难回答。那就实行实用主义, 只要好用就可以用。于是当你构建 Web Services 的时候, 不需要再犹豫是用 REST 还是 SOAP? 两者都可以使用, 只要有实际的需要。下面从概念层面和设计层面来讨论 REST 和 SOAP 两种 Web Services 的结合。

3 1 概念层面

REST 反映资源状态的转移变化, 属于状态机的范畴; SOAP 是一个通信协议, 可用于分布式异构系统之间的交互。虽然两种系统都存在交互, 但交互的方式有很大不同, 如表 3 所示。

表 3 REST 和 SOAP 交互方式比较

交互的要素	REST	SOAP
信息类型 (数据)	文本、json、xml	XML
上下文	没有或模糊的	XML Schema WSDL 服务描述等
交互模式	请求响应	请求响应、会话

REST 系统的交互简单、高效但不保证可靠性; SOAP 系统的交互更复杂、承诺更多的保证, 但损失效率。RESTful Web Services 是一种轻量级的服务, 而 SOAP Web Services 是一种重量级的服务, 两者适合不同的场合。在结合使用 REST 和 SOAP 系统的时候, 需要考虑的主要因素有: 带宽、规模、事务、功能等。这里简单说明一下两种类型的系统适合的场合。

- REST 的系统适合如下的场合
 - (1) Web Services 完全是无状态的;
 - (2) 服务的消费者 (往往指人) 应能理解服务提供者的约定, 并能适应服务的变化;
 - (3) 占用较小的带宽, 提供更多的连接^[5];
 - (4) 能较容易地融入已运行的站点, 服务和页面可以很好地混合。
- SOAP 的系统适合如下的场合
 - (1) 有正式的契约精确描述 Web Services 提供什么接口, 服务的消费者和提供者都要能理解契约并按合同办事;
 - (2) 架构能应付复杂的非功能性的需求, 如事务、安全、协调等^[5];
 - (3) 适合机机交互 (即服务器到服务器的交互, 而非人机交互), 这一点和 Jean Jacques Dubray 的观点正好一致的^[9]。

REST 和 SOAP 系统的结合处在于 Web Server (App Server 也可)。

3 2 设计层面

由于 REST 和 SOAP 两种 Web Services 系统的交互方式差别很大, 因此 REST 系统应该处在离客户更近的地方, 而 SOAP 系统处在离客户更远的地方 (当然这是逻辑上的)。图 1 为两种系统结合的示意图。

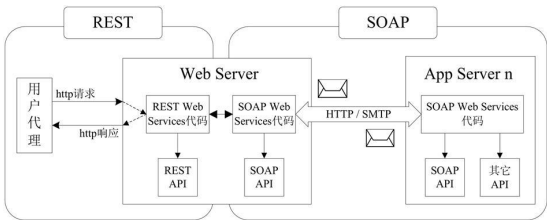


图 1 REST 和 SOAP 两种 Web Services 结合的结构图

为了模型的简单, 图中的某些地方做了简化, 因此有必要进一步说明:

- (1) API 包含接口和特定实现; (2) REST Web Services 代

码: 包含若干资源 (服务也是一种资源); (3) SOAP Web Services 代码: 只包含服务; (4) 信封: SOAP 消息; (5) SOAP API 包含 SOAP WSDL、XML 等 Web Services 等所需 API; (6) Web Server 和 App Server 的关系: 一对多的关系, 图中 n 表示任意一个。

这样的结合是很自然的, 没有必要做过多的解释, 而且一些公司和软件开发人员已经开始意识到这一点了^[10]。

4 REST 和 SOAP Web Service 的 Java 实现

RESTful Web Service 服务端的 Java 实现, 可以有两种方式: 一种利用 JAX-WS 2.0, 另一种则利用 JAX-RS^[6]。下面对用 Java 实现 REST 和 SOAP 两种 Web Services 做一些说明:

- (1) 利用 JAX-RS API 1.0 此 API 是 SUN JSR 311 (2008 9 8 发布 1.0 最终版) 的实现, 是稳定的产品版本, 因此可以用于正式的开发;
- (2) REST 客户端可使用目前很热门的 Ajax 技术来加强客户端和服务端之间的交互, 并使用一些 Javascript 库如 JQuery 来辅助开发, 为处理 json 和 XML 数据提供更多的便利;
- (3) 使用注解 (annotations) 来简化 RESTful Web Services 的开发, 可以减少很多模板代码, 并且可以减轻 XML 配置的烦恼。虽然注解的机制比较复杂, 但是使用注解对于开发人员并不困难。不过对于注解, 尚有一些担忧, JAX-RS 中包含的注解 (如 @GET、@POST 等), 在 Servlet 3.0 (目前是早期草案) 将被重用, 最终这些注解会移动到 Common Annotations specification (JSR 250) 中^[11]。所以, 注解的设计将考验 Java 类库的设计人员, 当然也会影响我们的代码;
- (4) 开发 SOAP Web Services 恐怕对于大多数人来说还是一个挑战。不过这个情况正在得到很大改善, 从 JDK (Java 的类库) 的发展来看, Java 6 在 Web Services 方面增加了很多包, 以后还会继续加强;

(5) XQuery 1.0 规范也于 2007 年发布, 我们几乎可以像使用 sql 操作关系数据一样用 XQuery 操作 XML, 有了 XQuery, Java 可以方便地从 XML 中获取数据, 这样可以减少 Web Services 中的 Java 和 XML 之间复杂的绑定和映射, 极大方便 Web Services 的开发。当然 XQuery 还需要完善, 如何使用 XQuery 和其它技术更有效地开发 SOAP Web Services 还需要进行更多的研究工作。

5 展 望

在分布式计算环境中, 松散组合的系统可能比紧密耦合的系统更有前景。Web 系统就是一个很好的证明。RESTful Web Services 可以构造松散组合的系统, 将会得到大量应用。我们期待的 Web Services 时代, 在 REST 和 SOAP 的携手下, 应该不会再跳票了。

参 考 文 献

[1] Tim O'Reilly. 什么是 Web 2.0 [J]. 互联网周刊, 2005, 40: 11-22.
[2] Uche Ogburn. The Past, Present and Future of Web Services part 1. 2002-9-28. [http://www.WebServices.org/categories/enterprise/strategy_architecture/the_past_present_and_future_of_Web_Services_part_1/\(go\)/Articles](http://www.WebServices.org/categories/enterprise/strategy_architecture/the_past_present_and_future_of_Web_Services_part_1/(go)/Articles)
[3] Roy Thomas Fielding. Architectural Styles and the Design of Network

- based Software Architectures[D]. UNIVERSITY OF CALIFORNIA, 2000: 88.
- [4] Roy Thomas Fielding. REST APIs must be hypertext driven. 2008-10-20. <http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven>.
- [5] Sameer Tyagi. RESTful Web Services. 2006-08. <http://java.sun.com/developer/technicalArticles/WebServices/restful/>.
- [6] 袁赞. Java 与 RESTful Web Services[J]. 电脑知识与技术: 学术交流, 2007, 21.
- [7] 陆小芳. Web 服务的两种调用模型的比较及开发[J]. 计算机应用, 2005, 25(1): 78-80.
- [8] 许卓明. 基于 RPC 和基于 REST 的 Web 服务交互模型比较分析[J]. 计算机工程, 2003, 29(20): 6-8.
- [9] 胡键. REST 在 SOA 中适合哪个位置. 2008-11-3. <http://www.irfoq.com/cn/news/2008/11/restsoa>.
- [10] Pingpangsong. SOA 2006 年终回顾以及 2007 展望(一). 2006-12-16. <http://pingpangsong.javaeye.com/blog/39348>.
- [11] Rajiv Mordani. Java™ Servlet Specification Version 3.0 early draft[M]. Sun Microsystems Inc. 2008, 5: 67.

(上接第 275 页)

测系统的检测, 攻击者一般会采取更加复杂的攻击模式。攻击者需要设计和实现一个可以自动检测不同操作系统并实施相应攻击的攻击代码(基本的逻辑结构如算法 1 所示)。

算法 1 攻击运行于多操作系统的软件的代码逻辑:

```
os_ret ( os_test() );
if is_win( os_ret ) then
    win_attack_code( );
else if is_unix( os_ret ) then
    unix_attack_code( );
else if is_mac( os_ret ) then
    mac_attack_code( );
end if
```

实现算法 1 中的攻击代码在现实情况中是非常困难的, 主要有以下两个原因。第一点是实现 os_test() 方法时, 必须保证该部分的(二进制)代码可以正常运行在不同的操作系统上, 即必须保证应用程序不崩溃, 并且根据操作系统的不同而返回不同的结果; 第二点是这类攻击代码, 大幅增加了攻击代码的长度, 通常情况下会超过了漏洞所提供的缓冲区的大小。

另一个相关的证据是卡巴斯基(Kaspersky)实验室所提供的病毒列表中至今只发现了三个可以跨越操作系统的病毒。卡巴斯基实验室的一份声明中指出这三个病毒都是理论上的示例(proof of concept)型恶意程序, 只是用来演示这种攻击是可能的, 这三个病毒中的任何一个至今都没有观察到有实际的应用实例。

4 结 论

本文分析了整个 2007 年所有公布的软件漏洞, 其中着重分析了应用软件漏洞, 来衡量软件相异性对于提高系统安全的有效性。分析的结果总结如下:

22.5% (590/2627) 的 CVE 漏洞是共享于多个软件产品中的, 其中 7.1% (29/410) 的漏洞是共享于功能相同的多个软件之间的。对于这部分的漏洞所对应的 69 个(29 组) 互相可替代

软件, 同一个攻击代码有 70% (14/20) 的几率可以攻破这些软件。因此即使类似于行为距离的入侵检测系统中使用的都是这些有漏洞的软件产品, 那么这些检测系统也只有低于 1.1% ($22.5\% \times 7.1\% \times 70\%$) 的几率会无效。

45.7% 的应用软件具有可以运行在多操作系统平台上的官方发行版本。在这些可以运行在多操作系统上的软件, 其中的 84.7% 的软件运行在不同操作系统上时共享同样的漏洞。但是由于内存布局和机器指令码上的显著差异, 使得构造可以同时攻击这些跨平台的软件的恶意代码的难度非常大。至今尚未发现此类跨平台恶意攻击程序的实际应用。

这些结果最终说明了利用现有的应用程序的相异性来构造入侵检测系统是十分有效的, 尤其是能够找到相同功能的互相可替代的软件时。本文主要的局限是耗费了大量的人工分析, 部分结果来源于采样分析。以后的研究中可以通过一些信息检索和人工智能的方法来提高分析效率, 并增加样本的覆盖面。

参 考 文 献

- [1] Elena Gabriela Barrantes David H Ackerly, Trek S Palmer et al. Randomized instruction set emulation to disrupt binary code injection attacks[C] // CCS'03: Proceedings of the 10th ACM conference on Computer and communications security, New York, NY, USA, 2003: 281-289.
- [2] Sandeep Bhakar, Daniel C DuVamey, Sekar r. Address obfuscation: an efficient approach to combat a board range of memory error exploits[C] // Proceedings of the 12th conference on USENIX Security Symposium, Berkeley, CA, USA, 2003: 8.
- [3] Cox B, Evans D, Filipi A, et al. N-variant systems - A secretless framework for security through diversity[C] // Proceedings of the 15th USENIX Security Symposium, August 2006.
- [4] Debin Gao Michael K Reiter Dawn Song. Behavioral distance for intrusion detection[C] // Proceedings of the 8th International Symposium on Recent Advances in Intrusion Detection (RAID 2005), 2005: 63-81.
- [5] Debin Gao Michael K Reiter Dawn Song. Behavioral distance measurement using hidden markov models[C] // Proceedings of the 9th International Symposium on Recent Advances in Intrusion Detection (RAID 2006), 2006: 19-40.
- [6] Geerd, Bace R, Gutmann P, et al. The cost of monopoly. Technical report. CCA, 2003.
- [7] Gaurav S Kç Angeles D Kermiyis Vassilis Prevekkis. Countering code-injection attacks with instruction-set randomization[C] // Proceedings of the 10th ACM conference on Computer and communications security, New York, NY, USA, ACM, 2003: 272-280.
- [8] Richard C Linger. Systematic generation of stochastic diversity as an intrusion barrier in survivable systems software[C] // HICSS'99: Proceedings of the Thirty-Second Annual Hawaii International Conference on System Sciences Volume 3 Washington, DC, USA, IEEE Computer Society, 1999: 3062.
- [9] Adam J O'Donnell Harish Sethu. On achieving software diversity for improved network security using distributed coloring algorithms[C] // CCS'04: Proceedings of the 11th ACM conference on Computer and communications security, New York, NY, USA, ACM, 2004: 121-131.
- [10] Salton G, Wong A, Yang C S. A vector space model for automatic indexing[J]. Communications of the ACM, 1975, 18(11): 613-620.
- [11] Mark Stamp. Risks of monoculture[J]. Communications of the ACM, 2004, 47(3): 120.