SOAP:

简单对象访问协议（Simple Object Access Protocol，SOAP）是一种基于 XML 的协议，可以和现存的许多因特网协议和格式结合使用，包括超文本传输协议（HTTP），简单邮件传输协议（SMTP），多用途网际邮件扩充协议（MIME），基于“通用”传输协议是 SOAP的一个优点。它还支持从消息系统到远程过程调用（Remote Procedure Call，RPC）等大量的应用程序。SOAP提供了一系列的标准，如WSRM（WS-Reliable Messaging）形式化契约确保可靠性与安全性，确保异步处理与调用；WS-Security、WS-Transactions和WS-Coordination等标准提供了上下文信息与对话状态管理。

相对而言，SOAP协议属于复杂的、重量级的协议，当前随着Web2.0的兴起，表述性状态转移（Representational State Transfer，REST）逐步成为一个流行的架构风格。

REST是一种轻量级的Web Service架构风格，其实现和操作比SOAP和XML-RPC更为简洁，可以完全通过HTTP协议实现，还可以利用缓存Cache来提高响应速度，性能、效率和易用性上都优于SOAP协议。REST架构对资源的操作包括获取、创建、修改和删除资源的操作正好对应HTTP协议提供的GET、POST、PUT和DELETE方法，这种针对网络应用的设计和开发方式，可以降低开发的复杂性，提高系统的可伸缩性。REST架构尤其适用于完全无状态的CRUD（Create、Read、Update、Delete，创建、读取、更新、删除）操作。

基于REST的软件体系结构风格（Software Architecture Style）称之为面向资源体系架构（Resource-oriented Architecture，ROA）。按照REST原则设计的软件、体系结构，通常被称为“REST式的”（RESTful），在本文中以下称之为RESTful Web服务，以便于和基于SOAP的Web服务区别。

服务器端采用J2EE，客户端采用JSP、Flex、JavaFX、AIR等可以直接调用Servlet，其他的实现技术基本上不能直接调用，但是无论是那种客户端，对于基于SOAP的Web服务或者基于RESTful Web服务务都是支持的，如AJAX的 XMLHttpRequest、Flex的HTTPService等。

HTTP 的 GET、HEAD 请求本质上应该是安全的调用，即：GET、HEAD 调用不会有任何的副作用，不会造成服务器端状态的改变。对于服务器来说，客户端对某一 URI 做 n 次的 GET、HAED 调用，其状态与没有做调用是一样的，不会发生任何的改变。

HTTP 的 PUT、DELTE 调用，具有幂指相等特性 , 即：客户端对某一 URI 做 n 次的 PUT、DELTE 调用，其效果与做一次的调用是一样的。HTTP 的 GET、HEAD 方法也具有幂指相等特性。

HTTP 这些标准方法在原则上保证你的分布式系统具有这些特性，以帮助构建更加健壮的分布式系统。

安全控制

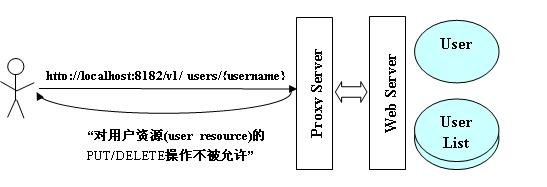
为了说明问题，基于上面的在线用户管理系统，我们给定以下场景：

参考一开始我们给出的用例图，对于客户端 Client2，我们只希望它能以只读的方式访问 User 和 User List 资源，而 Client1 具有访问所有资源的所有权限。

如何做这样的安全控制？

通行的做法是：所有从客户端 Client2 发出的 HTTP 请求都经过代理服务器 (Proxy Server)。代理服务器制定安全策略：所有经过该代理的访问 User 和 User List 资源的请求只具有读取权限，即：允许 GET/HEAD 操作，而像具有写权限的 PUT/DELTE 是不被允许的。

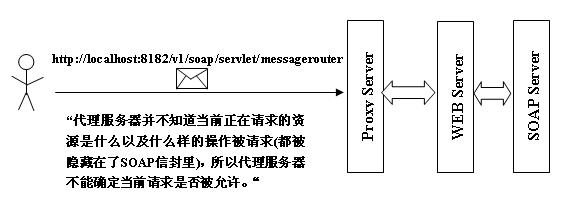
如果对于 REST，我们看看这样的安全策略是如何部署的。如下图所示：

REST 与代理服务器 (Proxy Servers)    
[](http://static.oschina.net/uploads/img/201312/12205116_b7R4.jpg)   

一般代理服务器的实现根据 (URI, HTTP Method) 两元组来决定 HTTP 请求的安全合法性。

当发现类似于（http://localhost:8182/v1/users/{username}，DELETE）这样的请求时，予以拒绝。

对于 SOAP，如果我们想借助于既有的代理服务器进行安全控制，会比较尴尬，如下图：

SOAP 与代理服务器 (Proxy Servers)    
[](http://static.oschina.net/uploads/img/201312/12205117_Plxo.jpg)   

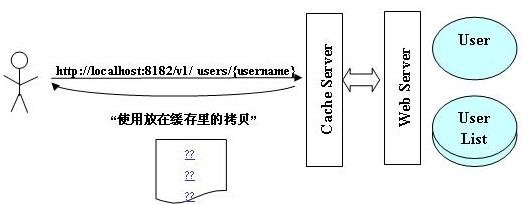
所有的 SOAP 消息经过代理服务器，只能看到（  
http://localhost:8182/v1/soap/servlet/messagerouter  
, HTTP POST）这样的信息，如果代理服务器想知道当前的 HTTP 请求具体做的是什么，必须对 SOAP 的消息体解码，这样的话，意味着要求第三方的代理服务器需要理解当前的 SOAP 消息语义，而这种 SOAP 应用与代理服务器之间的紧耦合关系是不合理的。

关于缓存

众所周知，对于基于网络的分布式应用，网络传输是一个影响应用性能的重要因素。如何使用缓存来节省网络传输带来的开销，这是每一个构建分布式网络应用的开发人员必须考虑的问题。

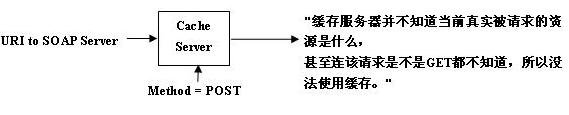
HTTP 协议带条件的 HTTP GET 请求 (Conditional GET) 被设计用来节省客户端与服务器之间网络传输带来的开销，这也给客户端实现 Cache 机制 ( 包括在客户端与服务器之间的任何代理 ) 提供了可能。HTTP 协议通过 HTTP HEADER 域：If-Modified-Since/Last- Modified，If-None-Match/ETag 实现带条件的 GET 请求。

REST 的应用可以充分地挖掘 HTTP 协议对缓存支持的能力。当客户端第一次发送 HTTP GET 请求给服务器获得内容后，该内容可能被缓存服务器 (Cache Server) 缓存。当下一次客户端请求同样的资源时，缓存可以直接给出响应，而不需要请求远程的服务器获得。而这一切对客户端来说都是透明的。

REST 与缓存服务器 (Cache Server)    
[](http://static.oschina.net/uploads/img/201312/12205119_plIS.jpg)   

而对于 SOAP，情况又是怎样的呢？

使用 HTTP 协议的 SOAP，由于其设计原则上并不像 REST 那样强调与 Web 的工作方式相一致，所以，基于 SOAP 应用很难充分发挥 HTTP 本身的缓存能力。

SOAP 与缓存服务器 (Cache Server)  
[](http://static.oschina.net/uploads/img/201312/12205120_CxXS.jpg)

两个因素决定了基于 SOAP 应用的缓存机制要远比 REST 复杂：

其一、所有经过缓存服务器的 SOAP 消息总是 HTTP POST，缓存服务器如果不解码 SOAP 消息体，没法知道该 HTTP 请求是否是想从服务器获得数据。

其二、SOAP 消息所使用的 URI 总是指向 SOAP 的服务器，如本文例子中的   
http://localhost:8182/v1/soap/servlet/messagerouter  
，这并没有表达真实的资源 URI，其结果是缓存服务器根本不知道那个资源正在被请求，更不用谈进行缓存处理。

关于连接性

在一个纯的 SOAP 应用中，URI 本质上除了用来指示 SOAP 服务器外，本身没有任何意义。与 REST 的不同的是，无法通过 URI 驱动 SOAP 方法调用。例如在我们的例子中，当我们通过

getUserList SOAP 消息获得所有的用户列表后，仍然无法通过既有的信息得到某个具体的用户信息。唯一的方法只有通过 WSDL 的指示，通过调用 getUserByName 获得，getUserList 与 getUserByName 是彼此孤立的。

而对于 REST，情况是完全不同的：通过   
http://localhost:8182/v1/users  
URI 获得用户列表，然后再通过用户列表中所提供的 LINK 属性，例如   
< link>http://localhost:8182/v1/users/tester</link>  
获得 tester 用户的用户信息。这样的工作方式，非常类似于你在浏览器的某个页面上点击某个 hyperlink, 浏览器帮你自动定向到你想访问的页面，并不依赖任何第三方的信息

REST 构建的系统其系统的扩展能力要强于 SOAP，这可以体现在它的统一接口抽象、代理服务器支持、缓存服务器支持等诸多方面, 而SOAP的成熟性可以给需要提供给多开发语言的，多传输方式的，对于安全性要求较高的接口设计带来便利