**REST和SOAP Web Service的比较**

**REST似乎在一夜间兴起了，这可能引起一些争议，反对者可以说REST是WEB诞生之始甚而是HTTP出现之日就相伴而生的原则。但是毋庸置疑的事实是，在Google和Yahoo等网络巨头发布的相同功能的Web Service API中，REST无疑受到更多的青睐，因此是不是可以这样说：RPC在一夜之间衰落了？  
  
在一篇作业的小文章里讨论整套RPC的原理，无疑太过庞大了，况且RPC在Web Service领域的应用也无过XML-RPC以及由此延伸的SOAP而已。在原理上唯一重要的，是传统程序的函数调用和返回在RPC中被请求和应答代替了而已。既然如此，在讨论REST之前先阐述SOAP，可能是合乎逻辑的顺序。  
  
什么是SOAP?  
  
SOAP (Simple Object Access Protocol) 顾名思义，是一个严格定义的信息交换协议，用于在Web Service中把远程调用和返回封装成机器可读的格式化数据。事实上SOAP数据使用XML数据格式，定义了一整套复杂的标签，以描述调用的远程过程、参数、返回值和出错信息等等。而且随着需要的增长，又不得增加协议以支持安全性，这使SOAP变得异常庞大，背离了简单的初衷。另一方面，各个服务器都可以基于这个协议推出自己的API，即使它们提供的服务极其相似，定义的API也不尽相同，这又导致了WSDL的诞生。WSDL (Web Service Description Language) 也遵循XML格式，用来描述哪个服务器提供什么服务，怎样找到它，以及该服务使用怎样的接口规范，简言之，服务发现。现在，使用Web Service的过程变成，获得该服务的WSDL描述，根据WSDL构造一条格式化的SOAP请求发送给服务器，然后接收一条同样SOAP格式的应答，最后根据先前的WSDL解码数据。绝大多数情况下，请求和应答使用HTTP协议传输，那么发送请求就使用HTTP的POST方法。  
  
  
什么是REST？  
  
REST (REpresentational State Transfort) 形式上应该表述为客户端通过申请资源来实现状态的转换，在这个角度系统可以看成一台虚拟的状态机。抛开R. T. Fielding博士论文里晦涩的理论不说，REST应该满足这样的特点：1)客户端和服务器结构；2)连接协议具有无状态性；3)能够利用Cache机制增进性能；4)层次化的系统；5)按需代码。说到底，REST只是一种架构风格，而不是协议或标准。但这种新的风格（也许已经历史悠久？）对现有的以SOAP为代表的Web Service造成的冲击也是革命性的，因为它面向资源，甚至连服务也抽象成资源，因为它和HTTP紧密结合，因为它服务器无状态。  
  
  
REST与SOAP的区别  
  
因为SOAP并不假定传输数据的下层协议，因此必须设计为能在各种协议上运行。即使绝大多数SOAP是运行在HTTP上，使用URI标识服务，SOAP也仅仅使用POST方法发送请求，用一个唯一的URI标识服务的入口。举一个图书馆在线查询管理系统为例，服务提供者必须为每一本书提供一个内部标识，然后可能定义一个listBooks操作来返回一系列图书，一个getBook操作来返回指定的图书，一个createBook操作来向数据库加入新增的图书，一个deleteBook操作来删除作废的图书，每个操作都有各自的参数，尤其是用内部标识来标识操作的图书。这种设计被诟病之处，在于deleteBook操作也要用POST方法来发送，而其实HTTP协议有更和逻辑的DELETE方法可用。REST正是这样设计的，REST为每一个资源（此处是图书）指定一个唯一的URI，而用HTTP的4种方法GET、POST、PUT、DELETE直观地表示获取、创建、更新和删除图书。同时图书集合也是和单本的图书不同的资源，如果用/books来代表图书列表，/books/ID来代表标识为ID的图书，那么对/books的GET操作就代表返回整个图书列表，对/books/ID的DELETE操作代表删除指定的图书，等等。**

**成熟度：**

SOAP虽然发展到现在已经脱离了初衷，但是对于异构环境服务发布和调用，以及厂商的支持都已经达到了较为成熟的情况。不同平台，开发语言之间通过SOAP来交互的web service都能够较好的互通（在部分复杂和特殊的参数和返回对象解析上，协议没有作很细致的规定，导致还是需要作部分修正）

REST国外很多大网站都发布了自己的开发API，很多都提供了SOAP和REST两种Web Service，根据调查部分网站的REST风格的使用情况要高于SOAP。但是由于REST只是一种基于Http协议实现资源操作的思想，因此各个网站的REST实现都自有一套，在后面会讲诉各个大网站的REST API的风格。也正是因为这种各自实现的情况，在性能和可用性上会大大高于SOAP发布的web service，但统一通用方面远远不及SOAP。由于这些大网站的SP往往专注于此网站的API开发，因此通用性要求不高。

由于没有类似于SOAP的权威性协议作为规范，REST实现的各种协议仅仅只能算是私有协议，当然需要遵循REST的思想，但是这样细节方面有太多没有约束的地方。REST日后的发展所走向规范也会直接影响到这部分的设计是否能够有很好的生命力。

总的来说SOAP在成熟度上优于REST。

**效率和易用性：**

       SOAP协议对于消息体和消息头都有定义，同时消息头的可扩展性为各种互联网的标准提供了扩展的基础，WS-\*系列就是较为成功的规范。但是也由于SOAP各种需求不断扩充其本身协议的内容，导致在SOAP处理方面的性能有所下降。同时在易用性方面以及学习成本上也有所增加。

       REST被人们的重视，其实很大一方面也是因为其高效以及简洁易用的特性。这种高效一方面源于其面向资源接口设计以及操作抽象简化了开发者的不良设计，同时也最大限度的利用了Http最初的应用协议设计理念。同时，在我看来REST还有一个很吸引开发者的就是能够很好的融合当前Web2.0的很多前端技术来提高开发效率。例如很多大型网站开放的REST风格的API都会有多种返回形式，除了传统的xml作为数据承载，还有（JSON,RSS,ATOM）等形式，这对很多网站前端开发人员来说就能够很好的mashup各种资源信息。

       因此在效率和易用性上来说，REST更胜一筹。

**安全性：**

       这点其实可以放入到成熟度中，不过在当前的互联网应用和平台开发设计过程中，安全已经被提到了很高的高度，特别是作为外部接口给第三方调用，安全性可能会高过业务逻辑本身。

       SOAP在安全方面是通过使用XML-Security和XML-Signature两个规范组成了WS-Security来实现安全控制的，当前已经得到了各个厂商的支持，.net ，php ，java 都已经对其有了很好的支持（虽然在一些细节上还是有不兼容的问题，但是互通基本上是可以的）。

       REST没有任何规范对于安全方面作说明，同时现在开放REST风格API的网站主要分成两种，一种是自定义了安全信息封装在消息中（其实这和SOAP没有什么区别），另外一种就是靠硬件SSL来保障,但是这只能够保证点到点的安全，如果是需要多点传输的话SSL就无能为力了。安全这块其实也是一个很大的问题，今年在BEA峰会上看到有演示采用SAML2实现的网站间SSO，其实是直接采用了XML-Security和XML-Signature，效率看起来也不是很高。未来REST规范化和通用化过程中的安全是否也会采用这两种规范，是未知的，但是加入的越多，REST失去它高效性的优势越多。

**应用设计与改造：**

       我们的系统要么就是已经有了那些需要被发布出去的服务，要么就是刚刚设计好的服务，但是开发人员的传统设计思想让REST的形式被接受还需要一点时间。同时在资源型数据服务接口设计上来说按照REST的思想来设计相对来说要容易一些，而对于一些复杂的服务接口来说，可能强要去按照REST的风格来设计会有些牵强。这一点其实可以看看各大网站的接口就可以知道，很多网站还要传入function的名称作为参数，这就明显已经违背了REST本身的设计思路。

       而SOAP本身就是面向RPC来设计的，开发人员十分容易接受，所以不存在什么适应的过程。

**总的来说，其实还是一个老观念，适合的才是最好的**

       技术没有好坏，只有是不是合适，一种好的技术和思想被误用了，那么就会得到反效果。REST和SOAP各自都有自己的优点，同时如果在一些场景下如果去改造REST，其实就会走向SOAP（例如安全）。

       REST对于资源型服务接口来说很合适，同时特别适合对于效率要求很高，但是对于安全要求不高的场景。而SOAP的成熟性可以给需要提供给多开发语言的，对于安全性要求较高的接口设计带来便利。所以我觉得纯粹说什么设计模式将会占据主导地位没有什么意义，关键还是看应用场景。

       同时很重要一点就是不要扭曲了REST现在很多网站都跟风去开发REST风格的接口，其实都是在学其形，不知其心，最后弄得不伦不类，性能上不去，安全又保证不了，徒有一个看似象摸象样的皮囊。

**REST的优点  
  
REST简单而直观，把HTTP协议利用到了极限，在这种思想指导下，它甚至用HTTP请求的头信息来指明资源的表示形式（如果一个资源有多种形式的话，例如人类友善的页面还是机器可读的数据？），用HTTP的错误机制来返回访问资源的错误。由此带来的直接好处是构建的成本减少了，例如用URI定位每一个资源可以利用通用成熟的技术，而不用再在服务器端开发一套资源访问机制。又如只需简单配置服务器就能规定资源的访问权限，例如通过禁止非GET访问把资源设成只读。  
  
服务器无状态带来了更多额外好处，因为每次请求都包含响应需要的所有信息，所有状态信息都存储在客户端，服务器的内存从庞大的状态信息中解放出来。而且现在即使一台服务器突然死机对客户的影响也微乎其微，因为另一台服务器可以马上代替它的位置，而不需要考虑恢复状态信息。更多的缓存也变成可能，而之前由于服务器有状态，对同一个URI的请求可能导致完全不同的响应。总体结果是，网络的容错性和延展性都增强了，这些本来是WEB设计的初衷，日趋复杂和定制的WEB把它们破坏了，现在REST又返璞归真，试图把Web Service带回简单的原则中来。  
  
  
REST是万能的吗？  
  
但是REST就是万能的吗？无状态带来了巨大的优势，同时也带来了难以解决的问题，例如，怎样授权特定用户才能使用的服务？怎样验证用户身份？如果坚持服务器无状态，也就是不记录用户登录状态，势必要求每一次服务请求都包含完整的用户身份和验证信息。在这种情况下，怎样避免冒认？怎样避免用户信息泄漏？事实上，构建REST附属的安全机制已经在讨论中，其结果无非导致另一个SOAP：复杂的需求摧残了易用性。  
  
REST的支持者声称REST的请求和应答数据简单可读，而SOAP则需要一系列繁琐的封装；即使如此，SOAP仍然不能达到接口的一致性，不同的厂商有各自的接口，而REST只使用HTTP定义的方法，因此是通用的。事实确实如此吗？试想用REST实现两数求和的服务，如果按照建议的做法，把服务（此处是加法）作为一个资源，参数（此处是两个加数）作为请求的参数，结果以XML或JSON语法返回，是否比SOAP更简单易用？通用接口仍然没法达到，因为资源的名称、参数的名称、结果的格式仍然是服务提供者定义的。为了解决这个问题，提出了WASL(Web Application Description Language)来描述REST接口。WADL就像是WSDL的REST版，随着REST被应用到复杂的领域，SOAP的影子无处不在。  
  
  
面向资源和面向事务**

**REST在面向资源的应用中左右逢源，但在面向事务的应用中却未如人意。面向资源的应用操作简单，无非创建、读取、改变、删除几项，但是面向事务的应用不允许用户直接操作资源，用户只需向系统提交一个事务说明要求，然后等待事务的完成，就如一个网上银行的用户不直接修改账户和存款，而是提交一个事务告诉银行自己要转账。如果把这样的服务看成一种资源，通过向资源发送POST请求完成事务，那不过是SOAP的翻版而已，无论是这样，还是通过PUT来创建事务，都改变了系统的状态（资源本身未改变，此处是改变了用户的余额），显然违背了REST直观的初衷。  
  
事实上，一些Web Service提供者提供的REST API只有REST的外壳，传输的请求和应答全然是简化了的SOAP，这种新瓶装旧酒的做法只是加深了标准的分歧而已。归根结底REST无法简单地解决一些应用，因此我们只能看到SOAP在REST外壳下的借尸还魂。没有一项技术能一劳永逸地解决所有问题，只需要在预定的约束下优美地解决所在领域的问题就足够了。一项新技术推出的时候总是引来无数的跟风和吹捧，只有当尘埃落定之后才能得到中肯的评价。**