### REST WebService与SOAP WebService的比较

在SOA的基础技术实现方式中WebService占据了很重要的地位，通常我们提到WebService第一想法就是SOAP消息在各种传输协议上交互。近几年REST的思想伴随着SOA逐渐被大家接受，同时各大网站不断开放API提供给开发者，也激起了REST风格WebService的热潮。

SOAP

什么是SOAP，我想不用多说，google一把满眼都是。其实SOAP最早是针对RPC的一种解决方案，简单对象访问协议，很轻量，同时作为应用协议可以基于多种传输协议来传递消息（Http,SMTP等）。但是随着SOAP作为WebService的广泛应用，不断地增加附加的内容，使得现在开发人员觉得SOAP很重，使用门槛很高。在SOAP后续的发展过程中，WS-\*一系列协议的制定，增加了SOAP的成熟度，也给SOAP增加了负担。

REST

REST其实并不是什么协议也不是什么标准，而是将Http协议的设计初衷作了诠释，在Http协议被广泛利用的今天，越来越多的是将其作为传输协议，而非原先设计者所考虑的应用协议。SOAP类型的WebService就是最好的例子，SOAP消息完全就是将Http协议作为消息承载，以至于对于Http协议中的各种参数（例如编码，错误码等）都置之不顾。其实，最轻量级的应用协议就是Http协议。Http协议所抽象的get,post,put,delete就好比数据库中最基本的增删改查，而互联网上的各种资源就好比数据库中的记录（可能这么比喻不是很好），对于各种资源的操作最后总是能抽象成为这四种基本操作，在定义了定位资源的规则以后，对于资源的操作通过标准的Http协议就可以实现，开发者也会受益于这种轻量级的协议。

REST的思想归结以下有如下几个关键点：

1．面向资源的接口设计

所有的接口设计都是针对资源来设计的，也就很类似于我们的面向对象和面向过程的设计区别，只不过现在将网络上的操作实体都作为资源来看待，同时URI的设计也是体现了对于资源的定位设计。后面会提到有一些网站的API设计说是REST设计，其实是RPC-REST的混合体，并非是REST的思想。

2．抽象操作为基础的CRUD

这点很简单，Http中的get,put,post,delete分别对应了read,update,create,delete四种操作，如果仅仅是作为对于资源的操作，抽象成为这四种已经足够了，但是对于现在的一些复杂的业务服务接口设计，可能这样的抽象未必能够满足。其实这也在后面的几个网站的API设计中暴露了这样的问题，如果要完全按照REST的思想来设计，那么适用的环境将会有限制，而非放之四海皆准的。

3．Http是应用协议而非传输协议

这点在后面各大网站的API分析中有很明显的体现，其实有些网站已经走到了SOAP的老路上，说是REST的理念设计，其实是作了一套私有的SOAP协议，因此称之为REST风格的自定义SOAP协议。

4．无状态，自包含

这点其实不仅仅是对于REST来说的，作为接口设计都需要能够做到这点，也是作为可扩展和高效性的最基本的保证，就算是使用SOAP的WebService也是一样。

REST vs SOAP

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由于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。