REST 从资源的角度来观察整个网络，分布在各处的资源由URI确定，而客户端的应用通过URI来获取资源的表示方式。获得这些表徵致使这些应用程序转变了其状态。随着不断获取资源的表示方式，客户端应用不断地在转变着其状态，所谓表述性状态转移（Representational State Transfer）。

这一观点不是凭空臆造的，而是通过观察当前Web互联网的运作方式而抽象出来的。Roy Fielding 认为，

“设计良好的网络应用表现为一系列的网页，这些网页可以看作的虚拟的状态机，用户选择这些链接导致下一网页传输到用户端展现给使用的人，而这正代表了状态的转变。”

REST是设计风格而**不是**标准。REST通常基于使用HTTP，URI，和XML以及HTML这些现有的广泛流行的协议和标准。

* 资源是由URI来指定。
* 对资源的操作包括获取、创建、修改和删除资源，这些操作正好对应HTTP协议提供的GET、POST、PUT和DELETE方法。
* 通过操作资源的表现形式来操作资源。
* 资源的表现形式则是XML或者HTML，取决于读者是机器还是人，是消费web服务的客户软件还是web浏览器。当然也可以是任何其他的格式。

##### **REST的要求**

* 客户端和服务器结构
* 连接协议具有无状态性
* 能够利用Cache机制增进性能
* 层次化的系统
* 隨需代碼 - Javascript （可選）

**RESTful Web 服务**

RESTful Web 服务（也称为 RESTful Web API）是一个使用HTTP并遵循REST原则的Web服务。它从以下三个方面资源进行定义：URI，比如：http://example.com/resources/。

§ Web服务接受与返回的互联网媒体类型，比如：JSON，XML ，YAML 等。

§ Web服务在该资源上所支持的一系列请求方法（比如：POST，GET，PUT或DELETE）。

该表列出了在实现RESTful Web 服务时HTTP请求方法的典型用途。

**HTTP 请求方法在RESTful Web 服务中的典型应用**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **资源** | **GET** | **PUT** | **POST** | **DELETE** |
| **一组资源的URI，比如http://example.com/resources/** | **列出** URI，以及该资源组中每个资源的详细信息（后者可选）。 | 使用给定的一组资源**替换**当前整组资源。 | 在本组资源中**创建/追加**一个新的资源。 该操作往往返回新资源的URL。 | **删除** 整组资源。 |
| **单个资源的URI，比如http://example.com/resources/142** | **获取** 指定的资源的详细信息，格式可以自选一个合适的网络媒体类型（比如：XML、JSON等） | **替换/创建** 指定的资源。并将其追加到相应的资源组中。 | 把指定的资源当做一个资源组，并在其下**创建/追加**一个新的元素，使其隶属于当前资源。 | **删除** 指定的元素。 |

PUT 和 DELETE 方法是幂等方法。GET方法是安全方法 （不会对服务器端有修改，因此也是幂等的）。

不像基于SOAP的Web服务，RESTful Web服务并没有的“正式”标准。 这是因为REST是一种架构，而SOAP只是一个协议。虽然REST不是一个标准，但在实现RESTful Web服务时可以使用其他各种标准（比如HTTP，URL，XML，PNG等）。

**REST的优点**

* 可以利用缓存Cache来提高响应速度
* 通讯本身的无状态性可以让不同的服务器的处理一系列请求中的不同请求，提高服务器的扩展性
* 浏览器即可作为客户端，简化软件需求
* 相对于其他叠加在HTTP协议之上的机制，REST的软件依赖性更小
* 不需要额外的资源发现机制
* 在软件技术演进中的长期的兼容性更好

**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。

http://www.cnblogs.com/springyangwc/archive/2012/01/18/2325784.html