## **Representational State Transfer--REST**

[版权声明](http://www.google.com/url?q=http%3A%2F%2Fcreativecommons.org%2Flicenses%2Fby%2F3.0%2Fdeed.zh&sa=D&sntz=1&usg=AFQjCNH2Eee4p9dKvJlFzHXA4cT-upmT1Q)：转载时请以超链接形式标明文章原始出处和作者信息及[本声明](http://www.google.com/url?q=http%3A%2F%2Fbangzhuzhongxin.blogbus.com%2Flogs%2F11205960.html&sa=D&sntz=1&usg=AFQjCNGEKZxTYBx_i9jKZ3A92TMw3VC3uw)

[http://soido2324.blogbus.com/logs/60353767.html](http://www.google.com/url?q=http%3A%2F%2Fsoido2324.blogbus.com%2Flogs%2F60353767.html&sa=D&sntz=1&usg=AFQjCNHe3vIGgmfSSZxPQV8IToOgqcRQ1Q)

　　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来获取资源的表形。获得这些表形致使这些应用程序转变了其状态。随 着不断获取资源的表形，客户端应用不断地在转变着其状态，所谓表形化的状态转变（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](http://www.google.com/url?q=http%3A%2F%2Fwww.verywin.cn&sa=D&sntz=1&usg=AFQjCNFnMyXdMvBD-laf_rdd0DS7sIXJaQ)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。 "

　　要点及标准

　　需要注意的是，REST是一种设计风格而不是一个标准。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 资源的表形来操作资源。

　　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

　　Code On Demand - Javascript

　　关于状态

　　应该注意区别应用的状态和连接协议的状态。REST对于连接的无状态性 实际上要求每次经过无状态的连接协议传送的信息必须包含应用中所有的状态信息。

　　实现举例

　　例如，一个简单的网 络商店应用，

　　列举所有商品，

　　GET http://www.store.com/products

　 　具体某一件商品，

　　GET http://www.store.com/product/12345

　　下单购 买，

　　POST http://www.store.com/order，

　　 <purchase-order>

　　<item> ... </item>

　 　</purchase-order>

　　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

　 　相对与其他叠加在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是英 文Representational State Transfer的缩写，中文翻译为"表述性状态转移"，他是由Roy Thomas Fielding博士在他的论文 《Architectural Styles and the Design of Network-based Software Architectures》中提出的一个术语。REST本身只是为分布式超媒体系统设计的一种架构风格，而不是标准。

　 　基于Web的架构，实际上就是各种规范的集合，这些规范共同组成了Web架构。比如Http协议，比如客户端服务器模式，这些都是规范。每当我们在原有 规范的基础上增加新的规范，就会形成新的架构。而REST正是这样一种架构，他结合了一系列的规范，而形成了一种新的基于Web的架构风格。

　 　传统的Web应用大都是B/S架构，它包括了如下一些规范。

　　客户－服务器

　　这种规范的提出，改善了用户接口 跨多个平台的可移植性，并且通过简化服务器组件，改善了系统的可伸缩性。最为关键的是通过分离用户接口和数据存储这两个关注点，使得不同用户终端享受相同 数据成为了可能。

　　无状态性

　　无状态性是在客户－服务器约束的基础上添加的又一层规范。他要求通信必须在本质上 是无状态的，即从客户到服务器的每个request都必须包含理解该request所必须的所有信息。这个规范改善了系统的可见性（无状态性使得客户端和 服务器端不必保存对方的详细信息，服务器只需要处理当前request，而不必了解所有的request历史），可靠性（无状态性减少了服务器从局部错误 中恢复的任务量），可伸缩性（无状态性使得服务器端可以很容易的释放资源，因为服务器端不必在多个request中保存状态）。同时，这种规范的缺点也是 显而易见得，由于不能将状态数据保存在服务器上的共享上下文中，因此增加了在一系列request中发送重复数据的开销,严重的降低了效率。

　 　缓存

　　为了改善无状态性带来的网络的低效性，我们填加了缓存约束。缓存约束允许隐式或显式地标记一个response中的数据，这 样就赋予了客户端缓存response数据的功能，这样就可以为以后的request共用缓存的数据，部分或全部的消除一部分交互，增加了网络的效率。但 是用于客户端缓存了信息，也就同时增加了客户端与服务器数据不一致的可能，从而降低了可靠性。

　　B/S架构的优点是其部署非常方便， 但在用户体验方面却不是很理想。为了改善这种情况，我们引入了REST。

　　REST在原有的架构上增加了三个新规范：统一接口，分层 系统和按需代码。

　　统一接口

　　REST架构风格的核心特征就是强调组件之间有一个统一的接口，这表现在REST 世界里，网络上所有的事物都被抽象为资源，而REST就是通过通用的链接器接口对资源进行操作。这样设计的好处是保证系统提供的服务都是解耦的，极大的简 化了系统，从而改善了系统的交互性和可重用性。并且REST针对Web的常见情况做了优化，使得REST接口被设计为可以高效的转移大粒度的超媒体数据， 这也就导致了REST接口对其它的架构并不是最优的。

　　分层系统

　　分层系统规则的加入提高了各种层次之间的独立 性，为整个系统的复杂性设置了边界，通过封装遗留的服务，使新的服务器免受遗留客户端的影响，这也就提高了系统的可伸缩性。

　　按需代 码

　　REST允许对客户端功能进行扩展。比如，通过下载并执行applet或脚本形式的代码，来扩展客户端功能。但这在改善系统可扩 展性的同时，也降低了可见性。所以它只是REST的一个可选的约束。

　　REST的设计准则

　　REST架构是针对 Web应用而设计的，其目的是为了降低开发的复杂性，提高系统的可伸缩性。REST提出了如下设计准则：

　　网络上的所有事物都被抽象 为资源（resource）；

　　每个资源对应一个唯一的资源标识符（resource identifier）；

　 　通过通用的连接器接口（generic connector interface）对资源进行操作；

　　对资源的各种操作不会改变资 源标识符；

　　所有的操作都是无状态的（stateless）。

　　REST中的资源所指的不是数据，而是数据和表 现形式的组合，比如"最新访问的10位会员"和"最活跃的10为会员"在数据上可能有重叠或者完全相同，而由于他们的表现形式不同，所以被归为不同的资 源，这也就是为什么REST的全名是Representational State Transfer的原因。资源标识符就是URI(Uniform Resource Identifier)，不管是图片，Word还是视频文件，甚至只是一种虚拟的服务，也不管你是xml格式,txt文件格式还是其它文件格式，全部通过 URI对资源进行唯一的标识。

　　REST是基于Http协议的，任何对资源的操作行为都是通过Http协议来实现。以往的Web开发 大多数用的都是Http协议中的GET和POST方法，对其他方法很少使用，这实际上是因为对Http协议认识片面的理解造成的。Http不仅仅是一个简 单的运载数据的协议，而是一个具有丰富内涵的网络软件的协议。他不仅仅能对互联网资源进行唯一定位，而且还能告诉我们如何对该资源进行操作。Http把对 一个资源的操作限制在4个方法以内：GET,POST,PUT和DELETE，这正是对资源CRUD操作的实现。由于资源和URI是一一对应的，执行这些 操作的时候URI是没有变化的，这和以往的Web开发有很大的区别。正由于这一点，极大的简化了Web开发，也使得URI可以被设计成更为直观的反映资源 的结构，这种URI的设计被称作RESTful的URI。这位开发人员引入了一种新的思维[方式](http://www.google.com/url?q=http%3A%2F%2Fwww.verywin.cn&sa=D&sntz=1&usg=AFQjCNFnMyXdMvBD-laf_rdd0DS7sIXJaQ)：通过URL来设计 系统结构。当然了，这种设计[方式](http://www.google.com/url?q=http%3A%2F%2Fwww.verywin.cn&sa=D&sntz=1&usg=AFQjCNFnMyXdMvBD-laf_rdd0DS7sIXJaQ)对一些特定情况也是不适用的，也就是说不是所有的URI都可以RESTful 的。

　　REST之所以可以提高系统的可伸缩性，就是因为它要求所有的操作都是无状态的。由于没有了上下文(Context)的约束， 做分布式和集群的时候就更为简单，也可以让系统更为有效的利用缓冲池(Pool）。并且由于服务器端不需要记录客户端的一系列访问，也减少了服务器端的性 能。

　　使用REST架构

　　对于开发人员来说，关心的是如何使用REST架构，这里我们来简单谈谈这个问题。 REST不仅仅是一种崭新的架构，它带来的更是一种全新的Web开发过程中的思维[方式](http://www.google.com/url?q=http%3A%2F%2Fwww.verywin.cn&sa=D&sntz=1&usg=AFQjCNFnMyXdMvBD-laf_rdd0DS7sIXJaQ)：通过URL来设计系统结构。在 REST中，所有的URL都对应着资源，只要URL的设计是良好的，那么其呈现的系统结构也就是良好的。这点和TDD(Test Driven Development)很相似，他是通过测试用例来设计系统的接口，每一个测试用例都表示一系列用户的需求。开发人员不需要一开始就编写功能，而只需要 把需要实现的功能通过测试用例的形式表现出来即可。这个和REST中通过URL设计系统结构的[方式](http://www.google.com/url?q=http%3A%2F%2Fwww.verywin.cn&sa=D&sntz=1&usg=AFQjCNFnMyXdMvBD-laf_rdd0DS7sIXJaQ)类似，我们只需要根 据需求设计出合理地URL，这些URL不一定非要链接到指定的页面或者完成一些行为，只要它们能够直观的表现出系统的用户接口。根据这些URL，我们就可 以方便的设计系统结构。从REST架构的概念上来看，所有能够被抽象成资源的东西都可以被指定为一个URL，而开发人员所需要做的工作就是如何能把用户需 求抽象为资源，以及如何抽象的精确。因为对资源抽象的越为精确，对REST的应用来说就越好。这个和传统MVC开发模式中基于Action的思想差别就非 常大。设计良好的URL，不但对于开发人员来说可以更明确的认识系统结构，对使用者来说也方便记忆和识别资源，因为URL足够简单和有意义。按照以往的设 计模式，很多URL后面都是一堆参数，对于使用者来说也是很不方便的。

　　既然REST这么好用，那么是不是所有的Web应用都能采取 此种架构呢？答案是否定的。我们知道，直到现在为止，MVC(Model-View-Controller)模式依然是Web开发最普遍的模式，绝大多数 的公司和开发人员都采取此种架构来开发Web应用，并且其思维[方式](http://www.google.com/url?q=http%3A%2F%2Fwww.verywin.cn&sa=D&sntz=1&usg=AFQjCNFnMyXdMvBD-laf_rdd0DS7sIXJaQ)也停留于此。MVC模式由数据，视图和控制器构 成，通过事件(Event)触发Controller来改变Model和View。加上Webwork,Struts等开源框架的加入，MVC开发模式已 经相当成熟，其思想根本就是基于Action来驱动。从开发人员角度上来说，贸然接受一个新的架构会带来风险，其中的不确定因素太多。并且REST新的思 维[方式](http://www.google.com/url?q=http%3A%2F%2Fwww.verywin.cn&sa=D&sntz=1&usg=AFQjCNFnMyXdMvBD-laf_rdd0DS7sIXJaQ)是把所有用户需求抽象为资源，这在实际开发中是比较难做到的，因为并不是所有的 用户需求都能被抽象为资源，这样也就是说不是整个系统的结构都能通过REST的来表现。所以在开发中，我们需要根据以上2点来在REST和MVC中做出选 择。我们认为比较好的办法是混用REST和MVC，因为这适合绝大多数的Web应用开发，开发人员只需要对比较容易能够抽象为资源的用户需求采取REST 的开发模式，而对其它需求采取MVC开发即可。这里需要提到的就是ROR(Ruby on Rails)框架，这是一个基于Ruby语言的越来越流行的Web开发框架，它极大的提高了Web开发的速度。更为重要的是，ROR(从1.2版本起)框 架是第一个引入REST做为核心思想的Web开发框架，它提供了对REST最好的支持，也是当今最成功的应用REST的Web开发框架。实际上，ROR的 REST实现就是REST和MVC混用，开发人员采用ROR框架，可以更快更好的构建Web应用

　　 ================================ REST是一种看待分布式超媒体系统架构的[方式](http://www.google.com/url?q=http%3A%2F%2Fwww.verywin.cn&sa=D&sntz=1&usg=AFQjCNFnMyXdMvBD-laf_rdd0DS7sIXJaQ)。之所以我们需要关注它，是因为很多web应用都是以这种[方式](http://www.google.com/url?q=http%3A%2F%2Fwww.verywin.cn&sa=D&sntz=1&usg=AFQjCNFnMyXdMvBD-laf_rdd0DS7sIXJaQ)组织的。

　　2000年，Roy Fielding在他的博士论文的第5章[1]里总结了REST背后的思想。按照REST的做法，服务器与客户之间的交互都通过无状态的连接进行：所有与 交互状态有关的信息都被编码到请求与应答中。需要长时间存在的状态以一组可标识的资源形式保存在服务器上，客户端则以一组定义良好的(并且严格约束的)资 源标识符(也就是URL )来访问这些资源。REST将"资源的内容"与"资源的表像"区分开，这是为了支持高伸缩性计算而做的设计，同时自然地降低应用架构中的耦合。

　 　这段叙述中出现了不少抽象的内容。那么在实际工作中，REST究竟意味着什么？

　　首先，REST的规范化意味着网络设计者们清楚地 知道自己可以在何时何地缓存应答内容。这也就意味着大量的数据装载工作可以被消除掉，从而增加整个网络的性能和可靠性、减少用户等待。

　 　其次，REST的约束可以让应用程序更容易编写(以及维护)。REST化的应用程序无须操心如何实现可供其他应用程序远程访问的服务，它们只要为一组资 源提供一个普通(并且简单)的接口就行了。你的应用程序实现资源列举、创建、编辑和删除的[方式](http://www.google.com/url?q=http%3A%2F%2Fwww.verywin.cn&sa=D&sntz=1&usg=AFQjCNFnMyXdMvBD-laf_rdd0DS7sIXJaQ)，你的客户做剩下的 事情。

　　说得再具体一点，在REST架构中，我们会使用一组简单的动词来操作各种各样的名词。如果使用HTTP，这些动词就对应于 HTTP方法(典型的方法包括GET、PUT、POST和DELETE等)，名词则是应用程序中的资源。我们用URL给资源命名。

　　 举例来说，一个内容管理系统可能包含一系列文章。其中潜在地存在着两类资源；其一是一篇一篇独立的文章，每篇文章构成一项资源；这里还有另一类资源：文章 的集合。

　　要获取所有文章的列表，我们可以发起针对"文章集合"资源--它可能位于/articles路径--的HTTP GET请求。要获取单篇文章，则必须提供它的标识符--Rails的做法是提供它的主键值(也就是id)。同样我们会发起一个GET请求，不过这次请求的 URL是/article/1。到目前为止，一切看起来都很熟悉。但如果要往集合中添加一篇文章又该怎么做呢？

　　在非REST化的应 用程序中，我们可能会开动脑筋发明一些动词短语来给action命名，例如articles/add\_article/1。而在REST的世界里，我们不 应该这样做，而应该用一组标准的动词来告诉资源应该怎么做。如果要以REST的[方式](http://www.google.com/url?q=http%3A%2F%2Fwww.verywin.cn&sa=D&sntz=1&usg=AFQjCNFnMyXdMvBD-laf_rdd0DS7sIXJaQ)往集合里添加一篇文章，我们应该 向/articles这个路径发起一个HTTP POST请求，同时发送与文章相关的数据。没错，这与我们获取文章列表所使用的路径相同：如果向它发起一个GET请求，它就会给出文章列表：如果发起 POST请求，它就会向集合中新增一篇文章。

　　更进一步。在前面我们已经看到，只要针对/article/1路径发起GET请求就可 以取回一篇文章的内容。如果要更新这篇文章，只要向同一个URL发起HTTP PUT请求即可。同样，如果要删除这篇文章，只需发起一个HTTP DELETE请求--还是同一个URL。

　　再进一步。也许我们的系统还需要记录用户信息。同样，这里会有一组资源供我们操作。 REST告诉我们，可以用同样的一组动词(GET、 POST、 PUT和DELETE)来操作一组类似的URL(/users、/user/1...)。