AWS
AICon
APICloud
PTC

全部话题
您目前处于： InfoQ首页 文章 理解本真的REST架构风格

理解本真的REST架构风格

喜欢 /作者 李锟 发布于 2013年8月26日. 估计阅读时间: 29 分钟 /QCon北京2018全面起航：开启与Netflix、微软、ThoughtWorks等公司的技术创新之路！ 30 讨论

本文是“深入探索REST”专栏系列深度内容中的第二篇，它将带您领略REST架构的起源、与Web的关系、REST架构的本质及特性，以及REST架构与其他架构风格之间的比较。

引子

在移动互联网、云计算迅猛发展的今天，作为一名Web开发者，如果您还没听说过“REST”这个buzzword，显然已经落伍了。夸张点说，甚至“出了门都不好意思跟别人打招呼”。尽管如此，对于REST这个泊来品的理解，大多数人（包括一些资深的架构师）仍然停留在“盲人摸象”的阶段。常常听到各种各样关于REST的说法，例如：有人说：“我们这套新的API决定不用Web Service（SOAP+WSDL），而是直接使用HTTP+JSON，也就是用RESTful的方式来开发。” 不用SOAP，甚至也不用XML，就自动变成了RESTful了。还有人认为：REST与传统的Web Service其实没有本质区别，只是对于URI的构造方式提出了更多要求，而这些要求Web Service完全都可以实现。潜台词是：既生瑜，何生亮。Web Service已经足够好了，干嘛还要再折腾什么REST。这些对于REST的不同说法，果真如此吗？REST究竟是什么？是一种新的技术、一种新的架构、还是一种新的规范？

对于这些问题笔者先不解答，为了深入理解REST是什么，我们需要回顾一下Web发展的最初年代，从源头上讲讲REST是怎么得来的。

Web技术发展与REST的由来

Web（万维网World Wide Web的简称）是个包罗万象的万花筒，不同的人从不同的角度观察，对于Web究竟是什么会得出大不相同的观点。作为Web开发者，我们需要从技术上来理解Web。从技术架构层面上看，Web的技术架构包括了四个基石：

- URI
- HTTP
- HyperText（除了HTML外，也可以是带有超链接的XML或JSON）
- MIME

这四个基石相互支撑，促使Web这座宏伟的大厦以几何级数的速度发展了起来。在这四个基石之上，Web开发技术的发展可以粗略划分成以下几个阶段：

1. 静态内容阶段：在这个最初的阶段，使用Web的主要是一些研究机构。Web由大量的静态HTML文档组成，其中大多是一些学术论文。Web服务器可以被看作是支持超文本的共享文件服务器。
2. CGI程序阶段：在这个阶段，Web服务器增加了一些编程API。通过这些API编写的应用程序，可以向客户端提供一些动态变化的内容。Web服务器与应用程序之间的通信，通过CGI（Common Gateway Interface）协议完成，应用程序被称作CGI程序。
3. 脚本语言阶段：在这个阶段，服务器端出现了ASP、PHP、JSP、ColdFusion等支持session的脚本语言技术，浏览器端出现了Java Applet、JavaScript等技术。使用这些技术，可以提供更加丰富的动态内容。
4. 瘦客户端应用阶段：在这个阶段，在服务器端出现了独立于Web服务器的应用服务器。同时出现了Web MVC开发模式，各种Web MVC开发框架逐渐流行，并且占据了统治地位。基于这些框架开发的Web应用，通常都是瘦客户端应用，因为它们是在服务器端生成全部的动态内容。
5. RIA应用阶段：在这个阶段，出现了多种RIA（Rich Internet Application）技术，大幅改善了Web应用的用户体验。应用最为广泛的RIA技术是DHTML+Ajax。Ajax技术支持在不刷新页面的情况下动态更新页面中的局部内容。同时诞生了大量的Web前端DHTML开发库，例如Prototype、Dojo、ExtJS、jQuery/jQuery UI等等，很多开发库都支持单页面应用（Single Page Application）的开发。其他的RIA技术还有Adobe公司的Flex、微软公司的Silverlight、Sun公司的JavaFX（现在为Oracle公司所有）等等。
6. 移动Web应用阶段：在这个阶段，出现了大量面向移动设备的Web应用开发技术。除了Android、iOS、Windows Phone等操作系统平台原生的开发技术之外，基于HTML5的开发技术也变得非常流行。

从上述Web开发技术的发展过程看，Web从最初其设计者所构思的主要支持静态文档的阶段，逐渐变得越来越动态化。Web应用的交互模式，变得越来越复杂：从静态文档发展到以内容为主的门户网站、电子商务网站、搜索引擎、社交网站，再到以娱乐为主的大型多人在线游戏、手机游戏。

在互联网行业，实践总是走在理论的前面。Web发展到了1995年，在CGI、ASP等技术出现之后，沿用了多年、主要面向静态文档的HTTP/1.0协议已经无法满足Web应用的开发需求，因此需要设计新版本的HTTP协议。在HTTP/1.0协议专家组之中，有一位年轻人脱颖而出，显示出了不凡的洞察力，后来他成为了HTTP/1.1协议专家组的负责人。这位年轻人就是Apache HTTP服务器的核心开发者Roy Fielding，他还是Apache软件基金会的合作创始人。

Roy Fielding和他的同事们在HTTP/1.1协议的设计工作中，对于Web之所以取得巨大成功，在技术架构方面的因素做了一番深入的总结。Fielding将这些总结纳入到了一套理论框架之中，然后使用这套理论框架中的指导原则，来指导HTTP/1.1协议的设计方向。HTTP/1.1协议的第一个草稿是在1996年1月发布的，经过了三年多时间的修订，于1999年6月成为了IETF的正式规范（包括了RFC 2616以及用于对客户端做身份认证的RFC 2617）。HTTP/1.1协议设计的极为成功，以至于发布之后整整10年时间里，都没有多少人认为有修订的必要。用来指导HTTP/1.1协议设计的这套理论框

架，最初是以备忘录的形式在专家组成员之间交流，除了IETF/W3C的专家圈子，并没有在外界广泛流传。Fielding在完成HTTP/1.1协议的设计工作之后，回到了加州大学欧文分校继续攻读自己的博士学位。第二年（2000年）在他的博士学位论文Architectural Styles and the Design of Network-based Software Architectures中，Fielding更为系统、严谨地阐述了这套理论框架，并且使用这套理论框架推导出了一种新的架构风格，并且为这种架构风格取了一个令人轻松愉快的名字“REST”——Representational State Transfer（表述性状态转移）的缩写。

在笔者看来，Fielding这篇博士论文在Web发展史上的价值，不亚于Web之父Tim Berners-Lee关于超文本的那篇经典论文。然而遗憾的是，这篇博士论文在诞生之后的将近5年时间里，一直没有得到足够的重视。例如Web Service相关规范SOAP/WSDL的设计者们，显然不大理解REST是什么，HTTP/1.1究竟是一个什么样的协议、为何要设计成这个样子。

这种情况在2005年之后有了很大的改善，随着Ajax、Ruby on Rails等新的Web开发技术的兴起，在Web开发技术社区掀起了一场重归Web架构设计本源的运动，REST架构风格得到了越来越多的关注。在2007年1月，支持REST开发的Ruby on Rails 1.2版正式发布，并且将支持REST开发作为Rails未来发展中的优先内容。Ruby on Rails的创始人DHH做了一个名为“World of Resources”的精彩演讲，DHH在Web开发技术社区中的强大影响力，使得REST一下子处在Web开发技术舞台的聚光灯之下。

今天，各种流行的Web开发框架，几乎没有不支持REST开发的了。大多数Web开发者都是通过阅读某种REST开发框架的文档，以及通过一些例子代码来学习REST开发的。然而，通过例子代码来学习REST有非常大的局限性。因为REST并不是一种具体的技术，也不是一种具体的规范，REST其实是一种内涵非常丰富的架构风格。通过例子代码来学习REST，除了学习到一种有趣的Web开发技术之外，并不能全面深入的理解REST究竟是什么。甚至还会误以为这些简单的例子代码就是REST本身，REST不过是一种简单的Web开发技术而已。就像盲人摸象一样，有的人摸到了象鼻子、有的人摸到了象耳朵、有的人摸到了象腿、有的人摸到了象尾巴。他们都坚信自己感觉到的大象，才是最真实的大象，而其他人的感觉都是错误的。

对于不理解REST的Web开发者，人们习惯于展示一些例子代码来让他们理解REST，笔者不赞同上述做法。如果Web开发者想要深入理解REST是什么，就很难避开Fielding的这篇博士论文。笔者在本文中对于REST是什么的介绍，也是基于Fielding的博士论文的。尽管如此，笔者强烈建议本文的读者亲自去通读一下Fielding的博士论文，就像想要了解孔子的思想应该直接去读《论语》等著作，而不是首先去读其他人的转述一样。笔者在本文中也仅仅是努力不做一个把经书念错了的歪嘴和尚而已。那么，下面我们言归正传。

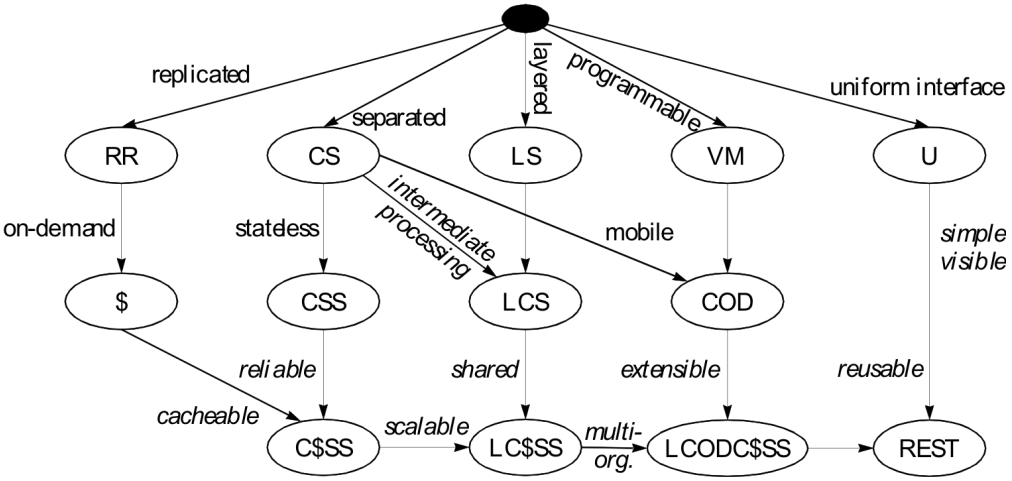
在Fielding的这篇名为Architectural Styles and the Design of Network-based Software Architectures的博士论文（中文版名为《架构风格与基于网络的软件架构设计》）中，提出了一整套基于网络的软件（即所谓的“分布式应用”）的设计方法，值得所有分布式应用的开发者仔细阅读、深入体会。

在论文的前三章中，Fielding在批判性继承前人研究成果的基础上，建立起来一整套研究和评价软件架构的方法论。这套方法论的核心是“架构风格”这个概念。架构风格是一种研究和评价软件架构设计的方法，它是比架构更加抽象的概念。一种架构风格是由一组相互协作的架构约束来定义的。架构约束是指软件的运行环境施加在架构设计之上的约束。

在论文的第四章中，Fielding研究了Web这样一个分布式系统对于软件架构设计提出了哪些需求。在第五章中，Fielding将第四章Web提出的需求具体化为一些架构约束，通过逐步添加各种架构约束，推导出来了REST这种新的架构风格。

REST架构风格的推导过程如下图所示：

图1：REST所继承的架构风格约束（原图可在这里下载）



在图1中，每一个椭圆形里面的缩写词代表了一种架构风格，而每一个箭头边的单词代表了一种架构约束。

REST架构风格最重要的架构约束有6个：

- 客户-服务器（Client-Server）

通信只能由客户端单方面发起，表现为请求-响应的形式。

- 无状态（Stateless）

通信的会话状态（Session State）应该全部由客户端负责维护。

- 缓存（Cache）

响应内容可以在通信链的某处被缓存，以改善网络效率。

- 统一接口（Uniform Interface）

通信链的组件之间通过统一的接口相互通信，以提高交互的可见性。

- 分层系统（Layered System）

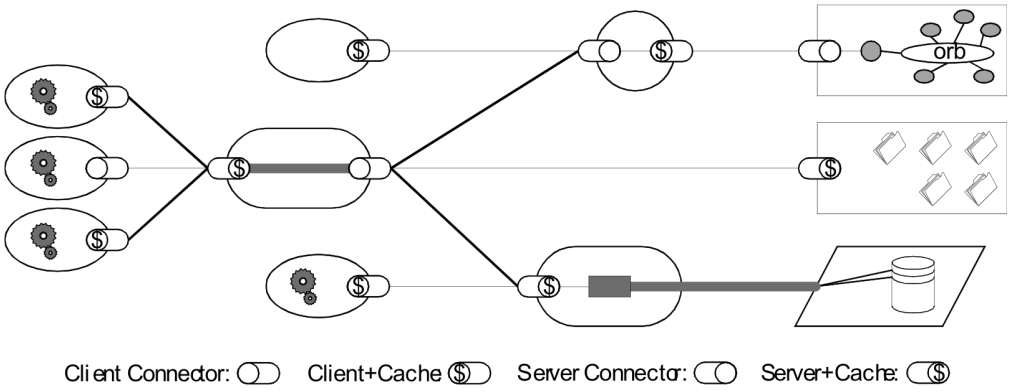
通过限制组件的行为（即，每个组件只能“看到”与其交互的紧邻层），将架构分解为若干等级的层。

- 按需代码（Code-On-Demand，可选）

支持通过下载并执行一些代码（例如Java Applet、Flash或JavaScript），对客户端的功能进行扩展。

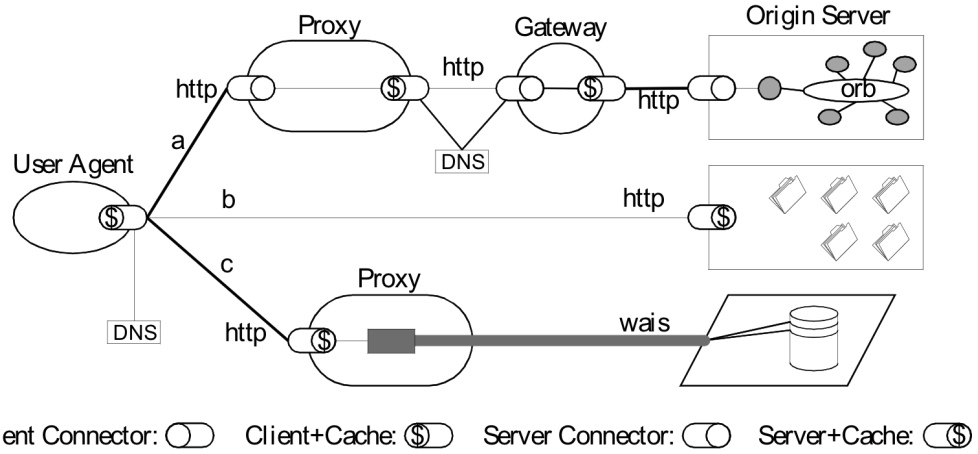
在论文中推导出的REST架构风格如下图所示：

图2：REST架构风格（原图可在这里下载）



而HTTP/1.1协议作为一种REST架构风格的架构实例，其架构如下图所示：

图3：一个基于REST的架构的过程视图（原图可在这里下载）



用户代理处在三个并行交互（a、b和c）的中间。用户代理的客户端连接器缓存无法满足请求，因此它根据每个资源标识符的属性和客户端连接器的配置，将每个请求路由到资源的来源。请求（a）被发送到一个本地代理，代理随后访问一个通过DNS查找发现的缓存网关，该网关将这个请求转发到一个能够满足该请求的来源服务器，服务器的内部资源由一个封装过的对象请求代理（object request broker）架构来定义。请求（b）直接发送到一个来源服务器，它能够通过自己的缓存来满足这个请求。请求（c）被发送到一个代理，它能够直接访问WAIS（一种与Web架构分离的信息服务），并将WAIS的响应翻译为一种通用的连接器接口能够识别的格式。每一个组件只知道与它们自己的客户端或服务器连接器的交互；整个过程拓扑是我们的视图的产物。

通过比较图2和图3，读者不难发现这两张图中的架构是高度一致的。对于HTTP/1.1协议为何要设计成这个样子，读者想必已经有所领悟。

在论文的第六章中，Fielding对于到2000年为止在Web基础架构协议的设计和开发方面的一些经验教训进行了深入的分析。其中，“HTTP不是RPC”、“HTTP不是一种传输协议”两部分值得读者反复阅读。时至13年之后的今日，对于HTTP协议的误解仍然广泛存在。

以上简要介绍了Fielding博士论文中的内容。为了帮助读者仔细阅读Fielding的博士论文，笔者整理了一套Fielding博士论文的导读，将在本专栏后续文章中载出。

REST详解

REST究竟是什么？因为REST的内涵非常丰富，所以很难用一两句话解释清楚这个问题。

首先，REST是Web自身的架构风格。REST也是Web之所以取得成功的技术架构方面因素的总结。REST是世界上最成功的分布式应用架构风格（成功案例：Web，还不够吗？）。它是为 运行在互联

网环境 的 分布式 超媒体系统量身定制的。互联网环境与企业内网环境有非常大的差别，最主要的差别是两个方面：

- 可伸缩性需求无法控制：并发访问量可能会暴涨，也可能会暴跌。
- 安全性需求无法控制：无法控制客户端发来的请求的格式，很可能会是恶意的请求。

而所谓的“超媒体系统”，即，使用了超文本的系统。可以把“超媒体”理解为超文本+媒体内容。

REST是HTTP/1.1协议等Web规范的设计指导原则，HTTP/1.1协议正是为实现REST风格的架构而设计的。新的Web规范，其设计必须符合REST的要求，否则整个Web的体系架构会因为引入严重矛盾而崩溃。这句话不是危言耸听，做个类比，假如苏州市政府同意在市区著名园林的附近大型土木，建造大量具有后现代风格的摩天大楼，那么不久之后世界闻名的苏州园林美景将不复存在。

上述这些关于“REST是什么”的描述，可以总结为一句话：REST是所有Web应用都应该遵守的架构设计指导原则。当然，REST并不是法律，违反了REST的指导原则，仍然能够实现应用的功能。但是违反了REST的指导原则，会付出很多代价，特别是对于大流量的网站而言。

要深入理解REST，需要理解REST的五个关键词：

1. 资源（Resource）
2. 资源的表述（Representation）
3. 状态转移（State Transfer）
4. 统一接口（Uniform Interface）
5. 超文本驱动（Hypertext Driven）

什么是资源?

资源是一种看待服务器的方式，即，将服务器看作是由很多离散的资源组成。每个资源是服务器上一个可命名的抽象概念。因为资源是一个抽象的概念，所以它不仅仅能代表服务器文件系统中的一个文件、数据库中的一张表等等具体的东西，可以将资源设计的要多抽象有多抽象，只要想象力允许而且客户端应用开发者能够理解。与面向对象设计类似，资源是以名词为核心来组织的，首先关注的是名词。一个资源可以由一个或多个URI来标识。URI既是资源的名称，也是资源在Web上的地址。对某个资源感兴趣的客户端应用，可以通过资源的URI与其进行交互。

什么是资源的表述?

资源的表述是一段对于资源在某个特定时刻的状态的描述。可以在客户端-服务器端之间转移（交换）。资源的表述可以有多种格式，例如HTML/XML/JSON/纯文本/图片/视频/音频等等。资源的表述格式可以通过协商机制来确定。请求-响应方向的表述通常使用不同的格式。

什么是状态转移?

状态转移（state transfer）与状态机中的状态迁移（state transition）的含义是不同的。状态转移说的是：在客户端和服务器端之间转移（transfer）代表资源状态的表述。通过转移和操作资源的表述，来间接实现操作资源的目的。

什么是统一接口?

REST要求，必须通过统一的接口来对资源执行各种操作。对于每个资源只能执行一组有限的操作。以HTTP/1.1协议为例，HTTP/1.1协议定义了一个操作资源的统一接口，主要包括以下内容：

- 7个HTTP方法：GET/POST/PUT/DELETE/PATCH/HEAD/OPTIONS

- HTTP头信息（可自定义）
- HTTP响应状态代码（可自定义）
- 一套标准的内容协商机制
- 一套标准的缓存机制
- 一套标准的客户端身份认证机制

REST还要求，对于资源执行的操作，其操作语义必须由HTTP消息体之前的部分完全表达，不能将操作语义封装在HTTP消息体内部。这样做是为了提高交互的可见性，以便于通信链的中间组件实现缓存、安全审计等等功能。

什么是超文本驱动?

“超文本驱动”又名“将超媒体作为应用状态的引擎”（Hypermedia As The Engine Of Application State，来自Fielding博士论文中的一句话，缩写为HATEOAS）。将Web应用看作是一个由很多状态（应用状态）组成的有限状态机。资源之间通过超链接相互关联，超链接既代表资源之间的关系，也代表可执行的状态迁移。在超媒体之中不仅仅包含数据，还包含了状态迁移的语义。以超媒体作为引擎，驱动Web应用的状态迁移。通过超媒体暴露出服务器所提供的资源，服务器提供了哪些资源是在运行时通过解析超媒体发现的，而不是事先定义的。从面向服务的角度看，超媒体定义了服务器所提供服务的协议。客户端应该依赖的是超媒体的状态迁移语义，而不应该对于是否存在某个URI或URI的某种特殊构造方式作出假设。一切都有可能变化，只有超媒体的状态迁移语义能够长期保持稳定。

一旦读者理解了上述REST的五个关键词，就很容易理解REST风格的架构所具有的6个的主要特征：

- 面向资源（Resource Oriented）
- 可寻址（Addressability）
- 连通性（Connectedness）
- 无状态（Statelessness）
- 统一接口（Uniform Interface）
- 超文本驱动（Hypertext Driven）

这6个特征是REST架构设计优秀程度的判断标准。其中，面向资源是REST最明显的特征，即，REST架构设计是以资源抽象为核心展开的。可寻址说的是：每一个资源在Web之上都有自己的地址。连通性说的是：应该尽量避免设计孤立的资源，除了设计资源本身，还需要设计资源之间的关联关系，并且通过超链接将资源关联起来。无状态、统一接口是REST的两种架构约束，超文本驱动是REST的一个关键词，在前面都已经解释过，就不再赘述了。

从架构风格的抽象高度来看，常见的分布式应用架构风格有三种：

- 分布式对象（Distributed Objects，简称DO）

架构实例有CORBA/RMI/EJB/DCOM/.NET Remoting等等

- 远程过程调用（Remote Procedure Call，简称RPC）

架构实例有SOAP/XML-RPC/Hessian/Flash AMF/DWR等等

- 表述性状态转移（Representational State Transfer，简称REST）

架构实例有HTTP/WebDAV

DO和RPC这两种架构风格在企业应用中非常普遍，而REST则是Web应用的架构风格，它们之间有非常大的差别。

REST与DO的差别在于：

- REST支持抽象（即建模）的工具是资源，DO支持抽象的工具是对象。在不同的编程语言中，对象的定义有很大差别，所以DO风格的架构通常都是与某种编程语言绑定的。跨语言交互即使能实现，实现起来也会非常复杂。而REST中的资源，则完全中立于开发平台和编程语言，可以使用任何编程语言来实现。
- DO中没有统一接口的概念。不同的API，接口设计风格可以完全不同。DO也不支持操作语义对于中间组件的可见性。
- DO中没有使用超文本，响应的内容中只包含对象本身。REST使用了超文本，可以实现更大粒度的交互，交互的效率比DO更高。
- REST支持数据流和管道，DO不支持数据流和管道。
- DO风格通常会带来客户端与服务器端的紧耦合。在三种架构风格之中，DO风格的耦合度是最大的，而REST的风格耦合度是最小的。REST松耦合的源泉来自于统一接口+超文本驱动。

REST与RPC的差别在于：

- REST支持抽象的工具是资源，RPC支持抽象的工具是过程。REST风格的架构建模是以名词为核心的，RPC风格的架构建模是以动词为核心的。简单类比一下，REST是面向对象编程，RPC则是面向过程编程。
- RPC中没有统一接口的概念。不同的API，接口设计风格可以完全不同。RPC也不支持操作语义对于中间组件的可见性。
- RPC中没有使用超文本，响应的内容中只包含消息本身。REST使用了超文本，可以实现更大粒度的交互，交互的效率比RPC更高。
- REST支持数据流和管道，RPC不支持数据流和管道。
- 因为使用了平台中立的消息，RPC风格的耦合度比DO风格要小一些，但是RPC风格也常常会带来客户端与服务器端的紧耦合。支持统一接口+超文本驱动的REST风格，可以达到最小的耦合度。

比较了三种架构风格之间的差别之后，从面向实用的角度来看，REST架构风格可以为Web开发者带来三方面的利益：

- 简单性

采用REST架构风格，对于开发、测试、运维人员来说，都会更简单。可以充分利用大量HTTP服务器端和客户端开发库、Web功能测试/性能测试工具、HTTP缓存、HTTP代理服务器、防火墙。这些开发库和基础设施早已成为了日常用品，不需要什么火箭科技（例如神奇昂贵的应用服务器、中间件）就能解决大多数可伸缩性方面的问题。

- 可伸缩性

充分利用好通信链各个位置的HTTP缓存组件，可以带来更好的可伸缩性。其实很多时候，在Web前端做性能优化，产生的效果不亚于仅仅在服务器端做性能优化，但是HTTP协议层面的缓存常常被一些资深的架构师完全忽略掉。

- 松耦合

统一接口+超文本驱动，带来了最大限度的松耦合。允许服务器端和客户端程序在很大范围内，相对独立地进化。对于设计面向企业内网的API来说，松耦合并不是一个很重要的设计关注点。但是对于设计面向互联网的API来说，松耦合变成了一个必选项，不仅在设计时应该关注，而且应该放在最优先位置。

有的读者可能会问：“你说了这么多，REST难道就没有任何缺点了吗？”当然不是，正如Fielding在博士论文中阐述的那样，评价一种软件架构的优劣，不能脱离开软件的具体运行环境。永远不存在适用于任何运行环境的、包治百病的银弹式架构。笔者在前面强调过REST是一种为运行在互联网环境中的Web应用量身定制的架构风格。REST在互联网这个运行环境之中已经占据了统治地位，然而，在企业内网运行环境之中，REST还会面临DO、RPC的巨大挑战。特别是一些对实时性要求很高的应用，REST的表现不如DO和RPC。所以需要针对具体的运行环境来具体问题具体分析。但是，REST可以带来的上述三方面的利益即使在开发企业应用时，仍然是非常有价值的。所以REST在企业应用开发，特别是在SOA架构的开发中，已经得到了越来越大的重视。本专栏将有一篇文章专门介绍REST在企业级应用中与SOA的结合。

到了这里，“REST究竟是什么”这个问题笔者就解答完了。本文开头那些说法是否正确，笔者还是笑而不语，读者此时应该已经有了自己的判断。在接下来的REST系列文章中，我将会为读者澄清一些关于HTTP协议和REST的常见误解。

参考资料：

Roy Fielding博士论文英文版

Roy Fielding博士论文中文版

HTTP/1.1协议RFC2616、RFC2617

感谢马国耀对本文的策划和审校。

关注IT趋势，承载前沿、深入、有温度的内容。感兴趣的读者可以搜索ID：laocuixiabian，或者扫描下方二维码加关注。

架构 & 设计 深入探索REST 架构 Web API Web服务 企业架构 API SOA REST

相关主题:

告诉我们您的想法