

## 一、Web Services

Web Services 是一种基于组件的软件平台，是面向服务的Internet 应用。Web Services 框架的核心技术包括SOAP ,WSDL 和UDDI ,它们都是以标准的XML 文档的形式表示。

SOAP ( “Simple Object Access Protocol” 的缩写)是Web Services 的通信协议。SOAP是一种简单的、轻量级的基于XML 的机制，用于在网络应用程序之间进行结构化数据交换。SOAP包括三部分:一个定义描述消息内容的框架的信封，一组表示应用程序定义的数据类型实例的编码 规则，以及表示远程过程调用和响应的约定。

1. 传递信息的格式为XML.这就使Web Services能够在任何平台上，用任何语言进行实现。
2. 远程对象方法调用的格式。规定了怎样表示被调用对象以及调用的方法名称和参数类型等。
3. 参数类型和XML格式之间的映射。这是因为，被调用的方法有时候需要传递一个复杂的参数，例如，一个Person对象。怎样用XML来表示一个对象参数，也是SOAP所定义的范围。

WSDL表示WEB服务说明语言。WSDL文件是一个XML 文档，用于说明一组SOAP消息以及如何交换这些消息。当实现了某种服务的时候(如：股票查询服务)，为了让别的程序调用，必须告诉大家服务接口。例如：服务名称，服务所在的机器名称，监听端口号，传递参数的类型，个数和顺序，返回结果的类型等等。这样别的应用程序才能调用该服务。WSDL协议就是规定了 有关Web Services描述的标准。

UDDI( “Universal Description, Discovery, and Integration” 的缩写)提供一种发布和查找服务描述的方法。UDDI 数据实体提供对定义业务和服务信息的支持。WSDL 中定义的服务描述信息是UDDI注册中心信息的补充。

**Web Services的工作原理如下：**

1. Web Services 服务提供方通过WSDL描述所提供的服务，并将这一描述告知Web Services 注册服务器。
2. 注册服务器依据WSDL 的描述，依照UDDI的协定更新服务目录并在Internet 上发布。
3. 用户在使用Web Services 前先向注册服务器发出请求，获得Web Services 提供者的地址和服务接口信息。
4. 使用SOAP 协议与Web Services 提供者建立连接,进行通信。

## 二、REST(Representational State Transfer) Web Services

通常我们提到Web Services第一想法就是SOAP消息在各种传输协议上交互。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，但是随着 SOAP作为Web Services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。在大并发情况下会有性能问题，在互联网上使用不太 普及，因此并不太适合Web 2.0网站服务使用，目前大量的Web 2.0网站使用另外一种解决方案——REST。

作为SOAP模式 的替代者，REST(是 “Representational State Transfer” 的缩写)是一种轻量级的Web Services架构风格，其实现和操作明显比SOAP和XML-RPC更为简洁，可以完全通过HTTP协议实现，还可以利用缓存Cache来提高响应速度，性能、效率和易用性上都优于SOAP协议。

### 1. RESTful Web Services是什么



REST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。在RESTful系统中，服务器利用URI暴露资源，客户端使用四个Http谓词来访问资源。由于客户端接收了资源，他们被置于某种状态。当他们访问一个新的资源，通常是点击下一个连接，他们改变了，或者说是过渡了他们的状态。为了工作，REST假设资源是能够使用普遍的标准语法来代表的。

传统的Web应用大都是B/S架构，它包括了如下一些规范。

**1.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。

**2.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。它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。

**3.缓存：**为了改善无状态性带来的网络的低效性，我们添加了缓存约束。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。

**1.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。

**2.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。

**3.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描述了一种设计Web应用的架构风格，它是一组架构约束条件和原则，满足这些约束条件和原则的应用程序或设计就是 RESTful风格的。而符合RESTful风格的Web Services，就是我们所说的RESTful Web Services。

## 2. REST原则如下：

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位会员”在数据上可能有重叠或者完全相同，而由于它们的表现形式不同，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。它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，这为开发人员引入了一种新的思维方式：通过URL来设计系统结构。当然，这种设计方式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，也减轻了服务器端的性能开销。

REST提出了如下设计准则

- **资源由URI来指定**：在Web应用中，所有的事物都应该拥有唯一的ID，代表ID的统一概念是：URI。URI构成了一个全局命名空间，使用URI标识你的关键资源意味着它们获得了一个唯一、全局的ID。
- **显式的使用HTTP方法**：REST要求开发人员显式地使用HTTP方法，并且使用方式与协议定义一致。这个基本REST设计原则建立了创建、读取、更新和删除(create, read, update, and delete, CRUD)操作与HTTP方法之间的一对一映射。根据此映射：
  - 若要在服务器上创建资源，应该使用POST方法。
  - 若要检索某个资源，应该使用GET方法。
  - 若要更改资源状态或对其进行更新，应该使用PUT方法。
  - 若要删除某个资源，应该使用DELETE方法。

**CRUD原则**：对于资源只需要四种行为：*Create(创建)*、*Read(读取)*、*Update(更新)*和*Delete(删除)*就可以完成对其操作和处理

**除了抽象操作为基础的CRUD**。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

- **资源多重表述**：针对不同的需求提供资源多重表述。这里所说的多重表述包括XML、JSON、HTML等。即服务器端需要向外部提供多种格式的资源表述，供不同的客户端使用。比如移动应用可以使用XML或JSON和服务器端通信，而浏览器则能够理解HTML。
- **无状态**：对服务器端的请求应该是无状态的，完整、独立的请求不要求服务器在处理请求时检索任何类型的应用程序上下文或状态。无状态约束使服务器的变化对客户端是不可见的，因为在两次连续的请求中，



客户端并不依赖于同一台服务器。一个客户端从某台服务器上收到一份包含链接的文档，当它要做一些处理时，这台服务器宕掉了，可能是硬盘坏掉而被拿去修理，可能是软件需要升级重启——如果这个客户端访问了从这台服务器接收的链接，它不会察觉到后台的服务器已经改变了。

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。

举个例子吧，HTTP GET 请求中的请求 URI 中的查询字符串包括一组参数，这些参数定义服务器用于查找一组匹配资源的搜索条件。但是在许多情况下，不优雅的 Web API 使用 HTTP GET 来触发服务器上的事务性操作——例如，向数据库添加记录。如：

```
GET /adduser?name=John HTTP/1.1
```

这不是非常好的设计，因为上面的 Web 方法支持通过 HTTP GET 进行状态更改操作。Web 服务器旨在通过检索与请求 URI 中的查询条件匹配的资源，并在响应中返回这些资源或其表示形式，从而响应 HTTP GET 请求，而不是向数据库添加记录。以这种方式使用 GET 是不一致的。

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：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，它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做为核心理念的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。



### 三、RESTful Web Services与基于SOAP的Web Services的比较

基于SOAP的Web Services也是解决异构系统间通信问题的常用方案，那么，RESTful Web Services相对于基于SOAP 的Web Services，有什么优势呢？或者说，我们为什么要开始学习RESTful Web Services，使用已经流行很久的基于SOAP的Web Services不就好了么？

- *RESTful Web Services接口更易于使用*

RESTful Web Services使用标准的 HTTP 方法 (GET/PUT/POST/DELETE) 来抽象所有 Web 系统的服务能力，而不同的是，SOAP 应用都通过定义自己个性化的接口方法来抽象 Web 服务。相对来说，RESTful Web Services接口更简单。

RESTful Web Services使用标准的 HTTP 方法的优势，从大的方面来讲：标准化的 HTTP 操作方法，结合其他的标准化技术，如 URI，HTML，XML 等，将会极大提高系统与系统之间整合的互操作能力。尤其在 Web 应用领域，RESTful Web Services所表达的这种抽象能力更加贴近 Web 本身的工作方式，也更加自然。

- 无状态性

HTTP 协议从本质上说是一种无状态的协议，客户端发出的 HTTP 请求之间可以相互隔离，不存在相互的状态依赖。基于 HTTP 的 ROA，以非常自然的方式来实现无状态服务请求处理逻辑。对于分布式的应用而言，任意给定的两个服务请求 Request 1 与 Request 2，由于它们之间并没有相互之间的状态依赖，就不需要对它们进行相互协作处理，其结果是：Request 1 与 Request 2 可以在任何的服务器上执行，这样的应用很容易在服务器端支持负载均衡 (load-balance)。

- 安全操作与幂指相等特性

HTTP 的 GET、HEAD 请求本质上应该是安全的调用，即：GET、HEAD 调用不会有任何的副作用，不会造成服务器端状态的改变。对于服务器来说，客户端对某一 URI 做 n 次的 GET、HEAD 调用，其状态与没有做调用是一样的，不会发生任何的改变。

HTTP 的 PUT、DELETE 调用，具有幂指相等特性，即：客户端对某一 URI 做 n 次的 PUT、DELETE 调用，其效果与做一次的调用是一样的。HTTP 的 GET、HEAD 方法也具有幂指相等特性。

HTTP 这些标准方法在原则上保证你的分布式系统具有这些特性，以帮助构建更加健壮的分布式系统。

- *RESTful Web Services更容易实现缓存*

众所周知，对于基于网络的分布式应用，网络传输是一个影响应用性能的重要因素。如何使用缓存来节省网络传输带来的开销，这是每一个构建分布式网络应用的开发人员必须考虑的问题。

HTTP 协议带条件的 HTTP GET 请求 (Conditional GET) 被设计用来节省客户端与服务器之间网络传输带来的开销，这也给客户端实现 Cache 机制（包括在客户端与服务器之间的任何代理）提供了可能。HTTP 协议通过 HTTP HEADER 域：If-Modified-Since/Last-Modified，If-None-Match/ETag 实现带条件的 GET 请求。

REST 的应用可以充分地挖掘 HTTP 协议对缓存支持的能力。当客户端第一次发送 HTTP GET 请求给服务器获得内容后，该内容可能被缓存服务器 (Cache Server) 缓存。当下一次客户端请求同样的资源时，缓存可以直接给出响应，而不需要请求远程的服务器获得。而这一切对客户端来说都是透明的。



- 一些缺陷：

先说成熟度，SOAP发展到现在虽然已经背离了初衷，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 Services都能够较好的互通。

反观REST，相比于SOAP的权威性协议规范，REST实现的各种协议只能算是私有协议，当然需要遵循REST的思想，在兼容性方面会差很多。

总的来说SOAP在成熟度上优于REST。

再说效率，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。REST被人们的重视，其实很大原因是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，REST还很好的融合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，等形式。

因此，相对于SOAP，REST的效率更胜一筹。

最后说安全性，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。REST没有任何规范对于安全方面作说明。

#### 四、案例：为Web项目构建一个简单的JSON控制器

在日常应用中，我们有大量的场合可以使用到RESTful Web Services，包括Web系统间的交互，移动客户端与Web服务器端的通信等。只有在日常工作中更多的实践RESTful，才能更好的理解RESTful Web Services。

无论项目使用的是哪种数据库后端，JavaScript Object Notation (JSON) 控制器都能简化开发工作。

假设一个PHP/MySQL项目：建立一个MySQL数据库，创建包含HTML的PHP视图，根据需要添加JavaScript代码和CSS文件，连接到数据库，从数据库提取内容来填充视图，等等。如果您熟悉web开发，您一定知道分隔功能代码的好处。例如，您知道要避免直接在视图中输入原始SQL查询，不会在从数据库提取数据的函数或类中混淆HTML标记。

但是，有时，您的项目可能扩展到您的正常PHP/MySQL舒适水平之外。例如，您可能不仅拥有需要来自一个数据库的数据的常规web视图，还拥有外部应用程序（比如Facebook），甚至还拥有访问相同数据的移动设备（比如智能手机）。

您可能会发现自己身陷这样一种情况：数据库更改，或者要求您处理某种类型的XML存储库。在这些情况下，您对MySQL的盲目依赖可能会阻碍您完成项目的工作。

此时可以考虑将一个RESTful JSON控制器放置到项目中，将它用作一个虚拟交通警察，负责发送请求并接收来自您的数据源的响应。下面将并展示一种建立控制器的方法。其结果是从一个数据源检索数据的简单方法，检索的数据采用标准化的格式，可以使用PHP或JavaScript代码轻松解析。

- 在一个典型的REST架构中，一个客户机发送一个请求到服务器，服务器使用请求资源的一个表示来进行响应。资源可以是任何信息对象，比如数据库或文档，它的表示通常是一个格式化的文档（通常是XML或JSON），充当它的当前或被请求状态的一个快照。

REST资源通常使用有类型的URI来访问，这些URI使用不同的请求动词GET、POST、PUT和



- REST 资源通常使用有意义的 URLs 标识，这些 URLs 接受不同的请求动词 GET、POST、PUT 和 DELETE。这些动词有点类似于许多开发人员都熟悉的 create-retrieve-update-delete (CRUD) 模型。
- 例如，如果您想检索数据，则使用 GET 请求；要创建数据，则使用 POST 请求；要更新数据，则使用 PUT 请求；最后，要删除数据，则使用 DELETE 请求。
- 另一个需要考虑的重要因素是响应。RESTful 服务通常在它的响应中提供两个有意义的组件：响应主体本身和一个状态码。许多 REST 服务实际上允许用户指定一个响应格式（比如 XML、CSV、序列化的 PHP 对象或纯文本），方法有两种：一是发送一个 ACCEPT 参数；二是指定一个文件扩展名（例如，/api/users.xml 或 /api/users.json）。其他 REST 服务器，比如您将在这里实现的服务器，拥有硬编码的响应格式。这些格式同样可以接受，只要它们已经有文档记载。
- 响应代码往往是 HTTP 状态码。这种模式的优点是可以使用知名的现有状态码来标识错误或成功。状态码 201 (CREATED) 是一个成功 POST 请求的完美响应。错误码 500 表明在您所处的这端（服务端）上发生了错误，但错误码 400 表明客户端上出现了失败 (BAD REQUEST)。如果服务器出现故障，将发送错误码 503 (SERVICE UNAVAILABLE)。

一个应用程序拥有的一个数据源包含一些用户信息，名、姓、邮件地址、以及 Twitter 帐户。如果您正在设置一个典型的 PHP 应用程序，您需要创建一个 mysql\_query() 包装器来使用一个 SQL 查询从数据库提取一个清单。您还需要编写一些 PHP 代码，用于调用那个函数并循环结果集，以便在应用程序视图中显示数据。

设置一个简单的 REST 控制器，该控制器允许一个针对 /users/list 的、不带任何参数的 GET 请求，然后调用适当的数据库函数并返回一个 JSON 格式的清单。接下来，您的应用程序可以解码那个 JSON 数据，以任何必要的方式循环该数据，以便显示数据内容。

另外，您可以通过测试检查是否有任何参数被发送到 /users/list。例如，如果您发送一个 GET 请求到 /users/list/1，那么响应将只包含 ID 为 1 的用户的细节。除 JSON 格式外，您甚至可以允许其他格式，比如 XML、CSV 和的 PHP 对象。

**一个 RESTful JSON 控制器对于您的开发工作的作用并非仅仅是在视图和数据源之间放置一个额外的功能层。**想想看，您的基本 PHP 视图也许不是请求信息的惟一组件。例如，您可能会使用 jQuery 通过一个 Ajax 接口请求数据，或者，您的用户可能会通过一部智能手机或一个 Facebook 应用程序请求数据。

在这些情况下，一个接收请求并以一种容易理解（和预测）的格式提供响应的 RESTful 接口可能会极大地简化您的开发工作。作为负责 PHP 视图（或者甚至 iPhone 应用程序）的开发人员，您可以发送一些请求到一个 URL 并接收一组预期响应。在 JSON 控制器的另一面，应用程序可以被钩挂（hook）到 MySQL、PostgreSQL、一个 XML 文件存储库、或者什么也不挂。

## 1. 首先创建一个简单的事件数据库架构 (MYSQL)

```
CREATE TABLE `events` (
  `id` INT NOT NULL AUTO_INCREMENT PRIMARYKEY ,
  `title` VARCHAR( 255 ) NOT NULL ,
  `address` VARCHAR( 255 ) NOT NULL ,
  `start_time` DATETIME NOT NULL ,
  `description` TEXT NOT NULL
);
```

2. 创建一个典型 PHP 模型文件，它连接到这个数据库并使用一个 SQL 查询来识别事件。

 View Code

对这个函数的一个简单调用时，您将得到如下所示的结果。

```

$EVENT = new Events;
$today = '2010-06-17';
$events = $EVENT->get_events($today);
print_r($events);
/* results in
Array
    ([0] => Array(
        [id] => 2
        [title] => Event #2
        [address] => 156 My Avenue, MyTown, USA 78727
        [start_time] => 2010-06-17 11:30:00
        [description] => Join us for lunch to hear
        FABULOUS SPEAKER.
    )
    [1] => Array(
        [id] => 1
        [title] => Event #1
        [address] => 123 My Street, Anytown USA 78727
        [start_time] => 2010-06-17 15:30:00
        [description] => A great event! Hope to see you there!
    )
)
*/
```

通过 `json_encode()` 运行相同的代码，将得到一个可移植的 JSON 对象（如 所示）

```

[
    {
        "id": "2",
        "title": "Event #2",
        "address": "156 My Avenue, MyTown, USA 78727",
        "start_time": "2010-06-17 11:30:00",
        "description": "Join us for lunch to hear FABULOUS SPEAKER. "
    },
    {
        "id": "1",
        "title": "Event #1",
        "address": "123 My Street, Anytown USA 78727",
        "start_time": "2010-06-17 15:30:00",
        "description": "A great event! Hope to see you there!"
    }
]
```



```
description : A great event! hope to see you there!
    }
}
```

目标是构建这样一个简单的控制器：它知道应该运行哪个模型和函数，然后返回一个 JSON 对象作为响应，这个响应可用于事务的远端。这个控制器非常简单，看起来如下所示。将如下所有代码粘贴到一个名为 json.php 的文件中。

```
class JSON{
    var $response = '';
    function JSON($model,$function,$params){
        $REQUEST = new $model;
        $data = $REQUEST->$function($params);
        $this->response = json_encode($data);
    }
}
```

调用的模型，实例化 JSON 类，然后传入 3 个参数：模型的类名、要运行的函数、以及该函数的参数。这个类然后调用那个函数并获取一个响应，该响应通过 json\_encode() 运行。

3. 最后一步是创建包含对 JSON 数据的请求的文件。这个特殊的文件（您可以称之为 listing.php）可以设置为接收 3 个 GET 变量（模型、函数和参数各一个），然后将这些变量传递给 JSON 类（如清单 7 所示）

请求代码

```
//this is the code that contains the model
require 'events.php';
//this is the JSON controller
require 'json.php';
//pass in your three GET parameters
$MODEL = $_GET['model'];
$FUNCTION = $_GET['function'];
//check to see if param is passed in
//if not, use today's date in this instance
if (isset($_GET['param'])){
    $PARAM = $_GET['param'];
}else{
    $PARAM = date("Y-m-d");
}
//invoke
$JSON = new JSON($MODEL,$FUNCTION,$PARAM);
//access the response variable
echo $JSON->response;
```




可以将这个文件加载到一个浏览器中，并获取一个 JSON 对象。再通过 `json_decode()` 将这个 JSON 对象发送回去，使用 JavaScript 代码处理它，或者让它保持原样。

整个这个流程的一个甚至更好的方法是创建一个更紧密模拟 RESTful 服务器的路径结构。例如，可以创建一个名为 `events/today` 的目录结构，该结构包含一个名为 `index.php` 的文件。通过将您的浏览器指向 `/events/today`，无需传入任何 GET 变量，您就可以基于如下的代码取回一个 JSON feed。

`/events/today/index.php` 中的代码



```
require '../events.php';
require '../json.php';
$MODEL = "Events";
$FUNCTION = "get_events";
$PARAM = date("Y-m-d");
//invoke
$JSON = new JSON($MODEL, $FUNCTION, $PARAM);
echo $JSON->response;
//prints out
[
    {
        "id": "3",
        "title": "Test Event 3",
        "address": "111 Main Street, Austin TX 78727",
        "start_time": "2010-06-10 15:15:00",
        "description": "Testing 456."
    }
]
```



使用这种方法，您可以为您的视图和支持的应用程序简化一些数据提取要求。开发人员无需记住底层数据库的所有细节，相反，他们可以轻松命中 URLs 并接收他们寻找的响应来继续他们的工作。

### 参考:

- <http://tech.it168.com/a2012/0919/1400/000001400252.shtml>
- [http://express.ruanko.com/ruanko-express\\_37/technologyexchange6.html](http://express.ruanko.com/ruanko-express_37/technologyexchange6.html)
- <http://nepoulia.blog.51cto.com/2722590/579151>
- 10 个技巧让你的 RESTful Web 服务更加实用
- 为您的 Web 项目构建一个简单的 JSON 控制器
- 在 CodeIgniter 框架中使用 RESTful 服务
- 浅谈 REST