RESTful，是目前最为流行的一种互联网软件架构。因为它结构清晰、符合标准、易于理解、扩展方便，所以正得到越来越多网站的采用。本小节我们将来学习它到底是一种什么样的架构？以及在Go里面如何来实现它。

**什么是REST**

REST(REpresentational State Transfer)这个概念，首次出现是在 2000年Roy Thomas Fielding（他是HTTP规范的主要编写者之一）的博士论文中，它指的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的。

要理解什么是REST，我们需要理解下面几个概念:

* 资源（Resources） REST是"表现层状态转化"，其实它省略了主语。"表现层"其实指的是"资源"的"表现层"。

那么什么是资源呢？就是我们平常上网访问的一张图片、一个文档、一个视频等。这些资源我们通过URI来定位，也就是一个URI表示一个资源。

* 表现层（Representation）

资源是做一个具体的实体信息，他可以有多种的展现方式。而把实体展现出来就是表现层，例如一个txt文本信息，他可以输出成html、json、xml等格式，一个图片他可以jpg、png等方式展现，这个就是表现层的意思。

URI确定一个资源，但是如何确定它的具体表现形式呢？应该在HTTP请求的头信息中用Accept和Content-Type字段指定，这两个字段才是对"表现层"的描述。

* 状态转化（State Transfer）

访问一个网站，就代表了客户端和服务器的一个互动过程。在这个过程中，肯定涉及到数据和状态的变化。而HTTP协议是无状态的，那么这些状态肯定保存在服务器端，所以如果客户端想要通知服务器端改变数据和状态的变化，肯定要通过某种方式来通知它。

客户端能通知服务器端的手段，只能是HTTP协议。具体来说，就是HTTP协议里面，四个表示操作方式的动词：GET、POST、PUT、DELETE。它们分别对应四种基本操作：GET用来获取资源，POST用来新建资源（也可以用于更新资源），PUT用来更新资源，DELETE用来删除资源。

综合上面的解释，我们总结一下什么是RESTful架构：

* （1）每一个URI代表一种资源；
* （2）客户端和服务器之间，传递这种资源的某种表现层；
* （3）客户端通过四个HTTP动词，对服务器端资源进行操作，实现"表现层状态转化"。

Web应用要满足REST最重要的原则是:客户端和服务器之间的交互在请求之间是无状态的,即从客户端到服务器的每个请求都必须包含理解请求所必需的信息。如果服务器在请求之间的任何时间点重启，客户端不会得到通知。此外此请求可以由任何可用服务器回答，这十分适合云计算之类的环境。因为是无状态的，所以客户端可以缓存数据以改进性能。

另一个重要的REST原则是系统分层，这表示组件无法了解除了与它直接交互的层次以外的组件。通过将系统知识限制在单个层，可以限制整个系统的复杂性，从而促进了底层的独立性。

下图即是REST的架构图：

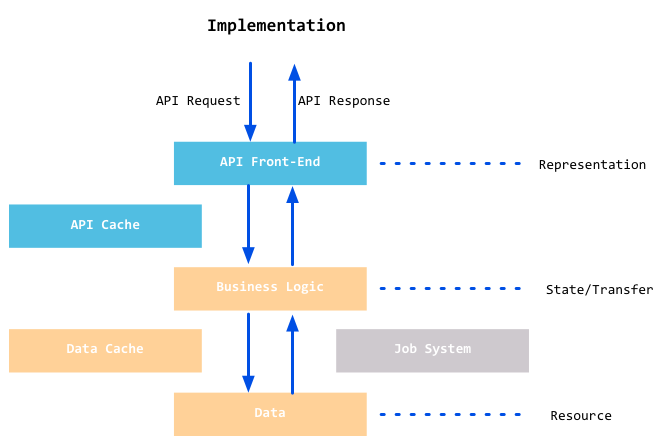
[](https://github.com/astaxie/build-web-application-with-golang/blob/master/zh/images/8.3.rest2.png?raw=true)

图8.5 REST架构图

当REST架构的约束条件作为一个整体应用时，将生成一个可以扩展到大量客户端的应用程序。它还降低了客户端和服务器之间的交互延迟。统一界面简化了整个系统架构，改进了子系统之间交互的可见性。REST简化了客户端和服务器的实现，而且对于使用REST开发的应用程序更加容易扩展。

下图展示了REST的扩展性：

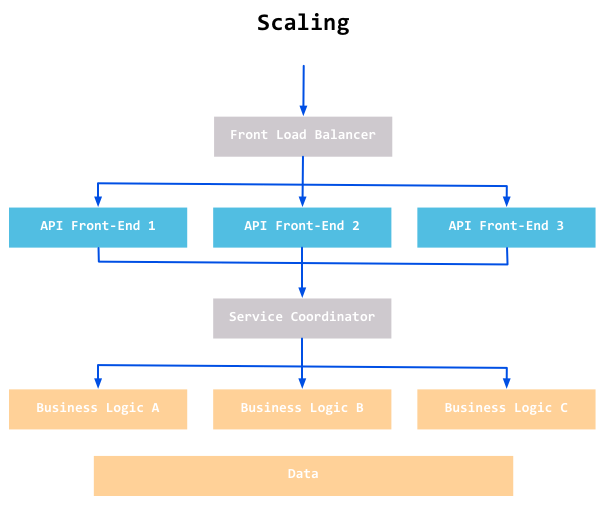
[](https://github.com/astaxie/build-web-application-with-golang/blob/master/zh/images/8.3.rest.png?raw=true)

图8.6 REST的扩展性

**RESTful的实现**

Go没有为REST提供直接支持，但是因为RESTful是基于HTTP协议实现的，所以我们可以利用net/http包来自己实现，当然需要针对REST做一些改造，REST是根据不同的method来处理相应的资源，目前已经存在的很多自称是REST的应用，其实并没有真正的实现REST，我暂且把这些应用根据实现的method分成几个级别，请看下图：

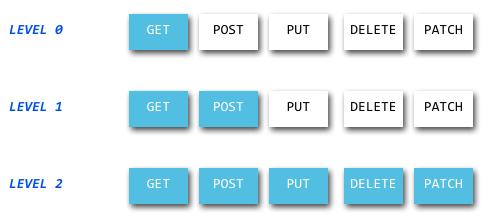
[](https://github.com/astaxie/build-web-application-with-golang/blob/master/zh/images/8.3.rest3.png?raw=true)

图8.7 REST的level分级

上图展示了我们目前实现REST的三个level，我们在应用开发的时候也不一定全部按照RESTful的规则全部实现他的方式，因为有些时候完全按照RESTful的方式未必是可行的，RESTful服务充分利用每一个HTTP方法，包括DELETE和PUT。可有时，HTTP客户端只能发出GET和POST请求：

* HTML标准只能通过链接和表单支持GET和POST。在没有Ajax支持的网页浏览器中不能发出PUT或DELETE命令
* 有些防火墙会挡住HTTP PUT和DELETE请求，要绕过这个限制，客户端需要把实际的PUT和DELETE请求通过 POST 请求穿透过来。RESTful 服务则要负责在收到的 POST 请求中找到原始的 HTTP 方法并还原。

我们现在可以通过POST里面增加隐藏字段\_method这种方式可以来模拟PUT、DELETE等方式，但是服务器端需要做转换。我现在的项目里面就按照这种方式来做的REST接口。当然Go语言里面完全按照RESTful来实现是很容易的，我们通过下面的例子来说明如何实现RESTful的应用设计。

package main

import (

"fmt"

"log"

"net/http"

"github.com/julienschmidt/httprouter"

)

func Index(w http.ResponseWriter, r \*http.Request, \_ httprouter.Params) {

fmt.Fprint(w, "Welcome!\n")

}

func Hello(w http.ResponseWriter, r \*http.Request, ps httprouter.Params) {

fmt.Fprintf(w, "hello, %s!\n", ps.ByName("name"))

}

func getuser(w http.ResponseWriter, r \*http.Request, ps httprouter.Params) {

uid := ps.ByName("uid")

fmt.Fprintf(w, "you are get user %s", uid)

}

func modifyuser(w http.ResponseWriter, r \*http.Request, ps httprouter.Params) {

uid := ps.ByName("uid")

fmt.Fprintf(w, "you are modify user %s", uid)

}

func deleteuser(w http.ResponseWriter, r \*http.Request, ps httprouter.Params) {

uid := ps.ByName("uid")

fmt.Fprintf(w, "you are delete user %s", uid)

}

func adduser(w http.ResponseWriter, r \*http.Request, ps httprouter.Params) {

// uid := r.FormValue("uid")

uid := ps.ByName("uid")

fmt.Fprintf(w, "you are add user %s", uid)

}

func main() {

router := httprouter.New()

router.GET("/", Index)

router.GET("/hello/:name", Hello)

router.GET("/user/:uid", getuser)

router.POST("/adduser/:uid", adduser)

router.DELETE("/deluser/:uid", deleteuser)

router.PUT("/moduser/:uid", modifyuser)

log.Fatal(http.ListenAndServe(":8080", router))

}

上面的代码演示了如何编写一个REST的应用，我们访问的资源是用户，我们通过不同的method来访问不同的函数，这里使用了第三方库github.com/julienschmidt/httprouter，在前面章节我们介绍过如何实现自定义的路由器，这个库实现了自定义路由和方便的路由规则映射，通过它，我们可以很方便的实现REST的架构。通过上面的代码可知，REST就是根据不同的method访问同一个资源的时候实现不同的逻辑处理。

**总结**

REST是一种架构风格，汲取了WWW的成功经验：无状态，以资源为中心，充分利用HTTP协议和URI协议，提供统一的接口定义，使得它作为一种设计Web服务的方法而变得流行。在某种意义上，通过强调URI和HTTP等早期Internet标准，REST是对大型应用程序服务器时代之前的Web方式的回归。目前Go对于REST的支持还是很简单的，通过实现自定义的路由规则，我们就可以为不同的method实现不同的handle，这样就实现了REST的架构。