21 | 网络编程：Go 语言如何玩转 RESTful API 服务?

从这一讲开始，我将带你学习本专栏的第五模块，在这个模块中，你将学到我们项目中最常用的编码操作，也就是编写 RESTful API 和 RPC 服务。在实际开发项目中，你编写的这些服务可以被其他服务使用，这样就组成了微服务的架构；也可以被前端调用，这样就可以前后端分离。

今天我就先来为你介绍什么是 RESTful API，以及 Go 语言是如何玩转 RESTful API 的。

什么是 RESTful API

RESTful API 是一套规范，它可以规范我们如何对服务器上的资源进行操作。在了解 RESTful API 之前，我先为你介绍下 HTTP Method，因为 RESTful API 和它是密不可分的。

说起 HTTP Method，最常见的就是**POST**和**GET**，其实最早在 HTTP 0.9 版本中，只有一个**GET**方法，该方法是一个**幂等方法**，用于获取服务器上的资源，也就是我们在浏览器中直接输入网址回车请求的方法。

在 HTTP 1.0 版本中又增加了**HEAD**和**POST**方法，其中常用的是 POST 方法，一般用于给服务端提交一个资源，导致服务器的资源发生变化。

随着网络越来越复杂，发现这两个方法是不够用的，就继续新增了方法。所以在 HTTP1.1 版本的时候，一口气增加到了 9 个，新增的方法有 HEAD、OPTIONS、PUT、DELETE、TRACE、PATCH 和 CONNECT。下面我为你一一介绍它们的作用。

1. GET 方法可请求一个指定资源的表示形式，使用 GET 的请求应该只被用于获取数据。
2. HEAD 方法用于请求一个与 GET 请求的响应相同的响应，但没有响应体。
3. POST 方法用于将实体提交到指定的资源，通常导致服务器上的状态变化或副作用。
4. PUT 方法用于请求有效载荷替换目标资源的所有当前表示。
5. DELETE 方法用于删除指定的资源。
6. CONNECT 方法用于建立一个到由目标资源标识的服务器的隧道。
7. OPTIONS 方法用于描述目标资源的通信选项。
8. TRACE 方法用于沿着到目标资源的路径执行一个消息环回测试。
9. PATCH 方法用于对资源应用部分修改。

从以上每个方法的介绍可以看到，HTTP 规范针对每个方法都给出了明确的定义，所以我们使用的时候也要尽可能地**遵循这些定义**，这样我们在开发中才可以更好地协作。

理解了这些 HTTP 方法，就可以更好地理解 RESTful API 规范了，因为 RESTful API 规范就是基于这些 HTTP 方法规范我们对服务器资源的操作，同时规范了 URL 的样式和 HTTP Status Code。

在 RESTful API 中，使用的主要是以下五种 HTTP 方法：

1. GET，表示读取服务器上的资源；
2. POST，表示在服务器上创建资源；
3. PUT，表示更新或者替换服务器上的资源；
4. DELETE，表示删除服务器上的资源；

5. PATCH，表示更新 / 修改资源的一部分。

以上 HTTP 方法在 RESTful API 规范中是一个操作，操作的就是服务器的资源，服务器的资源通过特定的 URL 表示。

现在我们通过一些示例让你更好地理解 RESTful API，如下所示：

```
HTTP GET https://www.flysnow.org/users
HTTP GET https://www.flysnow.org/users/123
```

以上是两个 GET 方法的示例：

- 第一个表示获取所有用户的信息；
- 第二个表示获取 ID 为 123 用户的信息。

下面再看一个 POST 方法的示例，如下所示：

```
HTTP POST https://www.flysnow.org/users
```

这个示例表示创建一个用户，通过 POST 方法给服务器提供创建这个用户所需的全部信息。

> 注意：这里 users 是个复数。

现在你已经知道了如何创建一个用户，那么如果要更新某个特定的用户怎么做呢？其实也非常简单，示例代码如下所示：

```
HTTP PUT https://www.flysnow.org/users/123
```

这表示要更新 / 替换 ID 为 123 的这个用户，在更新的时候，会通过 PUT 方法提供更新这个用户需要的全部用户信息。这里 PUT 方法和 POST 方法不太一样的是，从 URL 上看，PUT 方法操作的是单个资源，比如这里 ID 为 123 的用户。

> 小提示：如果要更新一个用户的部分信息，使用 PATCH 方法更恰当。

看到这里，相信你已经知道了如何删除一个用户，示例代码如下所示：

```
HTTP DELETE https://www.flysnow.org/users/123
```

DELETE 方法的使用和 PUT 方法一样，也是操作单个资源，这里是删除 ID 为 123 的这个用户。

一个简单的 RESTful API

相信你已经非常了解什么是 RESTful API 了，现在开始，我会带你通过一个使用 Golang 实现 RESTful API 风格的示例，加深 RESTful API 的理解。

Go 语言的一个很大的优势，就是可以很容易地开发出网络后台服务，而且性能快、效率高。在开发后端 HTTP 网络应用服务的时候，我们需要处理很多 HTTP 的请求访问，比如常见的RESTful API 服务，就要处理很多 HTTP 请求，然后把处理的信息返回给使用者。对于这类需求，Golang 提供了内置的 net/http 包帮我们处理这些 HTTP 请求，让我们可以比较方便地开发一个 HTTP 服务。

下面我们来看一个简单的 HTTP 服务的 Go 语言实现，代码如下所示：

ch21/main.go

```
func main() {
   http.HandleFunc("/users",handleUsers)
   http.ListenAndServe(":8080", nil)
}
func handleUsers(w http.ResponseWriter, r *http.Request){
   fmt.Fprintln(w,"ID:1,Name:张三")
   fmt.Fprintln(w,"ID:2,Name:李四")
   fmt.Fprintln(w,"ID:3,Name:王五")
}
```

这个示例运行后，你在浏览器中输入 http://localhost:8080/users, 就可以看到如下内容信息：

```
ID:1,Name:张三
ID:2,Name:李四
ID:3,Name:王五
```

也就是获取所有的用户信息，但是这并不是一个 RESTful API，因为使用者不仅可以通过 HTTP GET 方法获得所有的用户信息，还可以通过 POST、DELETE、PUT 等 HTTP 方法获得所有的用户信息，这显然不符合 RESTful API 的规范。

现在我对以上示例进行修改，使它符合 RESTful API 的规范，修改后的示例代码如下所示：

ch20/main.go

```
func handleUsers(w http.ResponseWriter, r *http.Request){
   switch r.Method {
   case "GET":
      w.WriteHeader(http.StatusOK)
      fmt.Fprintln(w,"ID:1,Name:张三")
      fmt.Fprintln(w,"ID:2,Name:李四")
      fmt.Fprintln(w,"ID:3,Name:王五")
   default:
      w.WriteHeader(http.StatusNotFound)
      fmt.Fprintln(w,"not found")
   }
}
```

这里我只修改了 handleUsers 函数，在该函数中增加了只在使用 GET 方法时，才获得所有用户的信息，其他情况返回 not found。

现在再运行这个示例，会发现只能通过 HTTP GET 方法进行访问了，使用其他方法会提示 not found。

RESTful JSON API

在项目中最常见的是使用 JSON 格式传输信息，也就是我们提供的 RESTful API 要返回 JSON 内容给使用者。

同样用上面的示例，我把它改造成可以返回 JSON 内容的方式，示例代码如下所示：

ch20/main.go

```
//数据源，类似MySQL中的数据
var users = []User{
    {ID: 1,Name: "张三"},
    {ID: 2,Name: "李四"},
    {ID: 3,Name: "王五"},
}
func handleUsers(w http.ResponseWriter, r *http.Request){
    switch r.Method {
    case "GET":
        users,err:=json.Marshal(users)
        if err!=nil {
            w.WriteHeader(http.StatusInternalServerError)
            fmt.Fprint(w,"{\"message\": \""+err.Error()+"\"}")
        }else {
            w.WriteHeader(http.StatusOK)
            w.Write(users)
        }
    default:
        w.WriteHeader(http.StatusNotFound)
        fmt.Fprint(w,"{\"message\": \"not found\"}")
    }
}
//用户
type User struct {
    ID int
    Name string
}
```

从以上代码可以看到，这次的改造主要是新建了一个 User 结构体，并且使用 users 这个切片存储所有的用户，然后在 handleUsers 函数中把它转化为一个 JSON 数组返回。这样，就实现了基于 JSON 数据格式的 RESTful API。

运行这个示例，在浏览器中输入 http://localhost:8080/users，可以看到如下信息：

```
[{"ID":1,"Name":"张三"},{"ID":2,"Name":"李四"},{"ID":3,"Name":"王五"}]
```

这已经是 JSON 格式的用户信息，包含了所有用户。

Gin 框架

虽然 Go 语言自带的 net/http 包，可以比较容易地创建 HTTP 服务，但是它也有很多不足：

- 不能单独地对请求方法（POST、GET 等）注册特定的处理函数；
- 不支持 Path 变量参数；
- 不能自动对 Path 进行校准；
- 性能一般；
- 扩展性不足；
- ……

基于以上这些不足，出现了很多 Golang Web 框架，如 Mux，Gin、Fiber 等，今天我要为你介绍的就是这款使用最多的 Gin 框架。

引入 Gin 框架

Gin 框架是一个在 Github 上开源的 Web 框架，封装了很多 Web 开发需要的通用功能，并且性能也非常高，可以让我们很容易地写出 RESTful API。

Gin 框架其实是一个模块，也就是 Go Mod，所以采用 Go Mod 的方法引入即可。我在第 18讲的时候详细介绍过如何引入第三方的模块，这里再复习一下。

首先需要下载安装 Gin 框架，安装代码如下：

```
$ go get -u github.com/gin-gonic/gin
```

然后就可以在 Go 语言代码中导入使用了，导入代码如下：

```
import "github.com/gin-gonic/gin"
```

通过以上安装和导入这两个步骤，就可以在你的 Go 语言项目中使用 Gin 框架了。

使用 Gin 框架

现在，已经引入了 Gin 框架，下面我就是用 Gin 框架重写上面的示例，修改的代码如下所示：

ch21/main.go

```
func main() {
   r:=gin.Default()
   r.GET("/users", listUser)
   r.Run(":8080")
}
func listUser(c *gin.Context)  {
   c.JSON(200,users)
}
```

相比 net/http 包，Gin 框架的代码非常简单，通过它的 GET 方法就可以创建一个只处理 HTTP GET 方法的服务，而且输出 JSON 格式的数据也非常简单，使用 c.JSON 方法即可。

最后通过 Run 方法启动 HTTP 服务，监听在 8080 端口。现在运行这个 Gin 示例，在浏览器中输入 http://localhost:8080/users，看到的信息和通过 net/http 包实现的效果是一样的。

获取特定的用户

现在你已经掌握了如何使用 Gin 框架创建一个简单的 RESTful API，并且可以返回所有的用户信息，那么如何获取特定用户的信息呢?

我们知道，如果要获得特定用户的信息，需要使用的是 GET 方法，并且 URL 格式如下所示：

```
http://localhost:8080/users/2
```

以上示例中的 2 是用户的 ID，也就是通过 ID 来获取特定的用户。

下面我通过 Gin 框架 Path 路径参数来实现这个功能，示例代码如下：

ch21/main.go

```
func main() {
    //省略没有改动的代码
    r.GET("/users/:id", getUser)
}
func getUser(c *gin.Context) {
    id := c.Param("id")
    var user User
    found := false
    //类似于数据库的SQL查询
    for _, u := range users {
        if strings.EqualFold(id, strconv.Itoa(u.ID)) {
            user = u
            found = true
            break
        }
    }
    if found {
        c.JSON(200, user)
    } else {
        c.JSON(404, gin.H{
            "message": "用户不存在",
        })
    }
}
```

在 Gin 框架中，路径中使用冒号表示 Path 路径参数，比如示例中的 :id，然后在 getUser 函数中可以通过 c.Param("id") 获取需要查询用户的 ID 值。

> 小提示：Param 方法的参数要和 Path 路径参数中的一致，比如示例中都是 ID。

现在运行这个示例，通过浏览器访问 http://localhost:8080/users/2，就可以获得 ID 为 2 的用户，输出信息如下所示：

```
{"ID":2,"Name":"李四"}
```

可以看到，已经正确的获取到了 ID 为 2 的用户，他的名字叫李四。

假如我们访问一个不存在的 ID，会得到什么结果呢？比如 99，示例如下所示：

```
➜ curl http://localhost:8080/users/99
{"message":"用户不存在"}%
```

从以上示例输出可以看到，返回了『用户不存在』的信息，和我们代码中处理的逻辑一样。

新增一个用户

现在你已经可以使用 Gin 获取所有用户，还可以获取特定的用户，那么你也应该知道如何新增一个用户了，现在我通过 Gin 实现如何新增一个用户，看和你想的方案是否相似。

根据 RESTful API 规范，实现新增使用的是 POST 方法，并且 URL 的格式为 http://localhost:8080/users ，向这个 URL 发送数据，就可以新增一个用户，然后返回创建的用户信息。

现在我使用 Gin 框架实现新增一个用户，示例代码如下：

```
func main() {
    //省略没有改动的代码
    r.POST("/users", createUser)
}
func createUser(c *gin.Context) {
    name := c.DefaultPostForm("name", "")
    if name != "" {
        u := User{ID: len(users) + 1, Name: name}
        users = append(users, u)
        c.JSON(http.StatusCreated,u)
    } else {
        c.JSON(http.StatusOK, gin.H{
            "message": "请输入用户名称",
        })
    }
}
```

以上新增用户的主要逻辑是获取客户端上传的 name 值，然后生成一个 User 用户，最后把它存储到 users 集合中，达到新增用户的目的。

在这个示例中，使用 POST 方法来新增用户，所以只能通过 POST 方法才能新增用户成功。

现在运行这个示例，然后通过如下命令发送一个新增用户的请求，查看结果：

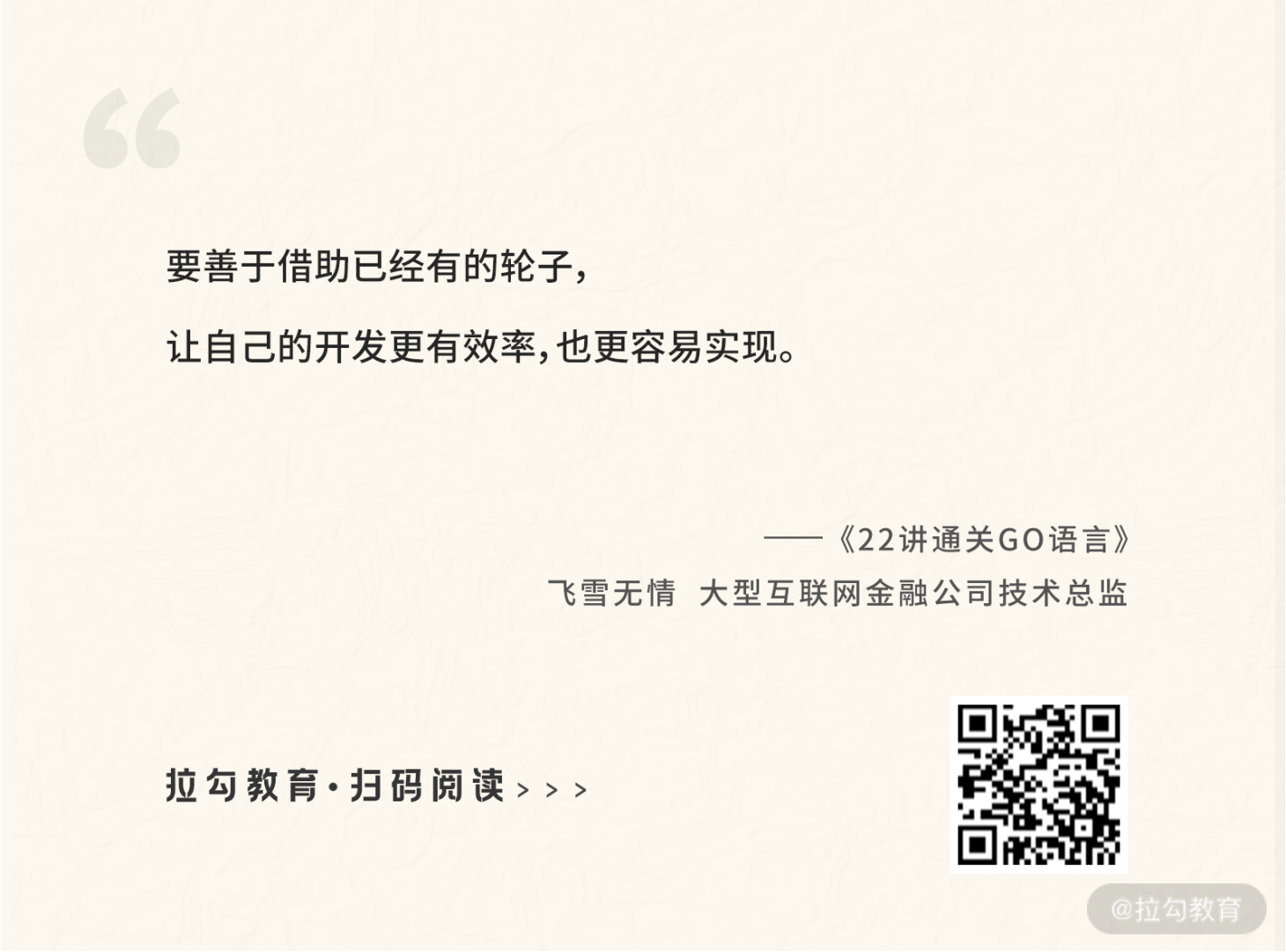
```
➜ curl -X POST -d 'name=飞雪' http://localhost:8080/users
{"ID":4,"Name":"飞雪"}
```

可以看到新增用户成功，并且返回了新增的用户，还有分配的 ID。

总结

Go 语言已经给我们提供了比较强大的 SDK，让我们可以很容易地开发网络服务的应用，而借助第三方的 Web 框架，可以让这件事情更容易、更高效。比如这篇文章介绍的 Gin 框架，就可以很容易让我们开发出 RESTful API，更多关于 Gin 框架的使用可以参考 Golang Gin 实战 系列文章。

在我们做项目开发的时候，要善于借助已经有的轮子，让自己的开发更有效率，也更容易实现。



在我们做项目开发的时候，会有增、删、改和查，现在增和查你已经学会了，那么就给你留 2 个作业，任选其中 1 个即可，它们是：

1. 修改一个用户的名字；
2. 删除一个用户。

下一讲，也就是本专栏的最后一讲，我将为你介绍如何使用 Go 语言实现 RPC 服务，记得来听课哦。