数据接口编程技术

[1 标题1 错误!未定义书签。](#_Toc449540698)

[1.1 标题2 **错误!未定义书签。**](#_Toc449540699)

[1.1.1 标题3 **错误!未定义书签。**](#_Toc449540700)

# 接口编程概述

web开发，发展到当前的典型模式。

基本架构思想：前后端分离，数据提供方与数据展示方分开的。

对比之前的经典展示的语法：



前后端分离：

浏览器端，称之为前端。

服务器端，称之为后端。

分离的是：前端，负责展示数据，从后端得到数据，完成与HTML的混合，展示给浏览器用户。后端，仅仅需要将数据处理好，发送给前端即可。

核心：原本在后端（服务器端）将数据和HTML混合，变为在前端将数据和HTML混合。

后端的核心：业务逻辑完整，高效，安全等方面

前端的核心：用户体验，效果等方面。



为什么要这个变化？

前端的多样性导致一份数据需要更多的展示方式。移动端，webapp，app，移动浏览器，pc端浏览器，都是前端终端的概念。

前端的性能足以满足在前端做渲染。

称呼的变化：

后端程序员，也叫接口开发人员。

前端程序员，也被叫做基于接口开发。

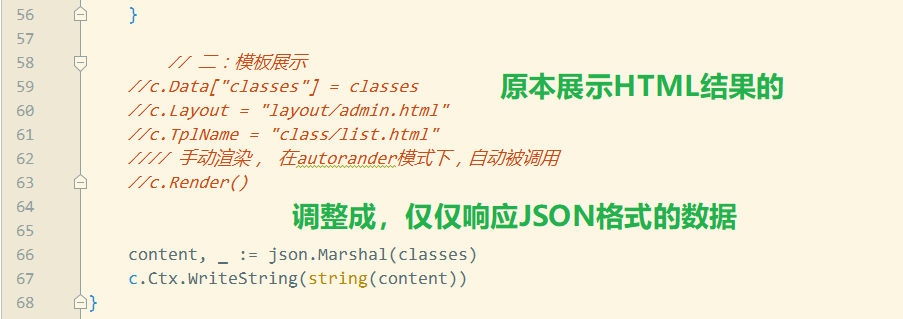
接口的意思：

前端程序，如何才能从后端程序获得数据？

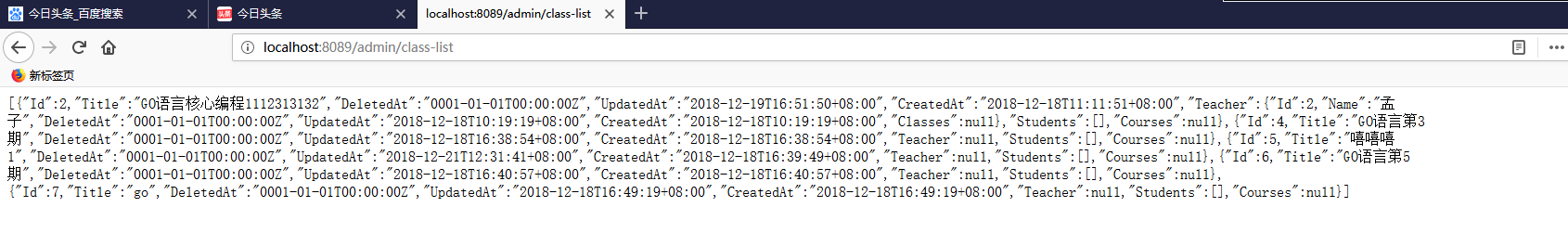
后端程序，如何才能将数据提供给前端程序？

通过后端程序提供的接口完成的。接口，基于某种协议的请求。典型的协议就是，HTTP，WebSocket，RPC协议。

例如，之前的班级列表，在不需要html模板处理后，直接响应json数据：

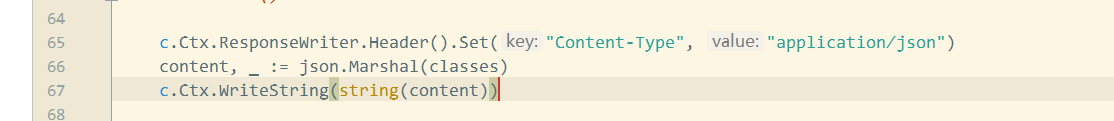


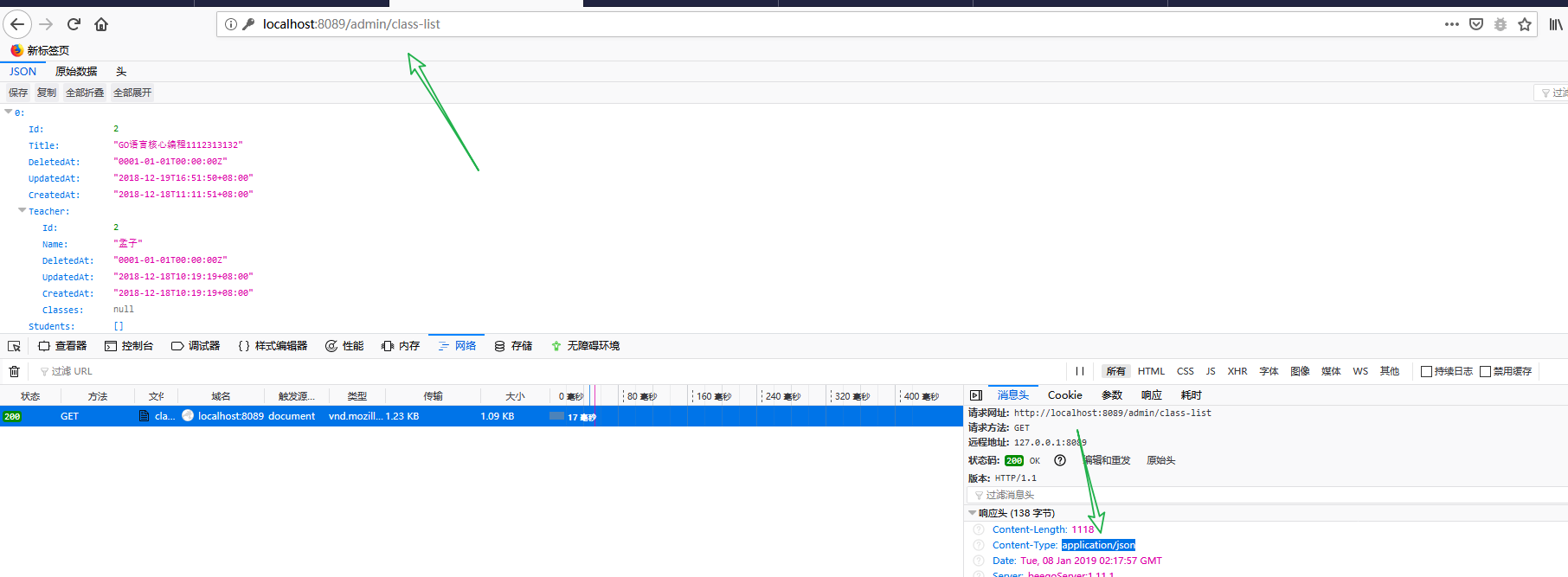
此时，浏览器（前端）请求，得到的是：



最好通过响应头Content-Type:application/json 的形式，告知浏览器，得到的是 json

结构：





前端主要是JS语法，JS通过向服务器发送请求，获取数据，再操作HTML（DOM）完成展示，都是在浏览器端完成的。

## 常见的前端



## 后端还需做什么？

后端的主要任务是为前端提供数据。

另一个主要任务就是：离线运算。上面的所说的，都叫在线运算。

在线运算：前端请求，后端立即给出响应，就是在线的。在线的主要问题是并发，即时响应。（reactor(epoll), 负载均衡，多进程，多线程）

离线运算：统计类工作，大数量的计算，都需要线下执行。利用已有的数据，将统计结果生成好，提供给用户。已有的数据，一定会滞后于当前最新实时数据。主要的难点是运算速度。

# JSON格式

JSON格式是目前主要的前后端交互的格式。

JSON：典型的Key=>Value 型数据结构。

格式如下：

|  |
| --- |
| {  **"key1"**:"stringValue",  **"key2"**:42,  **"key3"**:**true**,  **"key4"**: 42.2,  **"key5"**: {  **"key51"**: 23,  **"key52"**: "name",  },  **"key6"**: [  {  **"key611"**: 12,  **"key612"** "title"  },  {  **"key621"**: 45,  **"key622"**: "kang"  },  ]  } |

优势是：

* 支持常见的数据类型，支持嵌套，功能齐备
* 量级轻，语法简单。
* 与JS友好，JS是主要的前端语言

在web的前后端交互时，以json为主。

后端可以非常方便的生成json格式：

在Go中：content, \_ := json.Marshal(classes)

在Beego框架中：

c.Data["json"] = classes

c.ServeJSON()

# XML格式

extensible markup language，可扩展标记语言

与HTML的语法类似，区别是标签是自定义的。

格式如下：

<?xml version="1.0" encoding="utf-8"?>

<notes>

<note>

<to>George</to>

<from>John</from>

<heading>Reminder</heading>

<body>Don't forget the meeting!</body>

</note>

<note>

<to>Kang</to>

<from>Han</from>

<heading>Warning</heading>

<body>Don't forget the meeting!</body>

</note>

</notes>

服务器端，可以：

Go语法:xml.Marshal()

beego:

this.Data["xml"]=&mystruct

this.ServeXML()

# 响应固定格式数据和头信息

以 Beego 使用Ctx 演示为例：

// 设置头信息，告知浏览器，得到的内容为json格式

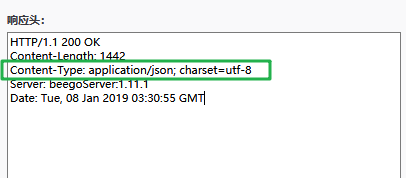
c.Ctx.ResponseWriter.Header().Set("Content-Type", "application/json")

// 发送json格式的响应主体

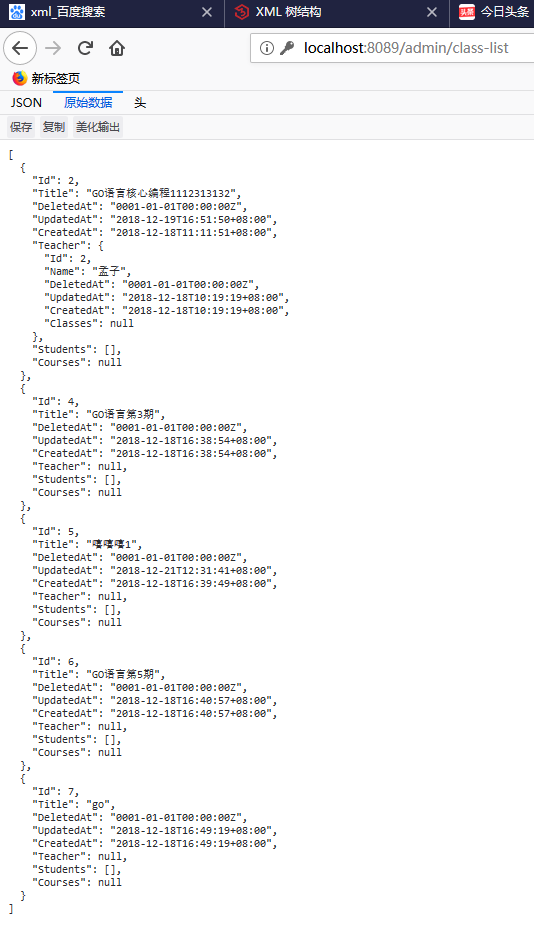
content, \_ := json.Marshal(classes)

c.Ctx.WriteString(string(content))

浏览器端得到的：

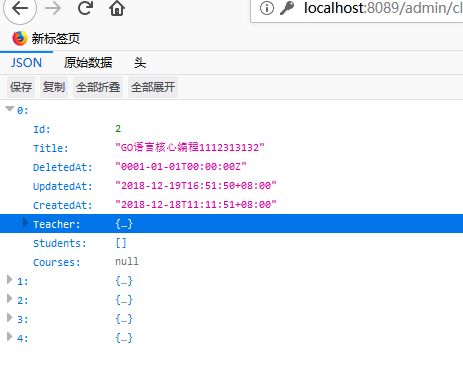


原始响应数据：字符串，json格式的字符串



前端就可以基于该格式，进行格式化输出：

比如浏览器：：支持折叠的展示了！



beego的c.ServeJSON()函数，已经可以完成设置头，和主体两件事情！

# token认证

在接口编程中，认证方式需要更改！

传统的认证方式，是基于在前（浏览器）端的cookie中存储session标识，完成的会话跟踪！

在接口的编程中，不能保证前端程序一定支持cookie技术。若前端不支持cookie，不能使用cookie来传递会话标识。

问题就是如何处理会话跟踪，如何处理用户认证？使用URL上的查询字符串参数，完成认证。

目前接口编程后端中，采用的认证模式，就是token模式，就是买票模式。

思路：

先通过用户名密码，得到票（token）。除了用户名密码也支持其他方式，例如secret-key。

带着token（票）请求需要的资源。

token（票），有时效性的概念，生成一张票，24分钟有效，携带票，请求时，判断票是否过期才可以。

前端的请求URL：

resource.json?token=kdjalkcciemd91j3mcdkmmid8e72

## 服务器端的处理

需要的操作。

判断用户合法性时，如果合法生成票。（原来：在session中存储登录标识）

验证是否认证时，需要使用票进行验证。（原来： 验证session中的标识）

演示生成token，响应给用户：



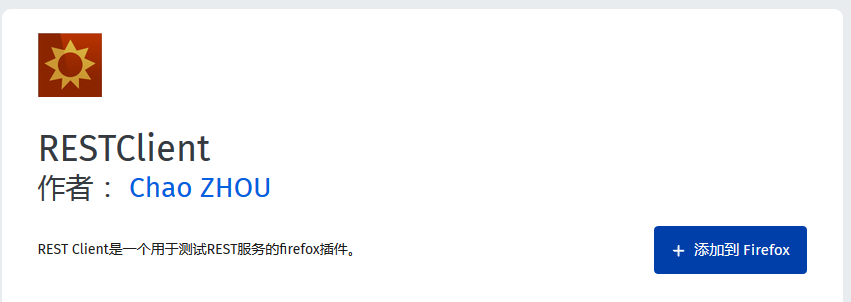
## 服务器端接口的验证

请求方式有多种多样。

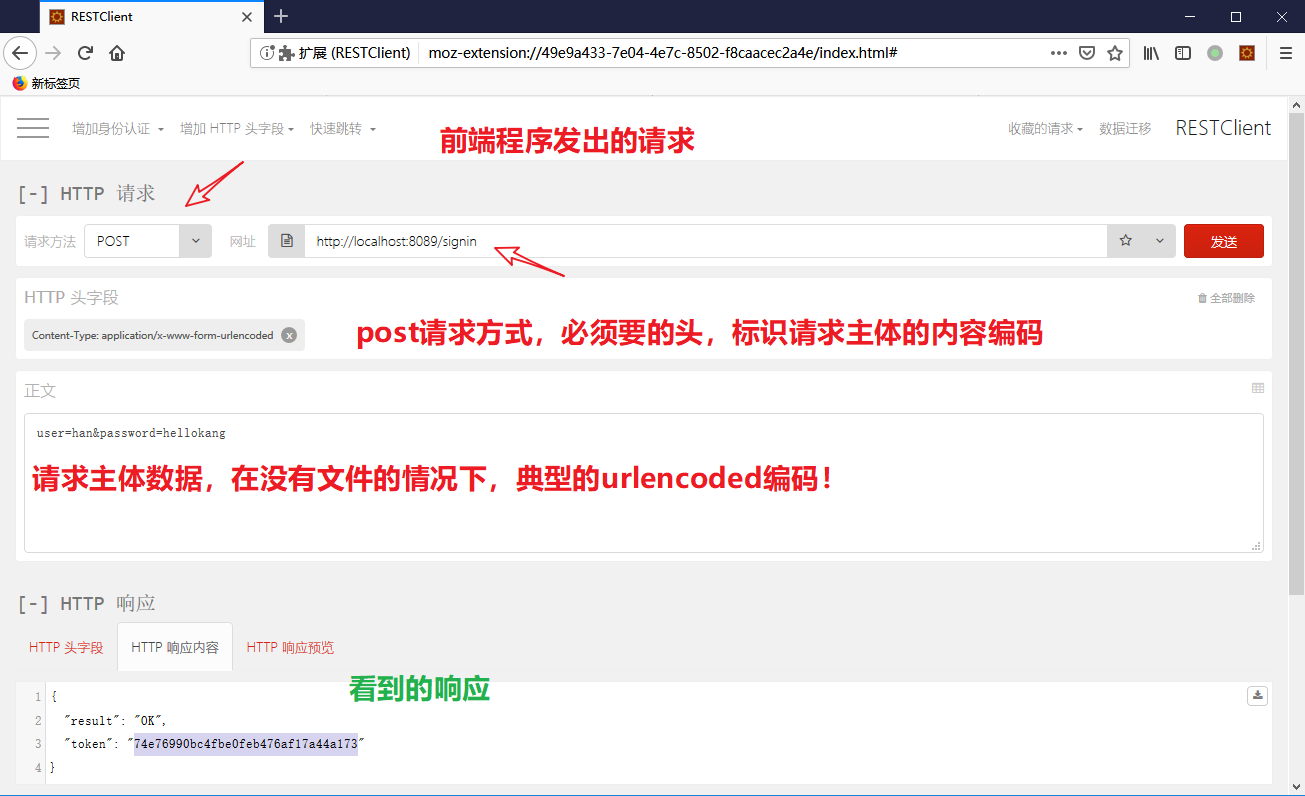
需要一个可以模拟各种请求的工具来测试服务器端口。

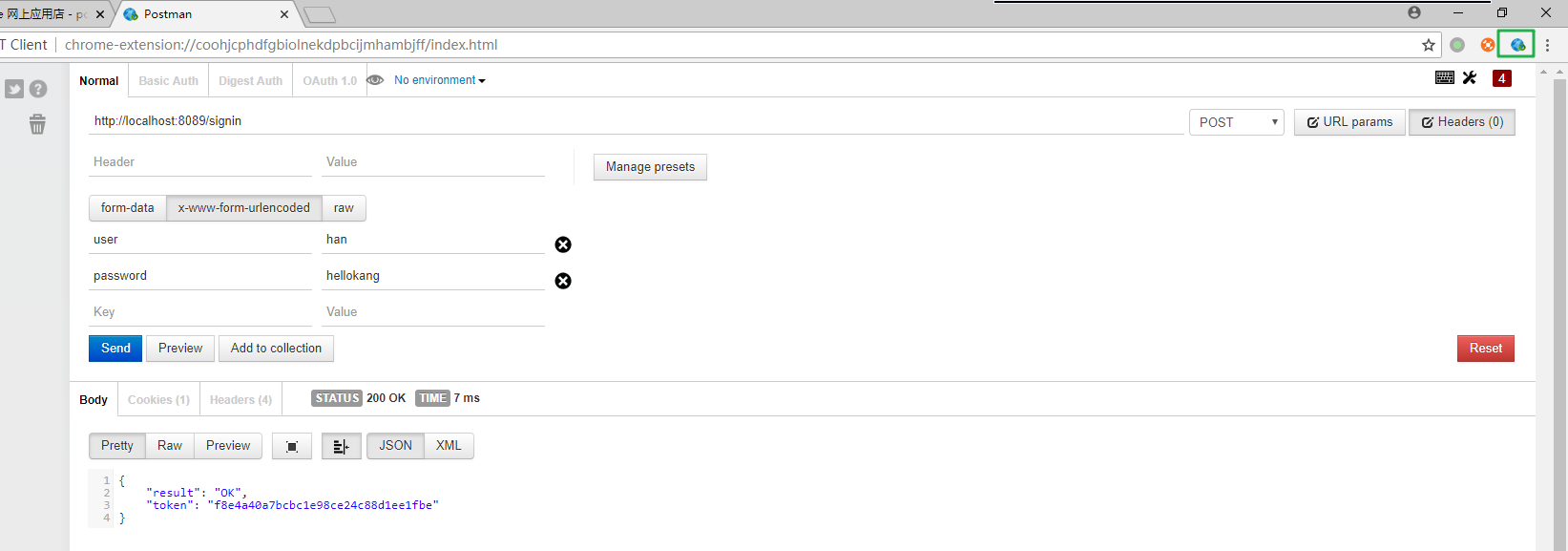
（通过浏览器在地址栏键入地址的方式，只能发出GET请求）

利用浏览器工具：postman, restclinet 工具完成。









## 设定票有效期，和与用户的关联

需要在生成票的时候，完成记录票的生成时间，同时记录与用户的关联。

需要知道：

* 票是否过期
* 票与用户的关联

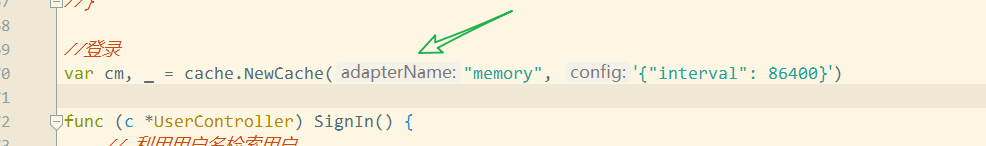
典型的存储方案：

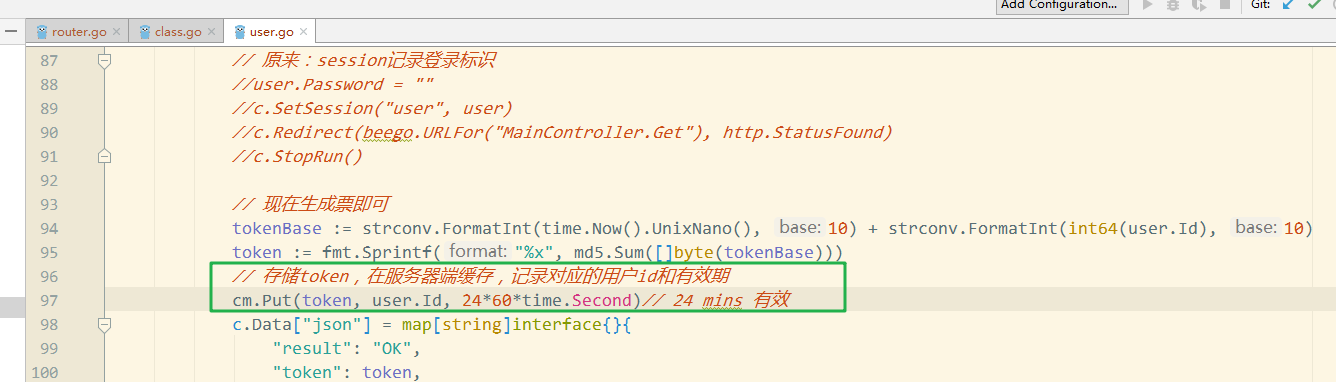
数据表：4个字段：ID(pk), token，user\_id, created\_time

缓存存储方案：key->value, 同时具有有效期。可以使用token=>user.Id，有效期记录有效时间

选择缓存方案：

在生成完毕token后：

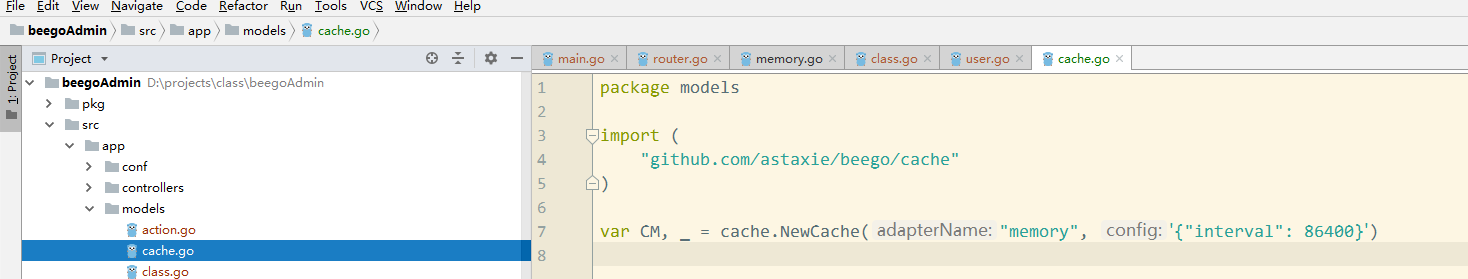




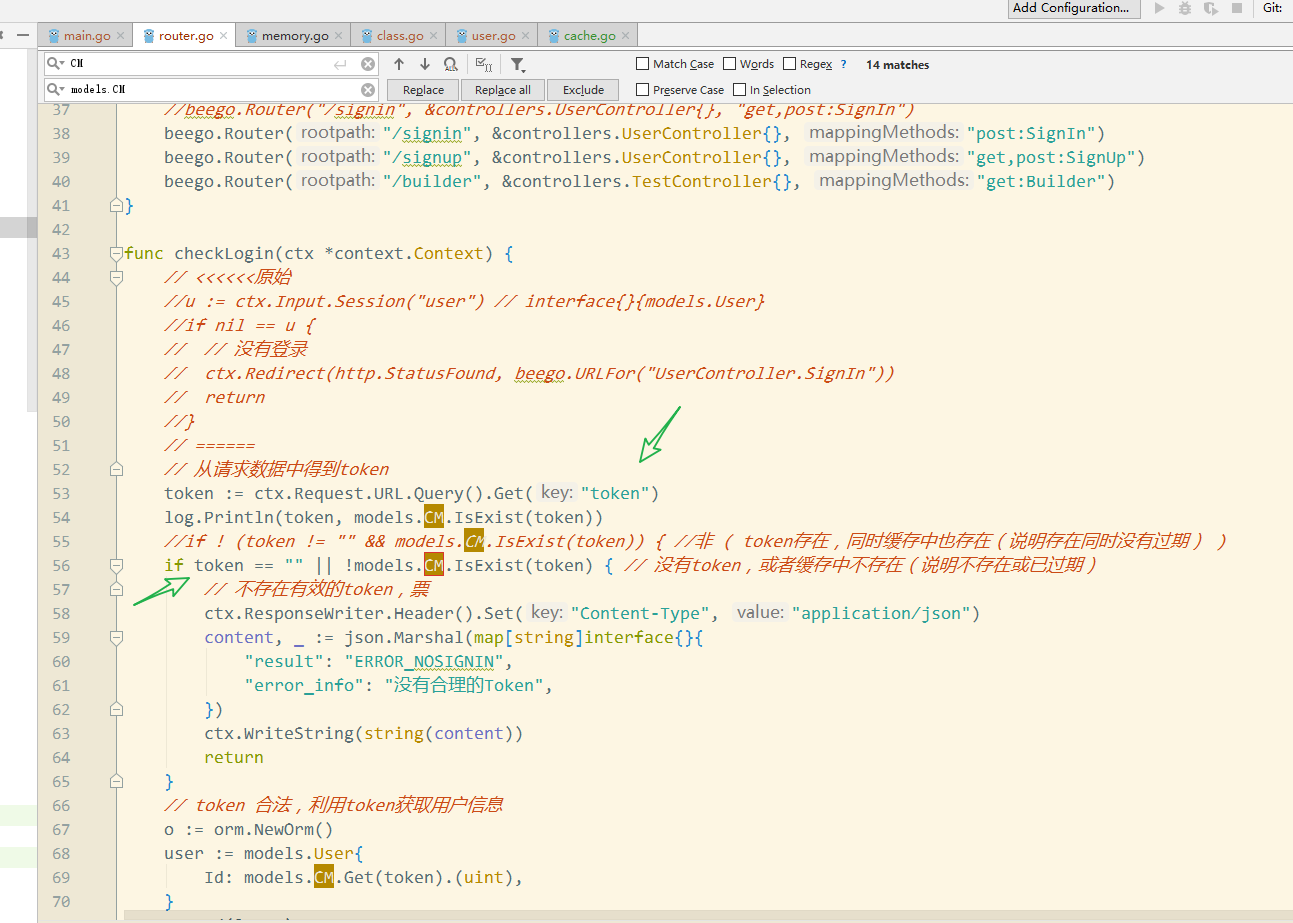
## 校验票

原来校验的session中的登录标识，现在需要校验缓存中的token！

在models.cache 中，完成了缓存的初始

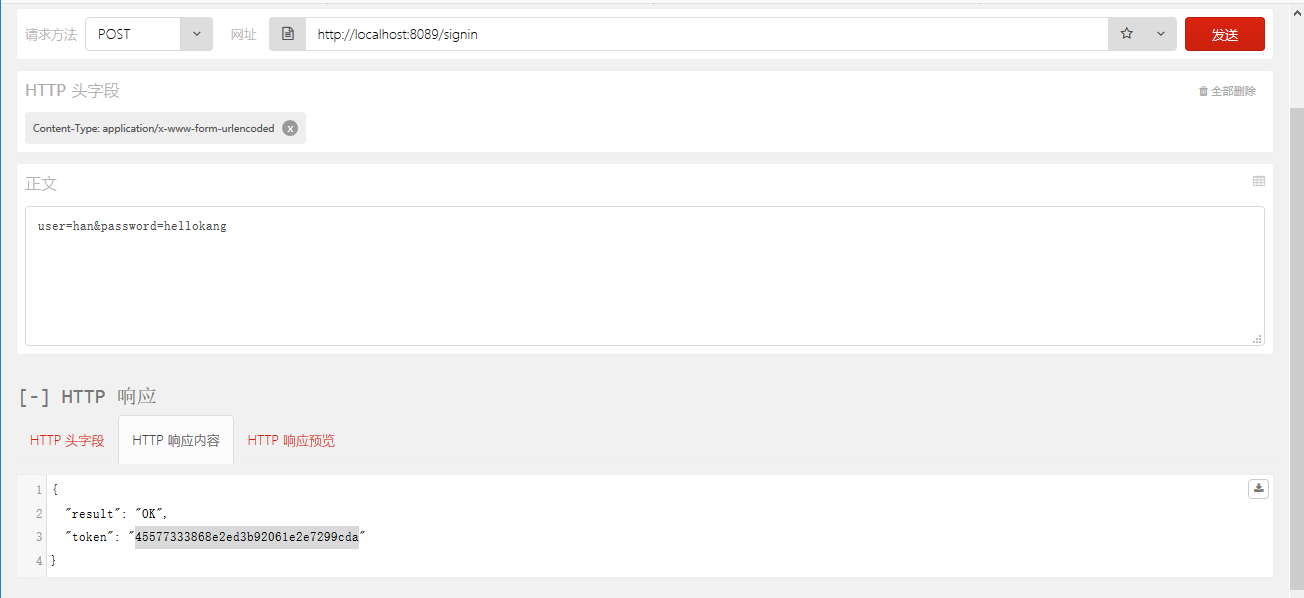


router.go

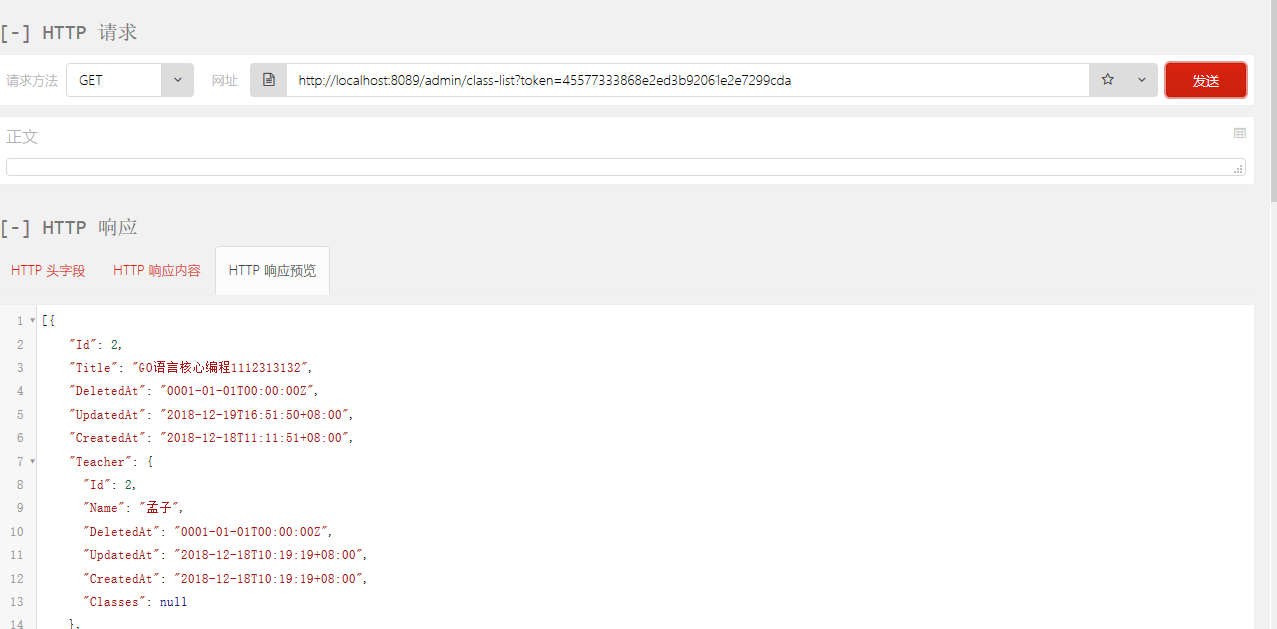


测试：

先请求获取token：



携带token获取数据：



前端在编程时，获取token后，需要存在前端一段时间例如24分钟。先判断前端本地是否有token，有就直接使用，否则再生产成本！

## 无状态的数据接口

没有会话（cookie，session）的操作！

# restful接口

一套接口规范

主要用在对资源CRUD管理领域。

若采用前后端分离的程序，做管理系统，对资源进行CRUD操作，适合使用restful。

（一种软件架构风格、设计风格）

REST: Representational State Transfer, 表述性状态传递。

主要目的，通过请求的method和URI的组合，确定需要对服务器的资源操作。

例如：对班级的管理，若将URI和Method的组合规划为下面的模式，在服务器端实现了restful接口：

|  |  |  |
| --- | --- | --- |
| Method | URI  统一资源标识符 | 操作 |
| GET | classes | 获取多个班级 |
| GET | class/42 | 获取标识为42的班级 |
| DELETE | class/42 | 删除标识为42的班级 |
| PUT | class/42 | 更新标识为42的班级 |
| POST | class | 添加一个班级 |

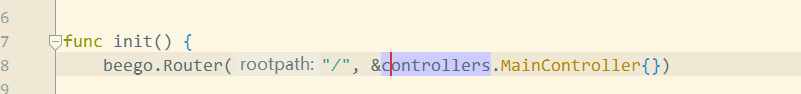
之前的前端只有浏览器，而浏览器仅仅支持post和get的方式。如果需要delete或者put完成删除或者更新的逻辑，需要URI上增加额外的参数来标识，class/delete, class/edit。

现在前端不光有浏览器，很多程序app，js都可以发出请求，希望按照原来的HTTP标准来实现，非浏览器发出请求，可以支持get,post,delete,put, 就可以通过method来区别操作。采用风格来发出请求。

编程时，若支持restful风格，路由体系需要，可以通过请求方式，来确定动作。

beego支持restful风格，默认支持。

路由：



控制器：

