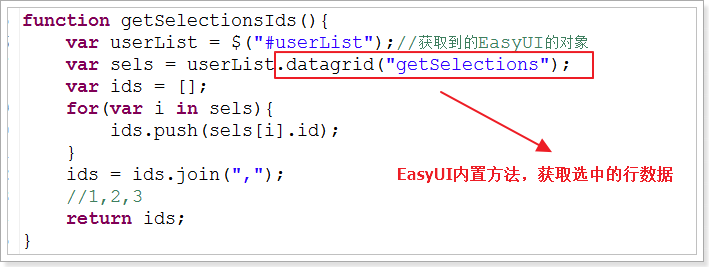
# 今日大纲

1. 完成用户管理系统，完成编辑、删除、导出；
2. 学习mybatis的高级用法之通用Mapper；
3. RESTful Web Service – 系统间的接口调用
4. 改造用户管理系统，使用通用Mapper和RESTful；

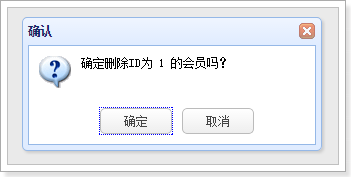
# 完成用户管理系统

## 删除

获取选中的行的数据id：

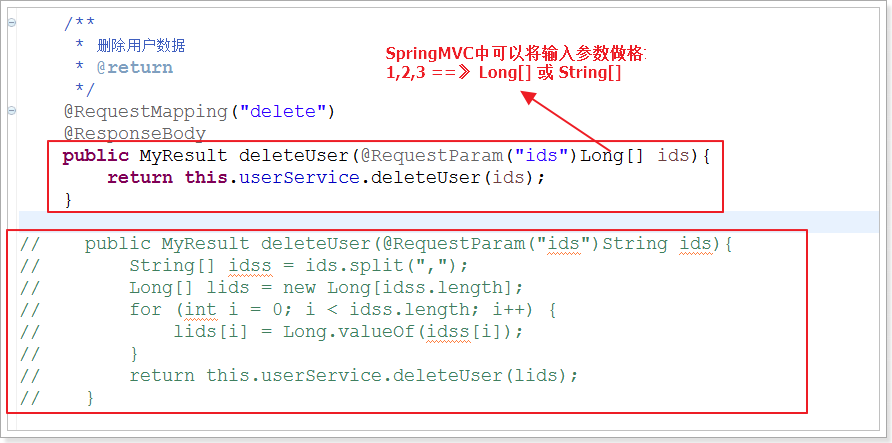


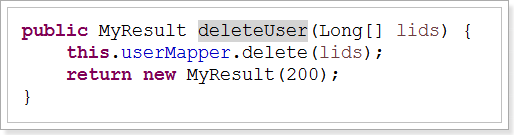
删除前的确认：

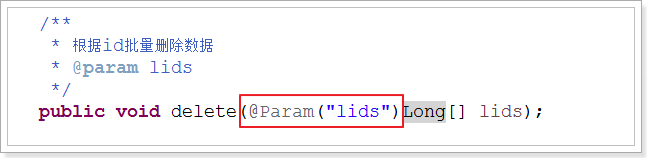


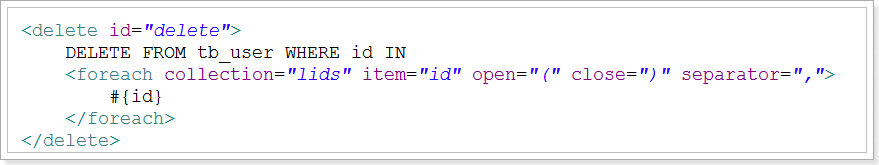


后台逻辑的实现：









## 编辑

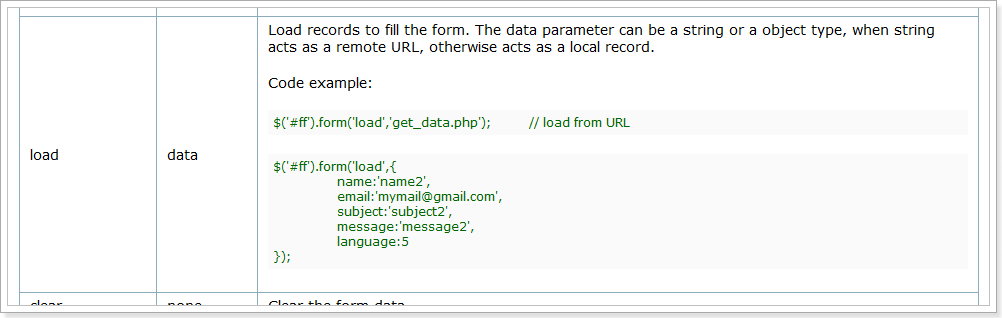
### 前台逻辑实现

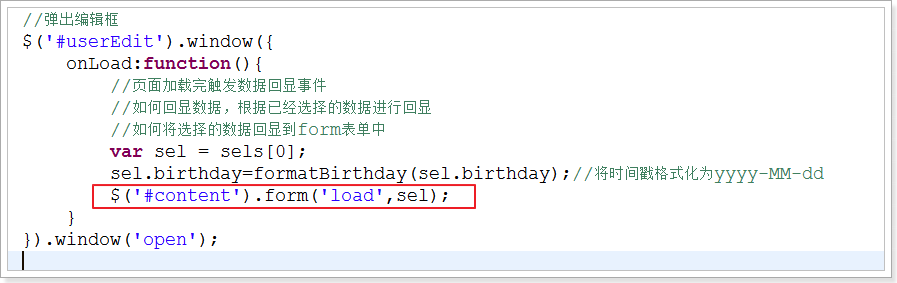
在CRUD中最麻烦的是编辑，因为有个数据回显的过程。

编辑的用户交互的过程：

1. 用户选择需要编辑的数据（判断是否选择一行数据）
2. 弹出编辑框，完成编辑页面的加载
3. 页面加载完成后对form表单做数据回显
4. 用户提交实现编辑

在EasyUI中如何实现数据的回显：

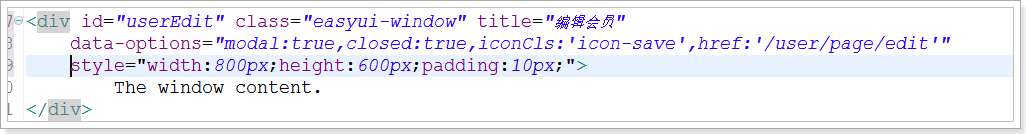




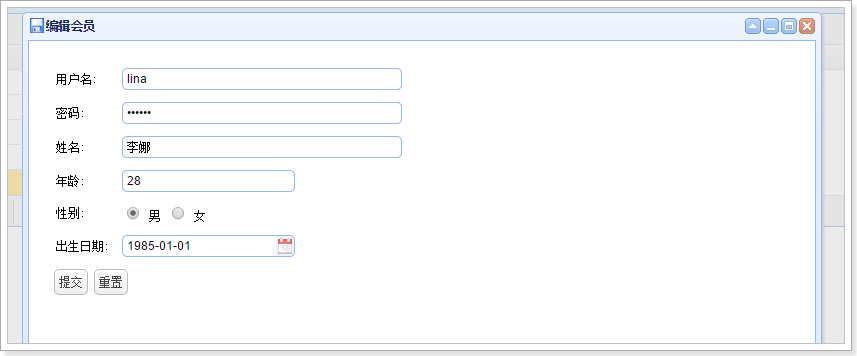
Sel：



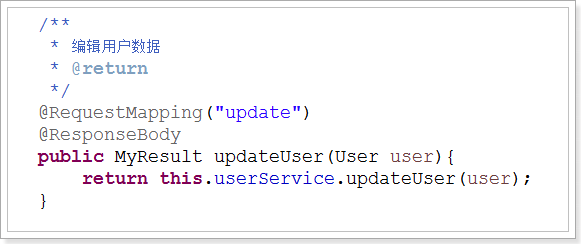
Window：

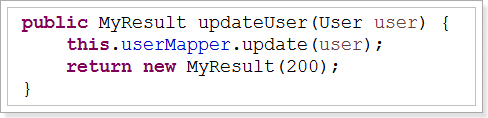


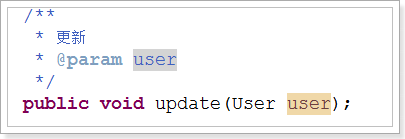
效果：

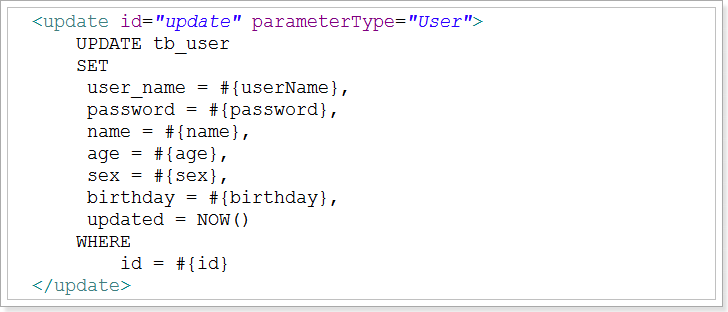


### 后台逻辑实现





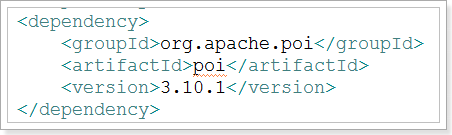
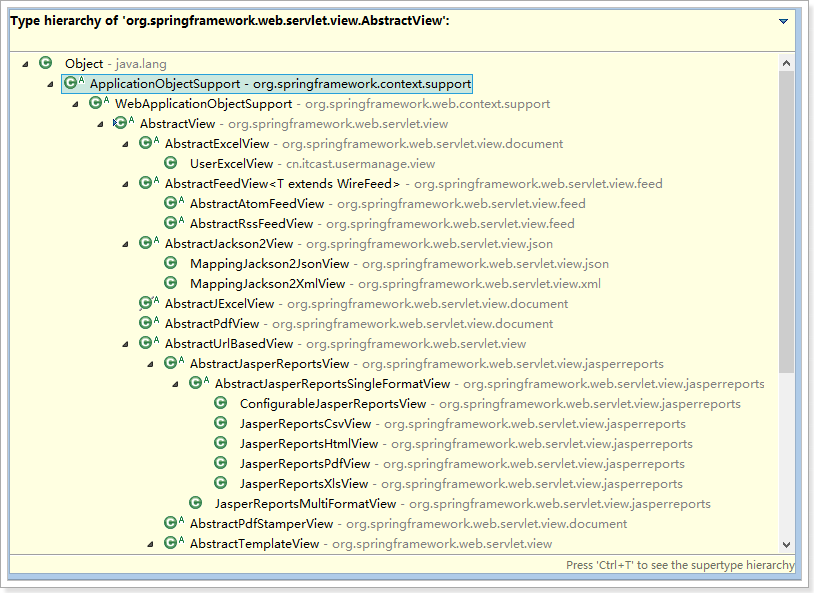
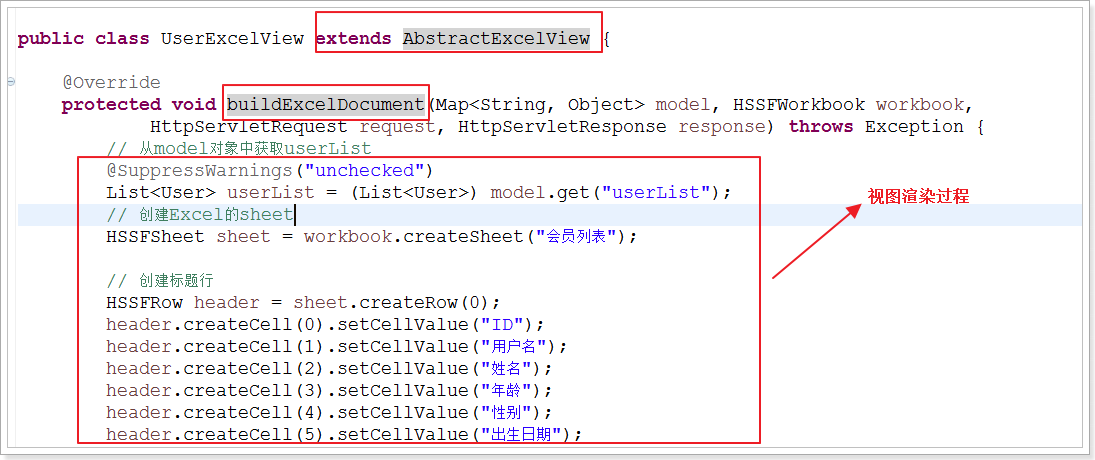
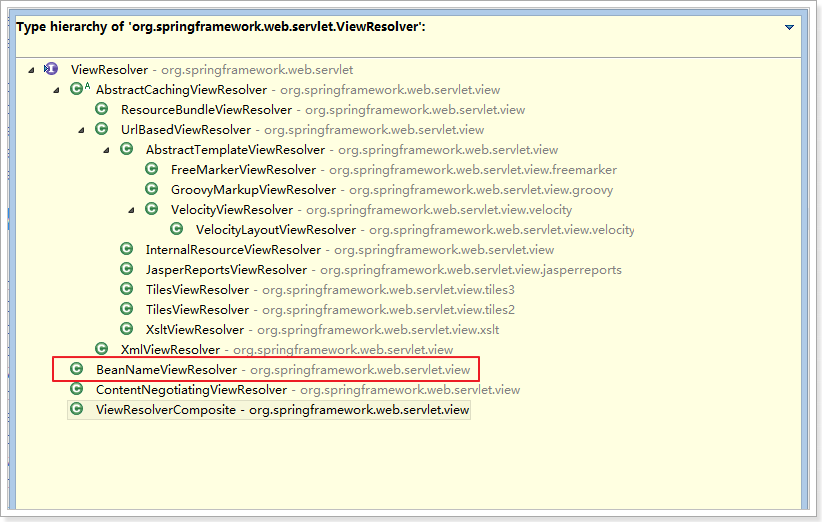
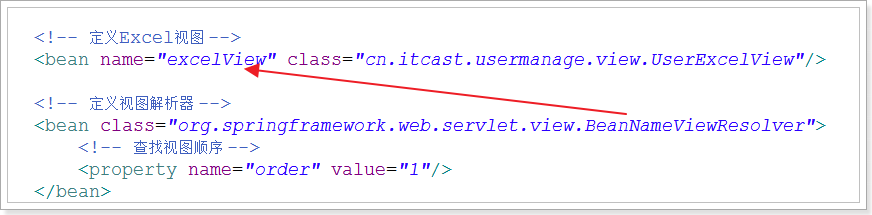




## 导出

产品经理提出需求：导出当前页的数据。

Excel视图实现思路：

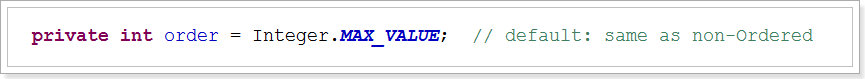
1. 定义Excel视图
   1. 导入poi依赖  
      
   2. 定义视图对象  
      SpringMVC提供的视图抽象类：  
        
      
2. 定义视图解析器  
     
   

多个视图解析器查找：

通过order属性定义查找的顺序。



默认顺序：



# 通用Mapper

问题：

Mapper接口有哪些常用的方法？

1. queryById
2. queryByWhere
3. queryListByWhere
4. queryAll
5. save
6. saveList
7. update（updateById）
8. delete（deleteById）

开发一个业务就需要写一套CRUD，能否写一个通用的CRUD实现？

如何实现通用的CRUD？

思路：

参数？

泛型？

第一个问题，SQL语句不能写死。

如何使SQL不写死？ 🡺

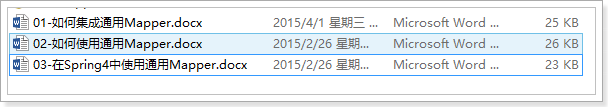
1. 传递，程序传递到Mapper.xml中
2. 动态生成
   1. 如何生成？ 🡺 POJO 生成 SQL语句
   2. 如何使用POJO生成SQL？ 使用反射实现

该在哪个阶段生成动态SQL？ 在Mapped Statement阶段生成。

生成的SQL该如何被执行器执行？ 拦截器或者插件。

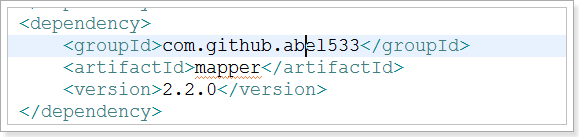
Service 🡺 Mapper接口 🡺 queryById 🡺 动态生成SQL 🡺 通过拦截器（插件） 🡺 传递到执行器中 🡺 执行器执行 🡺 数据

## 使用通用Mapper

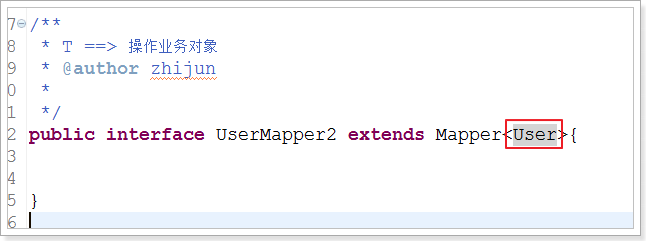
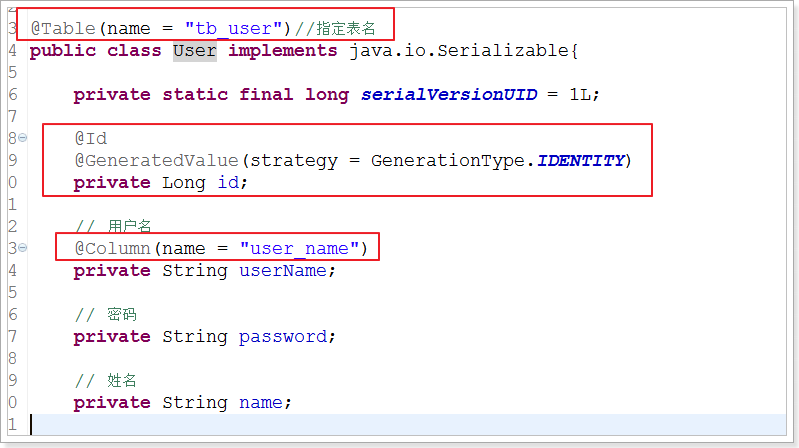


使用参考以上文档。

使用通用Mapper只需要4步：

1. 导入依赖  
   
2. 配置通用Mapper插件



1. 将Mapper接口继承通用Mapper  
   
2. 编写User中JPA注解  
   

使用通用Mapper之后省略了Mapper.xml的配置。

如果要使用通用Mapper之外的方法，需要自定义Mapper.xml扩展。

# RESTful Web Service

1、了解REST。

2、了解RESTful Web Service。

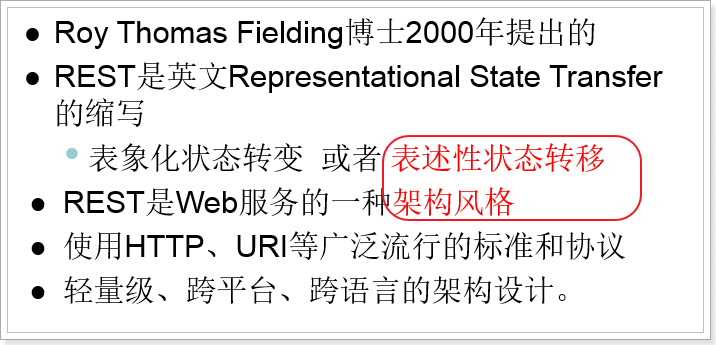
3、使用SpringMVC实现RESTful。

## REST

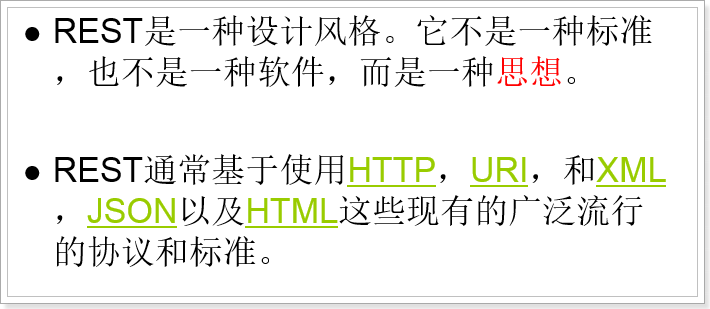
Roy Thomas Fielding

<http://en.wikipedia.org/wiki/Roy_Fielding>





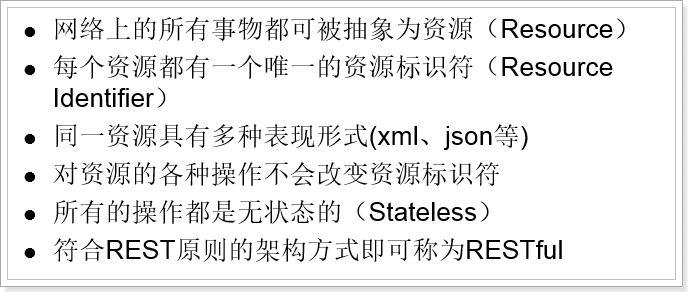
## REST到底是什么？



## RESTful是什么？



## REST 架构的主要原则

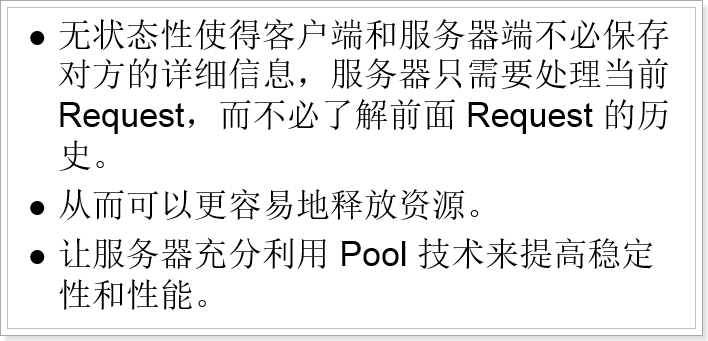


资源：一部电影、一部效果、一个URL等都可以称之为资源。

## URI和URL



## 无状态性



## 资源操作



对User实现CRUD：

GET <http://127.0.0.1/user/1> 获取用户数据

POST <http://127.0.0.1/user/save> form表单数据 新增用户数据

POST <http://127.0.0.1/user/update> 更新用户数据

POST <http://127.0.0.1/user/delete> 删除用户数据

REST的实现

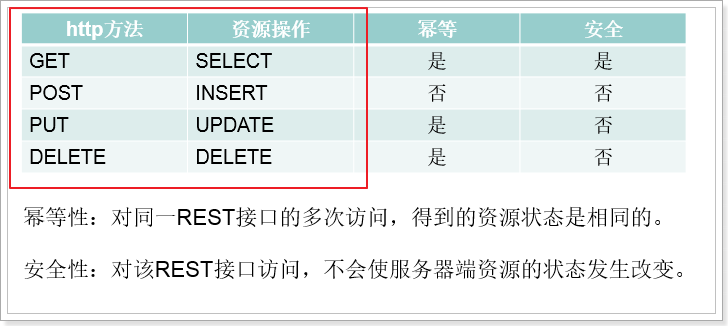
GET <http://127.0.0.1/user> 参数id=1 获取用户数据

POST http://127.0.0.1/user form表单数据 新增用户数据

PUT http://127.0.0.1/user 更新用户数据

DELETE http://127.0.0.1/user 删除用户数据

## REST接口定义

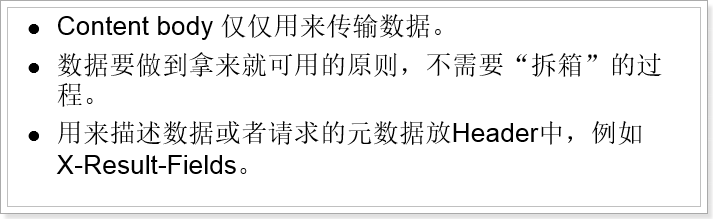


## 最佳实践：REST接口设计

接口：不是java中的接口，Web Service，系统提供给其他系统调用数据的地址。



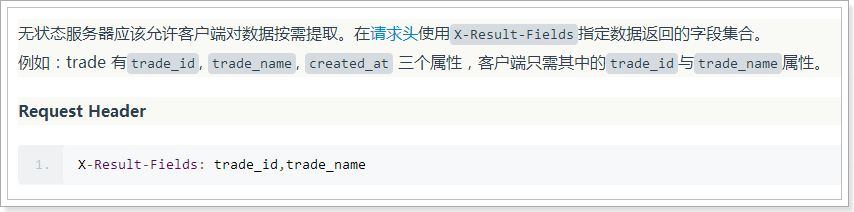
## 最佳实践：响应设计



### 响应示例



### 指定响应的属性字段



可以在http请求中添加自定义的头信息。

### http响应状态码



## SpringMVC实现RESTful服务



## 使用SpringMVC实现REST风格的接口

GET <http://127.0.0.1/user>/{id} 获取用户数据

POST http://127.0.0.1/user form表单数据 新增用户数据

PUT http://127.0.0.1/user 更新用户数据

DELETE http://127.0.0.1/user 删除用户数据

@Autowired

**private** UserService userService;

/\*\*

\* 根据ID查询用户数据

\* **@param** id

\* **@return**

\*/

@RequestMapping(value = "user/{id}", method = RequestMethod.***GET***)

@ResponseBody

**public** ResponseEntity<User> queryUserById(@PathVariable("id") Long id) {

**try** {

User user = **this**.userService.queryUserById(id);

**if** (user != **null**) {

**return** ResponseEntity.*ok*(user);

}

**return** ResponseEntity.*status*(HttpStatus.***NOT\_FOUND***).body(user);

} **catch** (Exception e) {

e.printStackTrace();

**return** ResponseEntity.*status*(HttpStatus.***INTERNAL\_SERVER\_ERROR***).body(**null**);

}

}

/\*\*

\* 新增用户数据

\* **@param** user

\* **@return**

\*/

@RequestMapping(value = "user", method = RequestMethod.***POST***)

**public** ResponseEntity<Void> saveUser(User user){

**try** {

**this**.userService.saveUser(user);

//返回201状态

**return** ResponseEntity.*status*(HttpStatus.***CREATED***).build();

} **catch** (Exception e) {

e.printStackTrace();

**return** ResponseEntity.*status*(HttpStatus.***INTERNAL\_SERVER\_ERROR***).build();

}

}

/\*\*

\* 更新数据

\* **@param** user

\* **@return**

\*/

@RequestMapping(value = "user", method = RequestMethod.***PUT***)

**public** ResponseEntity<Void> updateUser(User user){

**try** {

**this**.userService.updateUser(user);

//返回201状态

**return** ResponseEntity.*ok*(**null**);

} **catch** (Exception e) {

e.printStackTrace();

//500

**return** ResponseEntity.*status*(HttpStatus.***INTERNAL\_SERVER\_ERROR***).build();

}

}

/\*\*

\* 删除数据

\* **@param** user

\* **@return**

\*/

@RequestMapping(value = "user", method = RequestMethod.***DELETE***)

**public** ResponseEntity<Void> updateUser(@RequestParam("ids")Long[] ids){

**try** {

**this**.userService.deleteUser(ids);

//返回204状态

**return** ResponseEntity.*status*(HttpStatus.***NO\_CONTENT***).build();

} **catch** (Exception e) {

e.printStackTrace();

//500

**return** ResponseEntity.*status*(HttpStatus.***INTERNAL\_SERVER\_ERROR***).build();

}

}

# 小结

