REST风格详细介绍

REST简介

REST即表述性状态传递（英文：Representational State Transfer，简称REST）是Roy Fielding博士在2000年他的博士论文中提出来的一种软件架构风格。它是一种针对网络应用的设计和开发方式，可以降低开发的复杂性，提高系统的可伸缩性。

在三种主流的Web服务实现方案中，因为REST模式的Web服务与复杂的SOAP和XML-RPC对比来讲明显的更加简洁，越来越多的web服务开始采用REST风格设计和实现。例如，Amazon.com提供接近REST风格的Web服务进行图书查找；雅虎提供的Web服务也是REST风格的。

通过一个url来直观展示传统风格与REST风格的区别

传统风格：

当我们在浏览器上访问一些资源时，可以看到有些网页的url为

```
http: // localhost / students / selectById?id=1   （该地址表示查找id为1的students对象）
http: // localhost / students / saveStudent        （该地址表示保存students信息）
```

REST风格：

这两句url与上面两句功能是一样的

```
http: // localhosts / student / 1
http: // localhosts / student
```

通过这两种风格的对比，我们可以看到REST风格的一部分优点：

1. 可以隐藏资源的访问行为（如隐藏了selectById等），这样就无法通过地址得知对资源进行了哪种操作。
2. 可以明显的看到其书写简化了，不仅书写简化了，在开发时代码也可以简化。

RESTful风格的简单实现

传统风格在controller层

在学SpringMVC时，我们经常要在controller层编写特定的行为方法，用来与页面交互，这时通常需要使用@RequestMapping注解来给一个特定的行为进行地址映射

如下图：

```java
@Controller
@RequestMapping("/students")
public class StudentsController {

    @Resource( name = "studentsService")
    private StudentsService studentsService;

    @Resource(name = "objectsService")
    private ObjectsService objectsService;

    @RequestMapping("/findAll")
    public ModelAndView findAll(){
        ModelAndView modelAndView = new ModelAndView();
        List<Students> studentsList = studentsService.findAll();
        modelAndView.addObject( attributeName: "studentsList",studentsList);
        //设置视图
        modelAndView.setViewName("studentsList");
        System.out.println("controller层"+ studentsList);
        return modelAndView;
    }
```


当程序启动后，在浏览器地址栏输入： http://localhost:8080/students/findAll 就能够调用到这个方法，方法作用是查询所有的students

RESTful风格在controller层

```java
@Controller
@RequestMapping("/students")
public class StudentsController {

    @Resource( name = "studentsService")
    private StudentsService studentsService;

    @Resource(name = "objectsService")
    private ObjectsService objectsService;

    @RequestMapping(method = RequestMethod.GET)
    public ModelAndView findAll(){
        ModelAndView modelAndView = new ModelAndView();
        List<Students> studentsList = studentsService.findAll();
        modelAndView.addObject( attributeName: "studentsList",studentsList);
        //设置视图
        modelAndView.setViewName("studentsList");
        System.out.println("controller层"+ studentsList);
        return modelAndView;
    }
```


那么只需要将@RequestMapping中的映射地址改为一个 method = RequestMethod.GET 即可

这样只需要输入 http://localhost:8080/students 就可以访问到这个方法

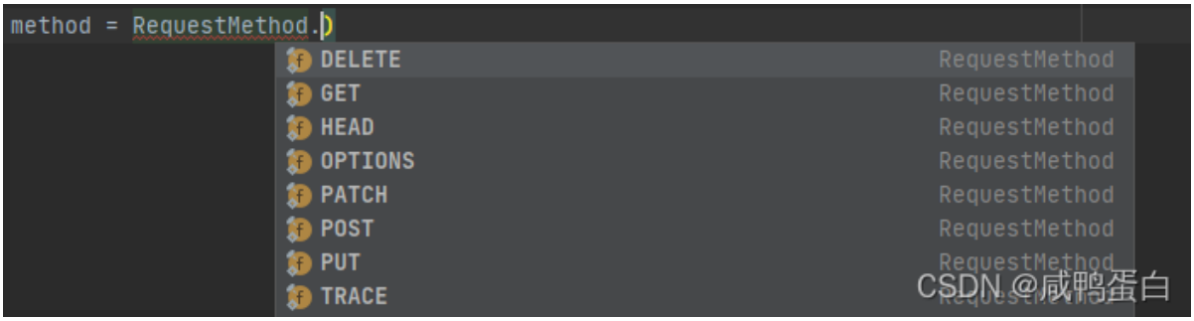
那么这个时候就会有人问

哎呀你这改了个啥呀？不就是@RequestMapping里面换了个东西嘛？还没我传统方法来的舒服呢！

别急，这只是单个例子，上面的代码过于繁杂，我新建一个controller来描述REST风格，请你耐心耐心看完~

RESTful风格的具体实现

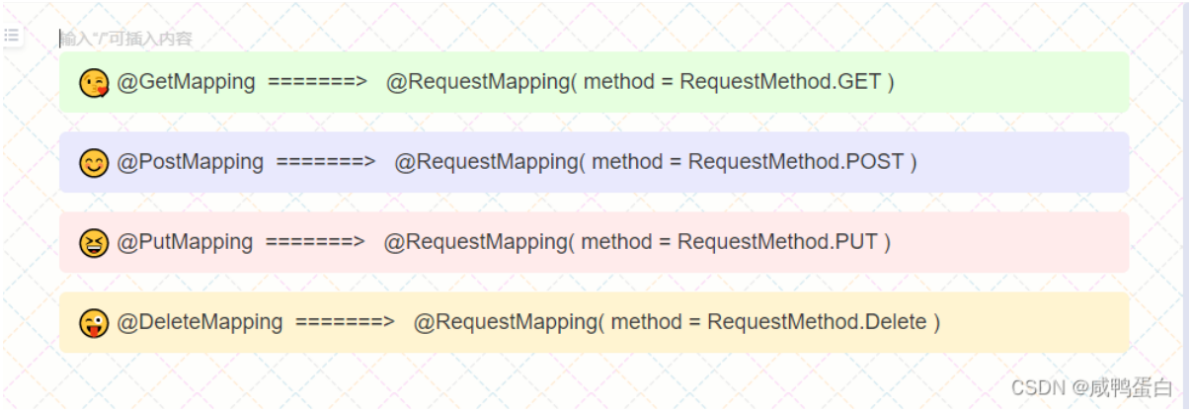
首先我们要知道，@RequestMapping注解中 method 方法可以写 8 种



而常用的四种

- GET （常用于查询）
- POST（常用于保存）
- PUT（常用于更新修改）
- DELETE（常用于删除）

当然，每次都写一遍@RequestMapping也很烦，直接写这些对应的注解




```
@GetMapping，处理的是get请求
@PostMapping，处理的是post请求
@PutMapping，处理的是put请求
@DeleteMapping，处理的是delete请求
```

首先我先把传统风格的一些增删改查方法写出来（简单的写）

传统风格增删改查：

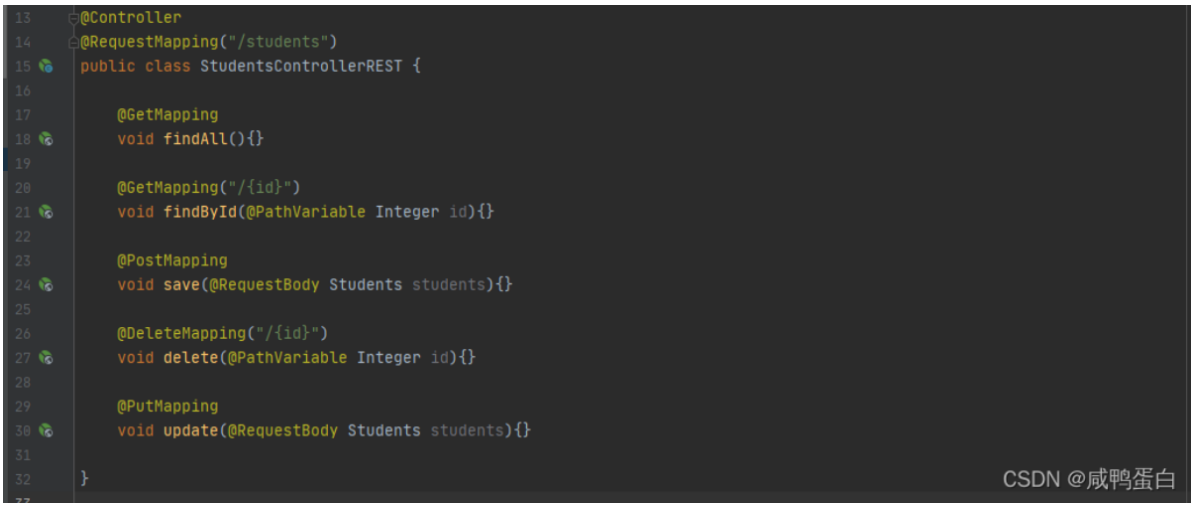
```java
@Controller
@RequestMapping("/students")
public class StudentsControllerChuanTong {

    @RequestMapping("/findAll")
    void findAll(){}

    @RequestMapping("/findById/{id}")
    void findById(@PathVariable Integer id){}

    @RequestMapping("/save")
    void save(Students students){}

    @RequestMapping("/delete/{id}")
    void delete(@PathVariable Integer id){}

    @RequestMapping("/update")
    void update(Students students){}

}
```


这样书写的话在浏览器中访问的url分别为

```
http://localhosst:8080/students/findAll
http://localhosst:8080/students/findById?id=1
http://localhosst:8080/students/save
http://localhosst:8080/students/delete?id=1
http://localhosst:8080/students/update
```

可以看出，其行为能一眼看出，这种方式虽然可以，但是不太好喔~

RESTful风格增删改查：



那么这样在浏览器访问的url为：

```
http://localhost:8080/students                    GET（查询所有学生）
http://localhost:8080/students/1                  GET（查询id为1的学生）
http://localhost:8080/students                    POST（保存学生信息）
http://localhost:8080/students/1                  DELETE（删除id为1的学生信息）
http://localhost:8080/students                    PUT（修改学生信息）
```

是不是简洁了很多？并且把行为都隐藏了，这就是RESTful风格，学废了嘛？

补充

风格与规则的区别：

风格是一种约定俗成的方式，这种约定并不是一定要遵守的，可以不去使用这种约定，即也可以使用传统风格。而规范呢是一种大家必须遵守的规则，你如果不按照这个规范来书写代码，那么就不被允许运行。 所以被叫做REST风格，而不是叫做REST规范。