URL设计规范——RESTful

> 前后端分离开发的项目中，前后端直接是在接口进行请求和响应的，后端向前端提供请求时必须对外暴力一个URL；URL的设计是不能随意的，需要遵循一定的设计规范——RESTful

1. RESTful的概述

RESTful是一种Web api的标准，也就是一种url设计风格/规范

RESTful对URL的设计有四点规范：

- 每个URL请求路径代表服务器上的唯一资源
- 使用不同的请求方式表示不同的操作
- 接口响应资源的表现形式采用JSON
- 前端（Android\ios\pc）通过无状态的HTTP协议与后端接口进行交互

2. RESTful的规范

1）每个URL请求路径代表服务器上的唯一资源

传统的URL设计风格的URL：

```
@RequestMapping("/delete")
public ResultVo delete(int bookId){
    System.out.println("------" + bookId);
    return new ResultVo(10000,"删除成功",null);
}
```

管理人员想要删除图书1，发起的请求是：

```
http://localhost:8080/book/delete?bookId=1
```

管理人员想要删除图书2，发起的请求是：

```
http://localhost:8080/book/delete?bookId=2
```

上面两个请求的url是

```
http://localhost:8080/book/delete
```

并不代表服务器上的唯一资源

而RESTful风格的设计是：

```
@RequestMapping("/delete/{bid}")
public ResultVo delete(@PathVariable("bid") int bookId){
    System.out.println("------" + bookId);
    return new ResultVo(10000,"删除成功",null);
}
```

@PathVariable：标注在请求参数之前，用于从请求路径中取值赋值给方法参数；占位符 {}

管理人员想要删除图书1，发起的请求是：

```
http://localhost:8080/book/delete/1
```

管理人员想要删除图书2，发起的请求是：

```
http://localhost:8080/book/delete/2
```

2）使用不同的请求方式表示不同的操作

SpringMVC对RESTful风格提供了很好的支持，在我们定义一个接口的url里，可以通过

```
@RequestMapping(value = "/{bid}",method = RequestMethod.DELETE)
```

形式指定请求方式，也可以通过指定请求方式的主键设定url

- @GetMapping 查询
- @PostMapping 添加
- @DeleteMapping 删除
- @PutMapping 修改

```
//根据id删除图书
@DeleteMapping (value = "/{bid}")
public ResultVo deleteBook(@PathVariable("bid") int bookId){
    System.out.println("------" + bookId);
    return new ResultVo(10000,"删除成功",null);
}
//添加图书
@PostMapping(value = "/add")
public ResultVo addBook(Book book){
    System.out.println("------" + book);
    return new ResultVo(10000,"添加成功",book);
```

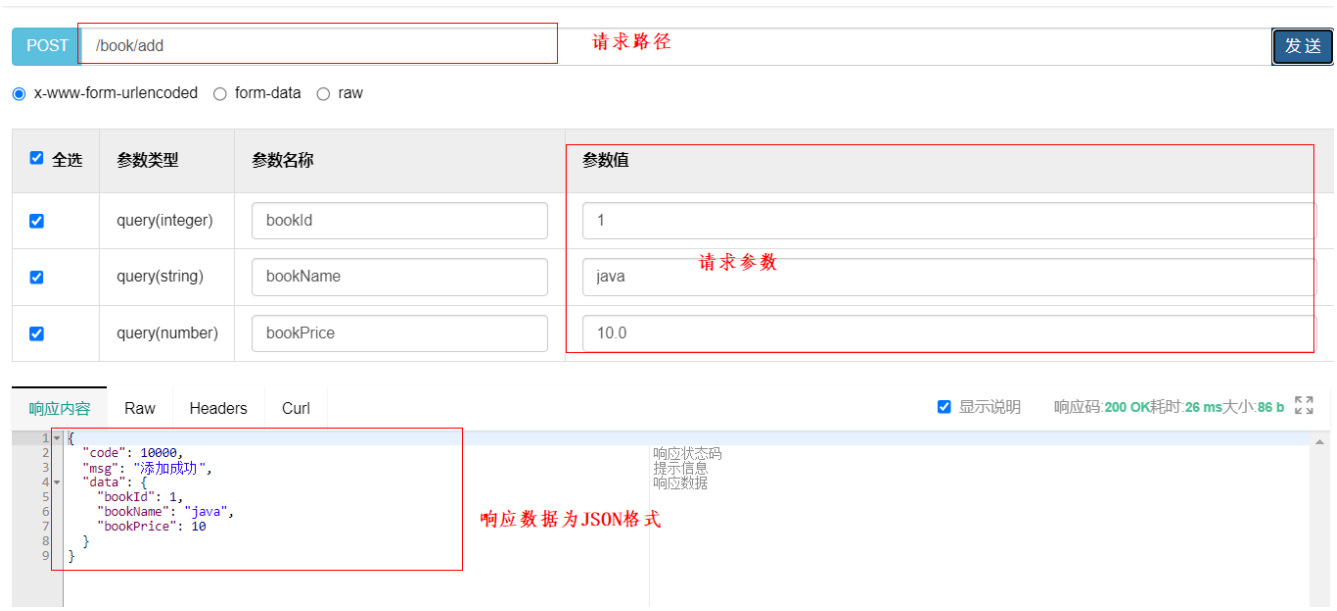
```java
}
//根据id更新图书
@PutMapping(value = "/{bid}")
public ResultVo updateBook(Book book){
    System.out.println("------" + book);
    return new ResultVo(10000,"修改成功",null);
}
//根据id查询图书
@GetMapping(value = "/{bid}")
public ResultVo getBook(@PathVariable("bid") int bookId){
    System.out.println("------" + bookId);
    return new ResultVo(10000,"查询成功",null);
}
```

3）接口响应资源的表现形式采用JSON

在控制器类或者每个接口方法添加@ResponseBody注解将返回数据格式化为JSON数据

或者直接在控制器类上添加@RestController注解声明控制器

```java
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Controller
@ResponseBody
public @interface RestController {
    @AliasFor(
        annotation = Controller.class
    )
    String value() default "";
}
```



4）前端（Android\ios\pc）通过无状态的HTTP协议与后端接口进行交互

前端请求后端接口等到响应后断开连接，下一次请求时，后端接口不知道前端之前是否请求过。