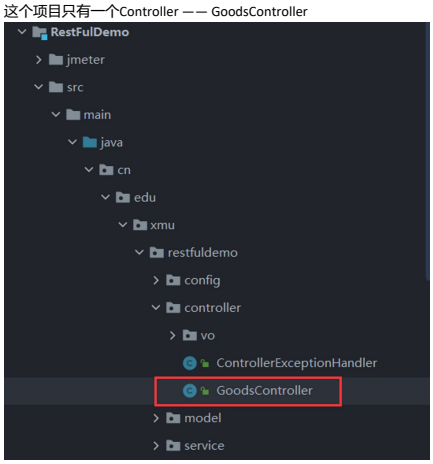
SpringMVC示例

2021年9月26日 21:33

以ResultfulDemo为例

这个项目只有一个Controller —— GoodsController



下面我们主要看这个类中的@RestController和五个Mapping注解

```
*/
@Api(value = "商品API", tags = "商品API")
@RestController /*Restful的Controller对象*/
@RequestMapping(value = "/goods", produces = "application/json;charset=UTF-8")
public class GoodsController {

```

在类之前有一个@RestController，这个注解的意思是它是一个RESTful的控制器，可以接受json的参数，也可以把值作为json返回回去

```
*/
@Api(value = "商品API", tags = "商品API")
@RestController /*Restful的Controller对象*/
@RequestMapping(value = "/goods", produces = "application/json;charset=UTF-8")
public class GoodsController {
```

在整个类之前定义了一个@RequestMapping，并没有指定method

这里是用value指定了一个**根的路径/goods**，说明这个类下所有的方法，如果不指定绝对路径，就会从/goods开始往后接相对路径

后面的produces，是在描述说类里的方法的返回的Response的Content类型是什么，因为我们定义了这是Restful风格的Controller，所以类型应该是json格式，编码charset是UTF-8

然后看类中的方法

```
@ApiOperation(value = "获得一个商品对象",  produces="application/json;charset=UTF-8")
@ApiImplicitParams({
        @ApiImplicitParam(paramType = "path", dataType = "Integer", name = "id", value ="商品对象id" ,required = true)
})
@ApiResponses({
})
@GetMapping("{id}")
public Object getGoodsById(@PathVariable("id") Integer id) {

    Goods goods = goodsService.findById(id);
    return ResponseUtil.ok(goods);
}
```

这是类中的第一个方法，方法前有一个**@GetMapping**，表明这个方法是映射到一个GET请求上

括号中是{id}，说明应该在url中包含一个id，结合类前面的根路径，如/goods/12 之类的请求会映射到这个方法上

id用花括号括起来，表明id是变量。因为这里的id是变量，所以方法参数中有一个**@PathVariable**，这个注解的作用是表明**要从路径上去拿到变量**，变量的名称就是路径中的id，然后把这个值赋给参数的id

另一个get的方法

```
@ApiOperation(value = "用商品名称搜索",  produces="application/json")
@ApiImplicitParams({
        @ApiImplicitParam(paramType = "query", dataType = "String", name = "name", value ="商品名称", required = true)
})
@ApiResponses({
})
@GetMapping("search")
public Object searchGoodsByName(@RequestParam String name) {

    Goods goods = goodsService.searchByName(name);
    return ResponseUtil.ok(goods);
}
```

这个@GetMapping表示只有/goods/search这样的url会映射到这个方法上

第三个方法

```java
@ApiOperation(value = "新建商品",  produces="application/json")
@ApiImplicitParams({
})
@ApiResponses({
})
@PostMapping("")
@ResponseStatus(HttpStatus.CREATED)
public Object createGood(@Validated @RequestBody GoodsVo goodsVo){
    Goods new_goods = goodsService.createGoods(goodsVo);
    return ResponseUtil.ok(new_goods);
}
```

第三个方法是一个post的方法，括号中没有内容，表示说这个createGoods方法是被映射到/goods上的
因为新建的商品的内容是在请求体RequestBody中传过来的，可以看到在参数中有一个注解@RequestBody，表明要从请求体中获得数据
后面是一个VO对象（Value Object），VO对象是用来接收从前端送过来的数据的，我们可以看一下GoodsVo的类

```java
/**
 * @Data
 * @ApiModel(description = "商品视图对象")
 */
public class GoodsVo {

    @NotBlank(message="商品名称不能为空")
    @ApiModelProperty(value = "商品名称")
    private String name;

    @ApiModelProperty(value = "商品描述")
    private String brief;

    @ApiModelProperty(value = "商品单位")
    private String unit;

    @ApiModelProperty(value = "商品类别Id")
    private Integer categoryId;

    @ApiModelProperty(value = "商品品牌Id")
    private Integer brandId;

    @ApiModelProperty(value = "商品可选规格")
    private List<Specification> specList;

    @ApiModelProperty(value = "商品规格")
    private List<ProductVo> productList;

    /**
     * 由Vo对象创建Goods对象
     * @return Goods对象
     */
    public Goods createGoods(){
        Goods goods = new Goods();
        goods.setName(this.name);
        goods.setBrief(this.brief);
        goods.setUnit(this.unit);
        goods.setSpecList(this.specList);
        goods.setBrandId(this.brandId);
        goods.setCategoryId(this.categoryId);
        return goods;
    }

}
```

这个VO对象中定义了从前端发送过来的一个新建的商品
中有哪些属性
可以看到有名称、描述、单位、可选规格等
还有一个是这些不同规格会组合出多少个产品，用一个List去存储，每一个规格的产品用一个ProductVo来表示

```java
public class ProductVo {

    @ApiModelProperty(value = "规格序号")
    @NotBlank
    @NotNull
    private String productSn;

    @NotBlank
    @NotNull
    @ApiModelProperty(value = "描述")
    private String desc;

    @ApiModelProperty(value = "促销价")
    @Min(0)
    private Integer counterPrice;

    @ApiModelProperty(value = "零售价")
    @Min(0)
    private Integer retailPrice;

    @ApiModelProperty(value = "重量(克)")
    @Min(0)
    private Integer weight;

    /**
     * 由Vo对象创建Product对象
     * @return Product对象
     */
    public Product createProduct(){
        Product product = new Product();
        product.setProductSn(this.productSn);
        product.setCounterPrice(this.counterPrice);
        product.setRetailPrice(this.retailPrice);
        product.setWeight(this.weight);
        return product;
    }
}
```

每一个ProductVo对象有一个序号
序号来自于前面的可选规格的组合，按照一定规格组成序号
然后是规格的描述、促销价、零售价、重量

所以Controller中的createGood方法会把一个商品的可选规格、以及由商品产生的所有规格的组合建成VO对象，然后传到方法中
在这个createGood方法前还有一个注解@ResponseStatus(HttpStatus.CREATED)，这个是用来指定Http的返回码
我们知道POST方法如果执行成功应该返回201而不是200，在前面的get方法中，不标出返回码，默认都会返回200
对于post的方法，我们专门指定它的返回码是201

```java
@ApiOperation(value = "修改商品",  produces="application/json")
@ApiImplicitParams({
})
@ApiResponses({
        @ApiResponse(code = 0, message = "成功"),
})
@PutMapping("{id}")
public Object modiGood(@PathVariable Integer id, @RequestBody GoodsVo goodsVo){
    goodsService.modifyGoods(id, goodsVo);
    return ResponseUtil.ok();
}
```

第四个方法是put方法，put方法是用来修改对象的
@PutMapping中指定PUT /goods/{id}的url会对应到这个方法上，我们注意到参数中的**@PathVariable没有指定参数名称**，因为如果参数名和path中的名称是一样的，会自动对应过去
同时，我们知道PUT修改的对象的属性是放在Request Body里的，所以我们用一个GoodsVo来接收从前端送过来的一个对象的属性

```java
@ApiOperation(value = "删除商品",  produces="application/json")
@ApiImplicitParams({
        @ApiImplicitParam(paramType = "path", dataType = "Integer", name = "id", value ="商品对象id" ,required = true)
})
@ApiResponses({
})
@DeleteMapping("{id}")
public Object delGoods(@PathVariable("id") Integer id) {

    goodsService.delGoods(id);
    return ResponseUtil.ok();
}
```

最后是一个DELETE方法，这里的方法是用来删除一个商品