编写commons模块

继续编写commons模块的dto和实体类

order.dto包下

```
@ApiModel(value = "新增订单的DTO")
@Data
public class OrderAddDTO implements Serializable {
    @ApiModelProperty(value = "用户id",name="userId",example = "UU100")
    private String userId;
    @ApiModelProperty(value = "商品编号",name="commodityCode",example = "PC100")
    private String commodityCode;
    @ApiModelProperty(value = "购买数量",name="count",example = "5")
    private Integer count;
    @ApiModelProperty(value = "总金额",name="money",example = "500")
    private Integer money;

}
```

order.model包下

```
@Data
public class Order implements Serializable
{

    private Integer id;
    private String userId;
    private String commodityCode;
    private Integer count;
    private Integer money;

}
```

stock.dto库存

```
@ApiModel(value = "商品减库存DTO")
@Data
public class StockReduceCountDTO implements
Serializable {

    @ApiModelProperty(value = "商品编
号",name="commodityCode",example = "PC100")
    private String commodityCode;
    @ApiModelProperty(value = "减库存
数",name="reduceCount",example = "5")
    private Integer reduceCount;
}
```

stock.model

```java
@Data
public class Stock implements Serializable
{

    private Integer id;
    private String commodityCode;
    private Integer reduceCount;
}
```

创建cn.tedu.csmall.commons.restful包

这个包下创建ResponseCode枚举

这个枚举是定义常见响应状态码的封装

```java
/**
 * 错误代码枚举类型
 */
public enum ResponseCode {

    OK(200),
    BAD_REQUEST(400),
    UNAUTHORIZED(401),
    FORBIDDEN(403),
    NOT_FOUND(404),
    NOT_ACCEPTABLE(406),
    CONFLICT(409),
    INTERNAL_SERVER_ERROR(500);

    private Integer value;

    ResponseCode(Integer value) {
        this.value = value;
    }

    public Integer getValue() {
        return value;
    }

}
```

下面要创建自定义异常类

和统一异常处理类

创建包cn.tedu.csmall.commons.exception

包中创类CoolSharkServiceException

```
@Data
@EqualsAndHashCode(callSuper = false)
public class CoolSharkServiceException
extends RuntimeException {

    private ResponseCode responseCode;

    public
CoolSharkServiceException(ResponseCode
responseCode, String message) {
        super(message);
        setResponseCode(responseCode);
    }

}
```

下面是SpringMvc提供的统一异常处理功能

创建包cn.tedu.csmall.commons.exception.handler

包中创建类GlobalControllerExceptionHandler

```
@RestControllerAdvice
@Slf4j
public class
GlobalControllerExceptionHandler {

    /**
     * 处理业务异常
     */

```

```
 @ExceptionHandler({CoolSharkServiceException.class})
    public JsonResult<Void> handleCoolSharkServiceException(CoolSharkServiceException e) {
        log.debug("出现业务异常，业务错误码={}，描述文本={}", e.getResponseCode().getValue(), e.getMessage());
        e.printStackTrace();
        JsonResult<Void> result = JsonResult.failed(e);
        log.debug("即将返回：{}", result);
        return result;
    }

    /**
     * 处理绑定异常（通过Validation框架验证请求参数时的异常）
     */
    @ExceptionHandler(BindException.class)
    public JsonResult<Void> handleBindException(BindException e) {
        log.debug("验证请求数据时出现异常：{}", e.getClass().getName());
        e.printStackTrace();
        String message = e.getBindingResult().getFieldError().getDefaultMessage();
        JsonResult<Void> result = 
JsonResult.failed(ResponseCode.BAD_REQUEST, 
message);
        log.debug("即将返回：{}", result);
        return result;
    }

    /**
     * 处理系统（其它）异常
     */
    @ExceptionHandler({Throwable.class})
    public JsonResult<Void> 
handleSystemError(Throwable e) {
        log.debug("出现系统异常，异常类型={}，描
述文本={}", e.getClass().getName(), 
e.getMessage());
        e.printStackTrace();
        JsonResult<Void> result = 
JsonResult.failed(ResponseCode.INTERNAL_SER
VER_ERROR, e);
        log.debug("即将返回：{}", result);
        return result;
    }
}
```

赋值过来报错没关系

我们再创建JsonResult类在restful包下

```
@Data
```

```
public class JsonResult<T> implements
Serializable {

    /**
     * 状态码
     */
    @ApiModelProperty(value = "业务状态码",
position = 1, example = "200, 400, 401,
403, 404, 409, 500")
    private Integer state;
    /**
     * 消息
     */
    @ApiModelProperty(value = "业务消息",
position = 2, example = "登录失败！密码错误！")
    private String message;
    /**
     * 数据
     */
    @ApiModelProperty(value = "业务数据",
position = 3)
    private T data;

    /**
     * 创建响应结果对象，表示"成功"，不封装其它任何
数据
     * @return 响应结果对象
     */
    public static JsonResult<Void> ok() {
        return ok("OK");
    }

    public static JsonResult ok(String
message){
        JsonResult jsonResult=new
JsonResult();

 jsonResult.setState(ResponseCode.OK.getVal
ue());
        jsonResult.setMessage(message);
        jsonResult.setData(null);
        return jsonResult;
    }
    /**
     * 创建响应结果对象，表示"成功"，且封装客户端期
望响应的数据
     * @param data 客户端期望响应的数据
     * @return 响应结果对象
     */
    public static <T> JsonResult<T>
ok(String message,T data) {
        JsonResult<T> jsonResult = new
JsonResult<>();

 jsonResult.setState(ResponseCode.OK.getVal
ue());
        jsonResult.setData(data);
        return jsonResult;
    }
    /**
```

```
     * 创建响应结果对象，表示"失败"，且封装"失败"的描述
     *
     * @param e CoolSharkServiceException异常对象
     * @return 响应结果对象
     */
    public static JsonResult<Void> failed(CoolSharkServiceException e) {
        return failed(e.getResponseCode(), e);
    }

    /**
     * 创建响应结果对象，表示"失败"，且封装"失败"的描述
     *
     * @param responseCode "失败"的状态码
     * @param e            "失败"时抛出的异常对象
     * @return 响应结果对象
     */
    public static JsonResult<Void> failed(ResponseCode responseCode, Throwable e) {
        return failed(responseCode, e.getMessage());
    }

    /**
```

```
     * 创建响应结果对象，表示"失败"，且封装"失败"的描述
     *
     * @param responseCode "失败"的状态码
     * @param message      "失败"的描述文本
     * @return 响应结果对象
     */
    public static JsonResult<Void> failed(ResponseCode responseCode, String message) {
        JsonResult<Void> jsonResult = new JsonResult<>();
        jsonResult.setState(responseCode.getValue());
        jsonResult.setMessage(message);
        return jsonResult;
    }

}
```

commons项目到此结束

创建Business项目

我们要完成的业务是订单的创建

需要一个项目来触发这个业务

它也是当前父项目的子项目,要父子相认

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project
xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema
-instance"

xsi:schemaLocation="http://maven.apache.org
/POM/4.0.0
https://maven.apache.org/xsd/maven-
4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>cn.tedu</groupId>
        <artifactId>csmall</artifactId>
        <version>0.0.1-SNAPSHOT</version>
    </parent>
    <groupId>cn.tedu</groupId>
    <artifactId>csmall-
business</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>csmall-business</name>
    <description>Demo project for Spring
Boot</description>
    <dependencies>
        <dependency>

 <groupId>org.springframework.boot</groupId
>
            <artifactId>spring-boot-
starter-web</artifactId>
```

```xml
        </dependency>
        <dependency>

 <groupId>com.github.xiaoymin</groupId>
            <artifactId>knife4j-spring-
boot-starter</artifactId>
        </dependency>
        <dependency>
            <groupId>cn.tedu</groupId>
            <artifactId>csmall-
commons</artifactId>
            <version>0.0.1-
SNAPSHOT</version>
        </dependency>
    </dependencies>
</project>
```

编写application.yml的配置

```yaml
server:
  port: 20000
#公共配置
mybatis:
  configuration:
    cache-enabled: false
    map-underscore-to-camel-case: true
    log-impl:
org.apache.ibatis.logging.stdout.StdOutImpl
knife4j:
  # 开启增强配置
  enable: true
  # 生产环境屏蔽，开启将禁止访问在线API文档
  production: false
  # Basic认证功能，即是否需要通过用户名、密码验证后才可以访问在线API文档
  basic:
    # 是否开启Basic认证
    enable: false
    # 用户名，如果开启Basic认证却未配置用户名与密码，默认是：admin/123321
    username: root
    # 密码
    password: root
spring:
  profiles:
    active: dev
```

创建application-dev.yml暂时为空

删除测试test文件夹

创建config包

包中编写类CommonConfiguration

```
// 因为我们要套用commons项目中的异常相关支持,所以要讲异常包扫描,讲对象保存到当前项目的spring容器中
@Configuration  // 当前类是一个Spring配置类
@ComponentScan(basePackages =
"cn.tedu.csmall.commons.exception")
public class CommonConfiguration {
}
```

到此为止配置完成

开始创建业务逻辑

创建service包

```
public interface IBusinessService {
    // 编写购买商品生成订单的方法声明
    void buy();
}
```

创建service.impl包

包中创建业务逻辑层实现类BusinessServiceImpl

```
@Service
@Slf4j
public class BusinessServiceImpl implements
IBusinessService {

    @Override
    public void buy() {
        // 模拟一个下单过程
        OrderAddDTO orderAddDTO=new
OrderAddDTO();

 orderAddDTO.setCommodityCode("PC100");
        orderAddDTO.setUserId("UU100");
        orderAddDTO.setCount(5);
        orderAddDTO.setMoney(500);
        log.info("要新增的订单信息为:
{}",orderAddDTO);
```

```
    }
}
```

创建控制层包controller

BusinessController类代码如下

```
@RestController
@RequestMapping("/base/business")
@Api(tags = "购买业务开始模块")
public class BusinessController {

    @Autowired
    private IBusinessService
businessService;

    @PostMapping("/buy") //
localhost:20000/base/business/buy
    @ApiOperation("发起购买")
    public JsonResult buy(){
        //  调用业务逻辑层方法,业务逻辑层方法没有
返回值
        businessService.buy();
        return JsonResult.ok("购买完成");
    }
}
```

启动当前business项目

测试路径

http://localhost:20000/doc.html

点击调试发送请求,观察输出结果和idea控制台