

## 什么Nacos

Nacos是Spring Cloud Alibaba提供的一个软件

这个软件主要具有注册中心和配置中心的功能

我们先学习它注册中心的功能

微服务中所有项目都必须注册到注册中心才能成为微服务的一部分

注册中心和企业中的人力资源管理部门有相似

注册

## Nacos的启动

我们要启动Nacos必须保证当前系统配置了java环境变量

简单来说就是要环境变量中,有**JAVA\_HOME**的配置,指向安装jdk的路径

确定了支持java后,可以通过下面路径下载

<https://github.com/alibaba/nacos/releases/download/1.4.3/nacos-server-1.4.3.zip>

## 安装启动Nacos

将下载好的Nacos压缩包解压

将压缩包解压(注意不要有中文路径或空格)

打开解压得到的文件夹后打开bin目录会有如下内容

解压后的nacos

1. cmd结尾的文件是windows版本的

2. sh结尾的文件是linux和mac版本的

startup是启动文件,shutdown是停止文件

Windows下启动Nacos不能直接双击cmd文件

需要进入dos命令运行

在当前资源管理器地址栏输入cmd

```
G:\pgm\nacos\bin>startup.cmd -m standalone
```

- -m是设置启动方式参数
- standalone翻译为标准的孤独的
- 意思是单机模式标准运行
- 运行成功默认占用8848端口,并且在代码中提示

如果不输入standalone运行会失败

```
startup.cmd -m standalone
```

验证Nacos的运行状态

打开浏览器输入http://localhost:8848/nacos

该页面运行成功,表示安装成功

如果是首次访问,会出现这个界面

登录系统

1. 用户名:nacos
2. 密码:nacos

登录之后可以进入后台列表

不能关闭启动nacos的dos窗口

我们要让我们编写的项目注册到Nacos,才能真正是微服务项目

## csmall项目案例

### 业务概述

这个项目作为学习微服务组件使用

我们需要完成一个添加订单的业务

模拟用户选中购物车中商品并且确定了数量的情况下点击提交订单时后端的操作

1. 减少选中商品sku的库存数
2. 删除用户再购物车中勾选的商品
3. 生成订单,将订单信息保存到数据库

上面三个步骤分别由3个模块完成

1. 库存模块:减少库存
2. 购物车模块:删除购物车信息
3. 订单模块:新增订单

## 创建csmall父项目

创建项目名称csmall

首先删除项目的src目录,因为我们使用不到

其次,pom文件有大量配置

直接从提供给大家的完整版中复制

其中的内容,我们会随着后面学习,逐一给大家讲解

最终的pom文件内容为

```
<?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>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-parent</artifactId>
<version>2.5.4</version>
<relativePath/> <!-- lookup parent from repository -->
  </parent>
<groupId>cn.celinf</groupId>
<artifactId>csmall</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>csmall</name>
<description>????????</description>
<packaging>pom</packaging>
<modules>
  <!--      <module>csmall-business</module>-->
  <!--      <module>csmall-commons</module>-->
  <!--      <module>csmall-cart</module>-->
  <!--      <module>csmall-order</module>-->
  <!--      <module>csmall-stock</module>-->
  </modules>
<properties>
  <java.version>1.8</java.version>
<spring-cloud.version>2020.0.3</Spring-cloud.version>
<spring-cloud-alibaba.version>2.2.2.RELEASE</spring-cloud-
alibaba.version>
<spring-boot.version>2.5.4</spring-boot.version>
<spring-boot-configuration-processor.version>
  2.3.0.RELEASE
</spring-boot-configuration-processor.version>
<spring-security-jwt.version>1.0.10.RELEASE</spring-security-
jwt.version>
<MyBatis-spring-boot.version>2.2.0</mybatis-spring-
boot.version>
<mybaits-plus.version>3.4.1</mybaits-plus.version>
<pagehelper-spring-boot.version>1.4.0</pagehelper-spring-
boot.version>
```

```
<MySQL.version>8.0.26</mysql.version>
<lombok.version>1.18.20</lombok.version>
<knife4j-spring-boot.version>2.0.9</knife4j-spring-
boot.version>
<spring-rabbit-test.version>2.3.10</spring-rabbit-
test.version>
<spring-security-test.version>5.5.2</spring-security-
test.version>
<fastjson.version>1.2.45</fastjson.version>
<druid.version>1.1.20</druid.version>
<jjwt.version>0.9.0</jjwt.version>
<seata-server.version>1.4.2</seata-server.version>
</properties>
<dependencies>
  <dependency>
    <groupId>org.projectlombok</groupId>
<artifactId>lombok</artifactId>
</dependency>
</dependencies>
<!-- ???? -->
<dependencyManagement>
  <dependencies>
    <!--seata-all-->
    <dependency>
      <groupId>io.seata</groupId>
<artifactId>seata-all</artifactId>
<version>${seata-server.version}</version>
</dependency>
<!-- Lombok -->
<dependency>
  <groupId>org.projectlombok</groupId>
<artifactId>lombok</artifactId>
<version>${lombok.version}</version>
</dependency>
<!-- MySQL -->
<dependency>
  <groupId>mysql</groupId>
<artifactId>mysql-connector-java</artifactId>
<version>${mysql.version}</version>
```

```
<scope>runtime</scope>
</dependency>
<!-- Alibaba Druid -->
<dependency>
  <groupId>com.alibaba</groupId>
<artifactId>druid</artifactId>
<version>${druid.version}</version>
</dependency>
<!-- MyBatis Spring Boot??????MyBatis?? -->
<dependency>
  <groupId>org.mybatis.spring.boot</groupId>
<artifactId>mybatis-spring-boot-starter</artifactId>
<version>${mybatis-spring-boot.version}</version>
</dependency>
<!-- MyBatis Plus Spring Boot?MyBatis?? -->
<dependency>
  <groupId>com.baomidou</groupId>
<artifactId>mybatis-plus-boot-starter</artifactId>
<version>${mybaits-plus.version}</version>
</dependency>
<!-- MyBatis Plus Generator?????? -->
<dependency>
  <groupId>com.baomidou</groupId>
<artifactId>mybatis-plus-generator</artifactId>
<version>${mybaits-plus.version}</version>
</dependency>
<!-- PageHelper Spring Boot?MyBatis?? -->
<dependency>
  <groupId>com.github.pagehelper</groupId>
<artifactId>pagehelper-spring-boot-starter</artifactId>
<version>${pagehelper-spring-boot.version}</version>
</dependency>
<!-- Spring Boot????? -->
<dependency>
  <groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter</artifactId>
<version>${spring-boot.version}</version>
</dependency>
<!-- Spring Boot Web?WEB?? -->
```

```
<dependency>
  <groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
<version>${spring-boot.version}</version>
</dependency>
<!-- Spring Boot Freemarker?MyBaits Plus Generator???? -->
<dependency>
  <groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-freemarker</artifactId>
<version>${spring-boot.version}</version>
</dependency>
<!-- Spring Boot Validation?????? -->
<dependency>
  <groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-validation</artifactId>
<version>${spring-boot.version}</version>
</dependency>
<!-- Spring Boot Security????? -->
<dependency>
  <groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-security</artifactId>
<version>${spring-boot.version}</version>
</dependency>
<!-- Spring Boot Oauth2????? -->
<dependency>
  <groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-oauth2-client</artifactId>
<version>${spring-boot.version}</version>
</dependency>
<!-- Spring Boot????? -->
<dependency>
  <groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-configuration-processor</artifactId>
<version>${spring-boot-configuration-
processor.version}</version>
</dependency>
<!-- Spring Security JWT -->
<dependency>
  <groupId>org.springframework.security</groupId>
```

```
<artifactId>spring-security-jwt</artifactId>
<version>${spring-security-jwt.version}</version>
</dependency>
<!-- Knife4j Spring Boot???API -->
<dependency>
  <groupId>com.github.xiaoymin</groupId>
  <artifactId>knife4j-spring-boot-starter</artifactId>
  <version>${knife4j-spring-boot.version}</version>
</dependency>
<!-- Spring Boot Data Redis??? -->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-redis</artifactId>
  <version>${spring-boot.version}</version>
</dependency>
<!-- Spring Boot Data MongoDB??? -->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-mongodb</artifactId>
  <version>${spring-boot.version}</version>
</dependency>
<!-- Spring Boot Data Elasticsearch????? -->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-
elasticsearch</artifactId>
  <version>${spring-boot.version}</version>
</dependency>
<!-- Spring Boot AMQP????? -->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-amqp</artifactId>
  <version>${spring-boot.version}</version>
</dependency>
<!-- Spring Boot Actuator????? -->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
  <version>${spring-boot.version}</version>
```

```
</dependency>
<!-- Spring Cloud?? -->
<dependency>
  <groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-dependencies</artifactId>
<version>${spring-cloud.version}</version>
<type>pom</type>
<scope>import</scope>
</dependency>
<!-- Spring Cloud Alibaba -->
<dependency>
  <groupId>com.alibaba.cloud</groupId>
<artifactId>spring-cloud-alibaba-dependencies</artifactId>
<version>${spring-cloud-alibaba.version}</version>
<type>pom</type>
<scope>import</scope>
</dependency>
<!-- Alibaba FastJson -->
<dependency>
  <groupId>com.alibaba</groupId>
<artifactId>fastjson</artifactId>
<version>${fastjson.version}</version>
</dependency>
<!-- JWT -->
<dependency>
  <groupId>io.jsonwebtoken</groupId>
<artifactId>jjwt</artifactId>
<version>${jjwt.version}</version>
</dependency>
<!-- Spring Boot Test??? -->
<dependency>
  <groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-test</artifactId>
<version>${spring-boot.version}</version>
<scope>test</scope>
</dependency>
<!-- Spring Rabbit Test??????? -->
<dependency>
  <groupId>org.springframework.amqp</groupId>
```

```
<artifactId>spring-rabbit-test</artifactId>
<version>${spring-rabbit-test.version}</version>
<scope>test</scope>
</dependency>
<!-- Spring Security Test?Security?? -->
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-test</artifactId>
  <version>${spring-security-test.version}</version>
  <scope>test</scope>
</dependency>
<!-- seata??springboot-->
<dependency>
  <groupId>io.seata</groupId>
  <artifactId>seata-spring-boot-starter</artifactId>
  <version>${seata-server.version}</version>
</dependency>
</dependencies>
</dependencyManagement>
</project>
```

## 创建通用项目commons

创建好之后

1. 删除test测试文件夹
2. 删除resources目录
3. 删除SpringBoot启动类

这些都用到

编写父项目的module配置

```
<module>csmall-commons</module>
```

在修改子项目pom文件

```
<?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.celinf</groupId>
<artifactId>csmall</artifactId>
<version>0.0.1-SNAPSHOT</version>
</parent>
<groupId>cn.celinf</groupId>
<artifactId>csmall-commons</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>csmall-commons</name>
<description>Demo project for Spring Boot</description>
<dependencies>
  <!--??api??-->
  <dependency>
    <groupId>com.github.xiaoymin</groupId>
<artifactId>knife4j-spring-boot-starter</artifactId>
</dependency>
<!-- Spring Boot Web?WEB?? -->
<dependency>
  <groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
<exclusions>
  <exclusion>
    <groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter</artifactId>
</exclusion>
<exclusion>
  <groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-json</artifactId>
</exclusion>
<exclusion>
  <groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-tomcat</artifactId>
</exclusion>
</exclusions>
</dependency>
```

```
</dependencies>  
</project>
```

## 创建包

cn.celinf.csmall.commons.pojo.cart.dto包

### 包中创建类CartAddDTO

```
//?????????????????  
@ApiModelProperty("?????DTO")  
@Data  
public class CartAddDTO implements Serializable {  
    @ApiModelProperty(value = "????",name = "commodityCode",example = "PC100")  
    private String commodityCode;  
    @ApiModelProperty(value = "????",name = "price",example = "188")  
    private Integer price;  
    @ApiModelProperty(value = "????",name = "count",example = "5")  
    private Integer count;  
    @ApiModelProperty(value = "??ID",name = "userId",example = "UU100")  
    private String userId;  
}
```

## 创建实体类

创建包 pojo.cart.entity 包中创建类Cart

```
@Data  
public class Cart implements Serializable {  
    private Integer id;  
    // ????  
    private String commodityCode;  
    // ??  
    private Integer price;  
    // ??
```

```
private Integer count;
// ??id
private Integer userId;
}
```

下面创建订单模块需要的类

pojo.order.dto.OrderAddDTO

```
@ApiModelProperty("?????DTO")
@Data
public class OrderAddDTO implements Serializable {
    @ApiModelProperty(value = "??id",name="userId",example = "
UU100")
    private String userId;
    @ApiModelProperty(value = "????",name="commodityCode",example = "PC100")
    private String commodityCode;
    @ApiModelProperty(value = "????",name="count",example = "5
")
    private Integer count;
    @ApiModelProperty(value = "???",name="money",example = "50
")
    private Integer money;
}
```

pojo.order.entity.Order

```
@Data
public class Order implements Serializable {
    private Integer id;
    private String userId;
    private String commodityCode;
    private Integer count;
    private Integer money;
}
```

最后是库存相关的类

## pojo.stock.dto.StockReduceCountDTO

```
@ApiModelProperty("??????DTO")
@Data
public class StockReduceCountDTO implements Serializable {
    @ApiModelProperty(value = "????",name="commodityCode",example = "PC100")
    private String commodityCode;
    @ApiModelProperty(value = "????",name="reduceCount",example = "5")
    private Integer reduceCount;
}
```

## pojo.stock.entity.Stock

```
@Data
public class Stock implements Serializable {
    private Integer id;
    private String commodityCode;
    private Integer reduceCount;
}
```

## 创建异常相关类

commons项目除了实体类之外

项目使用到的所有异常相关类,也统一编写在这个类中

创建cn.celinf.csmall.commons.restful包

首先创建常见响应状态码的枚举

```
// ???????
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;
    }
}
```

下面创建自定义异常类

创建包cn.celinf.csmall.common.exception

包中创建类CoolSharkServiceException

```
@Data
@EqualsAndHashCode(callSuper = false)
public class CoolSharkServiceException extends RuntimeException {
    private ResponseCode responseCode;

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

将restful包中用于控制器返回的JsonResult类复制

```
@Data
public class JsonResult<T> implements Serializable {
    /**
     * ???
     */
}
```

```
@ApiModelProperty(value = "?????", position = 1, example =
"200, 400, 401, 403, 404, 409, 500")
private Integer state;
/**
 * ??
 */
@ApiModelProperty(value = "????", position = 2, example =
"????????????")
private String message;
/**
 * ??
 */
@ApiModelProperty(value = "????", position = 3)
private T data;

/**
 * ??????????????"?"?????????????
 * @return ??????
 */
public static JsonResult<Void> ok() {
    return ok("OK");
}

public static JsonResult ok(String message){
    JsonResult jsonResult=new JsonResult();
    jsonResult.setState(ResponseCode.OK.getValue());
    jsonResult.setMessage(message);
    jsonResult.setData(null);
    return jsonResult;
}
/**
 * ??????????????"?"?????????????????
 * @param data ??????????????
 * @return ??????
 */
public static <T> JsonResult<T> ok(String message,T data) {
    JsonResult<T> jsonResult = new JsonResult<>();
    jsonResult.setState(ResponseCode.OK.getValue());
    jsonResult.setData(data);
}
```

```
    return JsonResult;
}
/**
 * ??????????????"?"?????"?"????
 *
 * @param e CoolSharkServiceException????
 * @return ??????
 */
public static JsonResult<Void> failed(CoolSharkServiceException e) {
    return failed(e.getResponseCode(), e);
}
/**
 * ??????????????"?"?????"?"????
 *
 * @param responseCode "??"????
 * @param e "??"????????????
 * @return ??????
 */
public static JsonResult<Void> failed(ResponseCode responseCode, Throwable e) {
    return failed(responseCode, e.getMessage());
}
/**
 * ??????????????"?"?????"?"????
 *
 * @param responseCode "??"????
 * @param message "??"??????
 * @return ??????
 */
public static JsonResult<Void> failed(ResponseCode responseCode, String message) {
    JsonResult<Void> JsonResult = new JsonResult<>();
    JsonResult.setState(responseCode.getValue());
    JsonResult.setMessage(message);
    return JsonResult;
}
}
```

## exception包下最后创建复制统一异常处理类

```
/**
 * ???????
 */
@RestControllerAdvice
@Slf4j
public class GlobalControllerExceptionHandler {

    /**
     * ???????
     */
    @ExceptionHandler({CoolSharkServiceException.class})
    public JsonResult<Void> handleCoolSharkServiceException(CoolSharkServiceException e) {
        log.debug("?????????????={}?????={}", e.getResponseCode().getValue(), e.getMessage());
        e.printStackTrace();
        JsonResult<Void> result = JsonResult.failed(e);
        log.debug("?????{}", result);
        return result;
    }

    /**
     * ?????????Validation????????????????
     */
    @ExceptionHandler(BindException.class)
    public JsonResult<Void> handleBindException(BindException e)
    {
        log.debug("?????????????{}", e.getClass().getName());
        e.printStackTrace();
        String message = e.getBindingResult().getFieldError().getDefaultMessage();
        JsonResult<Void> result = JsonResult.failed(ResponseCode.BAD_REQUEST, message);
        log.debug("?????{}", result);
        return result;
    }
}
```

```
/**
 * ??????????
 */
@ExceptionHandler({Throwable.class})
public JsonResult<Void> handleSystemError(Throwable e) {
    log.debug("?????????={}????={}", e.getClass().getName(),
        e.getMessage());
    e.printStackTrace();
    JsonResult<Void> result = JsonResult.failed(ResponseCode.I
INTERNAL_SERVER_ERROR, e);
    log.debug("?????{}", result);
    return result;
}
}
```

commons项目编写内容到此结束

## 创建business模块

搭建基本结构

business可以理解为业务的

我们创建这个模块是为了触发订单生成业务的

business

创建csmall-business 也要父子相认

最终子项目pom文件为:

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

## 创建application.yml文件

```
server:
  port: 20000
#????
mybatis:
  configuration:
    # ???
    cache-enabled: false
    # ??????????,????user_name???,????java?userName??
    map-underscore-to-camel-case: true
    # ?????sql????????
```



```
public class Knife4jConfiguration {

    /**
     * ??????Controller???
     */
    private String basePackage = "cn.celinf.csmall.business.
controller";
    /**
     * ???
     */
    private String groupName = "base-business";
    /**
     * ???
     */
    private String host = "http://java.celinf.cn";
    /**
     * ??
     */
    private String title = "?????????API?--??business-
web?";
    /**
     * ??
     */
    private String description = "????business-web?,????";
    /**
     * ?????URL
     */
    private String termsOfServiceUrl = "http://www.apache.or
g/licenses/LICENSE-2.0";
    /**
     * ???
     */
    private String contactName = "?????";
    /**
     * ???
     */
    private String contactUrl = "http://java.celinf.cn";
    /**
     * ???
     */

```

```
    */
    private String contactEmail = "java@celinf.cn";
    /**
     * ???
     */
    private String version = "1.0-SNAPSHOT";

    @Autowired
    private OpenApiExtensionResolver openApiExtensionResolve
r;

    @Bean
    public Docket docket() {
        String groupName = "1.0-SNAPSHOT";
        Docket docket = new Docket(DocumentationType.SWAGGER_2)
            .host(host)
            .apiInfo(apiInfo())
            .groupName(groupName)
            .select()
            .apis(RequestHandlerSelectors.basePackage(basePackage))
            .paths(PathSelectors.any())
            .build()
            .extensions(openApiExtensionResolver.buildExtensions(groupName));
        return docket;
    }

    private ApiInfo apiInfo() {
        return new ApiInfoBuilder()
            .title(title)
            .description(description)
            .termsOfServiceUrl(termsOfServiceUrl)
            .contact(new Contact(contactName, contactUrl, contactEmail))
            .version(version)
            .build();
    }
}
```

## 编写业务

因为business是业务的触发者,不需要连接数据库,所以从业务逻辑层开始写即可

创建service包 包中创建接口IBusinessService

```
public interface IBusinessService {  
    // business?????????????  
    void buy();  
}
```

业务逻辑层实现

创建包service.impl 创建类BusinessServiceImpl

```
@Service  
@Slf4j  
public class BusinessServiceImpl implements IBusinessService  
{  
    @Override  
    public void buy() {  
        // ??????????  
        // ??OrderAddDTO?,??????  
        OrderAddDTO orderAddDTO=new OrderAddDTO();  
        orderAddDTO.setCommodityCode("PC100");  
        orderAddDTO.setUserId("UU100");  
        orderAddDTO.setMoney(500);  
        orderAddDTO.setCount(5);  
        // ???????,??????,??????  
        log.info("?????????:{ }",orderAddDTO);  
    }  
}
```

控制层代码

创建controller 包中创建BusinessController类

```
@RestController  
@RequestMapping("/base/business")
```

```
// knife4j?????????  
@Api(tags = "????????")  
public class BusinessController {  
    @Autowired  
    private IBusinessService businessService;  
  
    @PostMapping("/buy") // localhost:20000/base/business/buy  
    @ApiOperation("????")  
    public JsonResult buy(){  
        // ??????????  
        businessService.buy();  
        return JsonResult.ok("????");  
    }  
}
```

可以启动当前项目 <http://localhost:20000/doc.html>

点击测试,观察输出结果和控制台输出内容是否正常

## 注册到Nacos

我们的项目要想称为微服务项目必须注册到nacos

做具体配置之前,要明确,启动business之前,一定保证nacos在运行状态,否则启动business会报错的

首先在business的pom文件中添加依赖

```
<dependency>  
    <groupId>com.alibaba.cloud</groupId>  
    <artifactId>spring-cloud-starter-alibaba-nacos-  
discovery</artifactId>  
</dependency>
```

然后再application-dev.yml文件中添加当前项目对nacos注册的配置

```
spring:  
application:  
# ??Springboot?????,???????????
```

```
name: nacos-business
cloud:
nacos:
discovery:
# ??nacos?????
server-addr: localhost:8848
```

做完上面的配置

我们在保证nacos已经启动的前提下,我们启动business项目

启动之后,business一切功能正常,而且可以在nacos的服务列表中看到nacos-business的名称

nacos注册中心

## Nacos心跳机制

常见面试题

Nacos内部注册的服务分为两大类

1. 临时实例(默认)
2. 持久化实例(永久实例)

我们可以通过设置属性来确定它是临时还是永久

```
cloud:
  nacos:
    discovery:
      # ephemeral????????????nacos??? true(??):????? false:?????
      ephemeral: true
```

临时实例和永久实例的区别

临时实例

默认情况下,启动服务后,每隔5秒会向nacos发送一个"心跳包",这个心跳包中包含了当前服务的基本信息

Nacos收到这个"心跳包"

"如果发现这个服务的信息不在注册列表中,就进行注册,如果这个服务的信息在注册列表中就表明这个服务还是健康的"

如果Nacos15秒内没接收到某个服务的心跳包,Nacos会将这个服务标记为不健康的状态

如果30秒内没有接收到这个服务的心跳包,Nacos会将这个服务从注册列表中剔除

这些时间都是可以通过配置修改的

持久化实例(永久实例)

持久化实例启动时向nacos注册,nacos会对这个实例进行持久化处理

心跳包的规则和临时实例一致,只是不会将该服务从列表中剔除

## 各类型使用时机

一般情况下,我们创建的服务都是临时实例

只有项目的主干业务才会设置为永久实例

## 使用Idea启动Nacos

我们每次开机都要先启动nacos才能启动其他服务

我们使用dos窗口敲代码启动nacos还是比较麻烦的

我们可以使用idea提供的功能,简化nacos启动的过程

第一步：右上启动下拉

第二步：选择shell script

第三步：如图编写

配置完成后,就可以在idea中启动nacos

其余几个模块也如这个模块一样的。所以就不一一粘贴了。

- cart 购物车模块 ( 修改pom.xml、链接数据库yml文件 )
- order订单模块 ( 修改pom.xml、链接数据库yml文件 )
- stock库存模块 ( 修改pom.xml、链接数据库yml文件 )

学习记录，如有侵权请联系删除。