

---

# 百战商城

## 一、 项目介绍

### 1 电商行业发展

商务部统计数据显示,2012 年到 2016 年,我国网络购物用户人数从 2.42 亿人增长至 4.67 亿人,增长近一倍。电子商务交易额从 8.1 万亿元增长至 26.1 万亿元,年均增长 34%。其中,网络零售交易额从 1.31 万亿元增长至 5.16 万亿元,年均增长 40%,对社会消费品零售总额增加值的贡献率从 17%增长至 30%。电子商务发展直接和间接带动的就业人数从 1500 万人增长至 3700 万人。

经过多年发展,目前规模较大电子商务平台企业纷纷开始构建生态系统,平台为商家和消费者提供交易、支付、物流等各方面全周期支持与服务,各大平台与平台商家之间依存越来越紧密,阿里系、腾讯系、百度系、京东系等主体均取得了显著规模效益。

### 2 电商行业模式

B2B:企业到企业,商家到商家。代表:阿里巴巴。

B2C:商家到客户。代表:京东、淘宝商城(B2B2C)、天猫网。

C2C:客户到客户。淘宝集市、闲鱼、转转。

O2O:线上到线下。

### 3 百战商城介绍

百战商城项目是一个综合性的 B2C 电子商务平台,功能类似于淘宝、京东。用户可以在系统中通过搜索商品、查看商品详情、加入购物车、购买商品并生成订单完成购物操作。

百战商城共分为两部分:

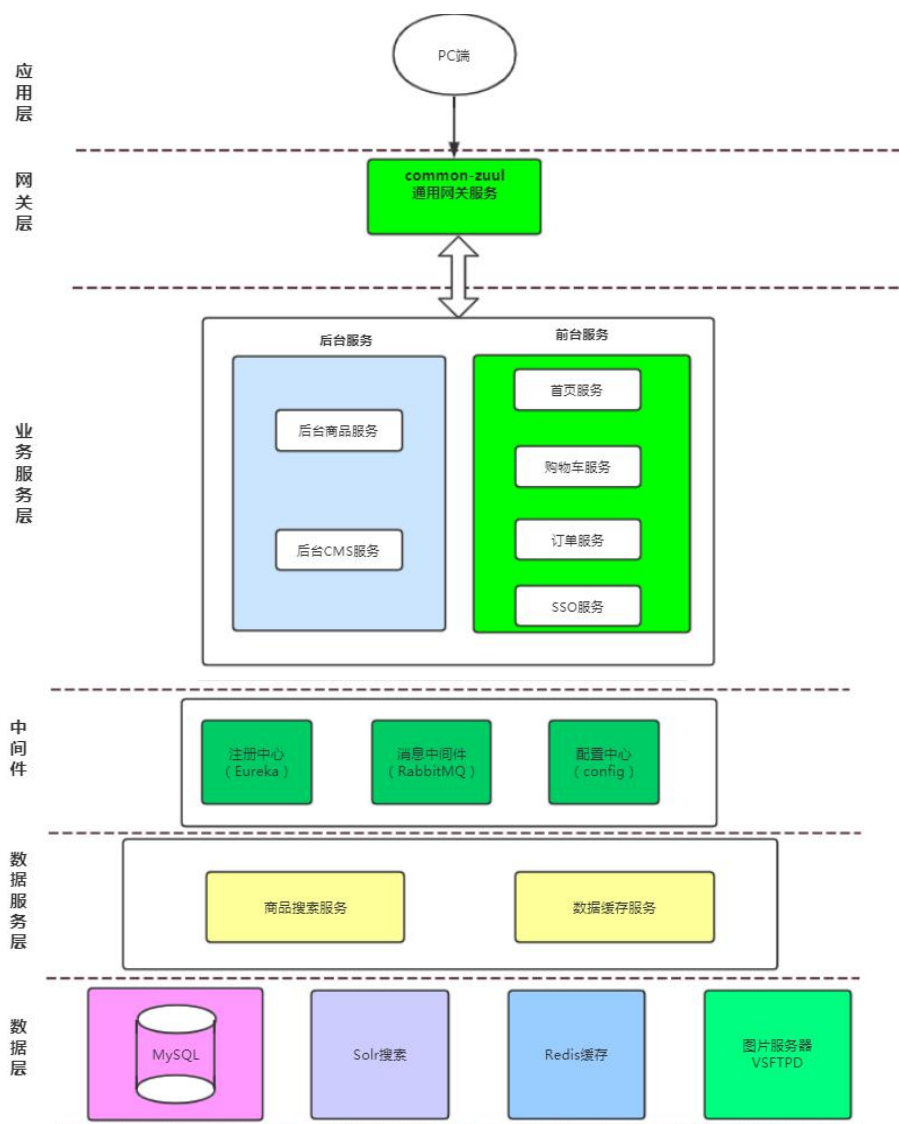
1) 商城后台管理系统：

主要实现对商品、商品分类、规格参数、CMS 等业务的处理。

2) 商城前台系统：

主要提供用户通过访问首页，完成购物流程的处理。

## 二、 项目架构介绍



---

### **三、 项目技术介绍**

#### **1 Spring Data**

##### **1.1 Spring Data Redis2.1.9.RELEASE**

##### **1.2 Spring Data Solr4.0.9.RELEASE**

#### **2 Spring Boot 2.1.6.RELEASE**

##### **2.1 Spring Boot Data Redis 2.1.6.RELEASE**

##### **2.2 Spring Boot Data Solr 2.1.6.RELEASE**

#### **3 Spring Cloud Greenwich.SR2**

##### **3.1 Spring Cloud Netflix Eureka**

##### **3.2 Spring Cloud Netflix Zuul**

##### **3.3 Spring Cloud Netflix Hystrix**

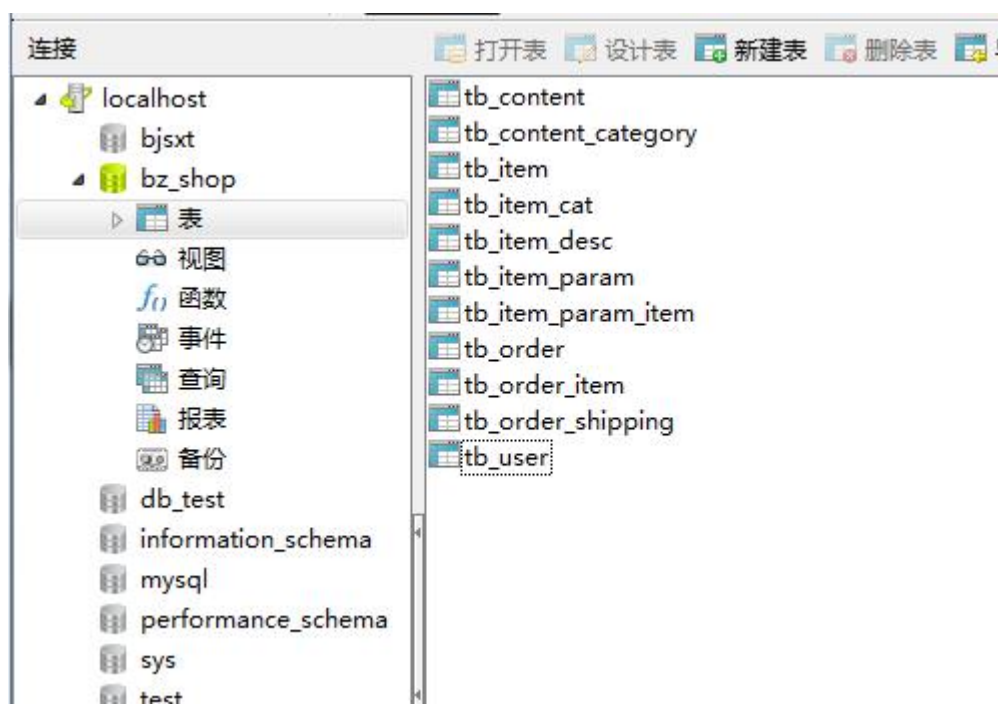
##### **3.4 Spring Cloud OpenFeign**

##### **3.5 Spring Cloud Config**

#### **4 TX-LCN 5.0.2.RELEASE**

## 四、 搭建项目环境

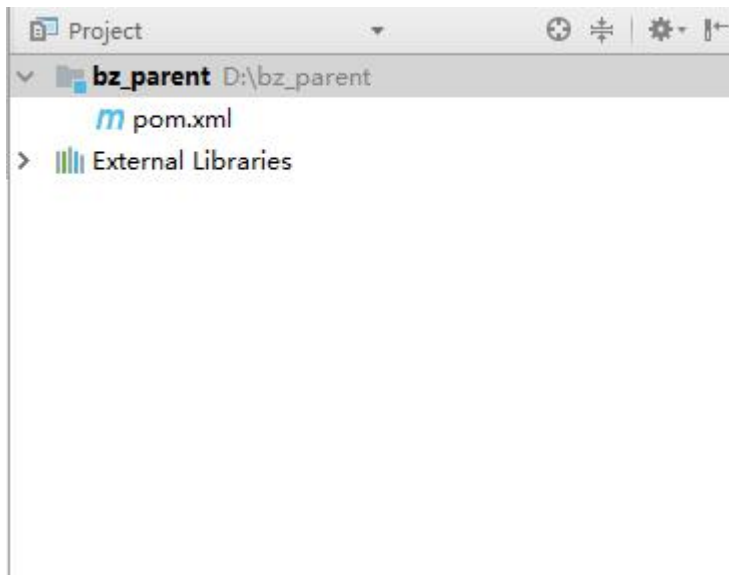
### 1 创建数据库并导入 sql 文件



## 2 创建项目

### 2.1 创建父工程

#### 2.1.1 创建项目



#### 2.1.2 修改 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
http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <groupId>com.bjsxt</groupId>

    <artifactId>bz_parent</artifactId>

    <packaging>pom</packaging>
```

```
<version>1.0-SNAPSHOT</version>
```

```
<!--Spring Boot 父工程-->
```

```
<parent>
```

```
<groupId>org.springframework.boot</groupId>
```

```
<artifactId>spring-boot-starter-parent</artifactId>
```

```
<version>2.1.6.RELEASE</version>
```

```
</parent>
```

```
<!-- 自定义属性标签 -->
```

```
<properties>
```

```
<mybatis-version>3.5.1</mybatis-version>
```

```
<mysql-connector-java-version>5.1.38</mysql-connector-java-version>
```

```
<druid-version>1.0.9</druid-version>
```

```
<pagehelper-version>1.2.10</pagehelper-version>
```

```
<logback-version>5.0</logback-version>
```

```
<spring-mybats-version>2.0.1</spring-mybats-version>
```

```
</properties>
```

```
<dependencyManagement>
```

```
<dependencies>
```

---

```
<!--Spring Cloud Platform-->
```

```
<dependency>
```

```
  <groupId>org.springframework.cloud</groupId>
```

```
  <artifactId>spring-cloud-dependencies</artifactId>
```

```
  <version>Greenwich.SR2</version>
```

```
  <type>pom</type>
```

```
  <scope>import</scope>
```

```
</dependency>
```

```
<!-- MyBatis -->
```

```
<dependency>
```

```
  <groupId>org.mybatis</groupId>
```

```
  <artifactId>mybatis</artifactId>
```

```
  <version>${mybatis-version}</version>
```

```
</dependency>
```

```
<!-- MySql Driver -->
```

```
<dependency>
```

```
  <groupId>mysql</groupId>
```

```
  <artifactId>mysql-connector-java</artifactId>
```

```
  <version>${mysql-connector-java-version}</version>
```

```
</dependency>
```

---

*<!--Alibaba DataBase Connection Pool-->*

**<dependency>**

**<groupId>**com.alibaba**</groupId>**

**<artifactId>**druid**</artifactId>**

**<version>**\${druid-version}**</version>**

**</dependency>**

*<!--PageHelper-->*

**<dependency>**

**<groupId>**com.github.pagehelper**</groupId>**

**<artifactId>**pagehelper-spring-boot-starter**</artifactId>**

**<version>**\${pagehelper-version}**</version>**

**</dependency>**

*<!--MyBatis And Spring Integration Starter-->*

**<dependency>**

**<groupId>**org.mybatis.spring.boot**</groupId>**

**<artifactId>**mybatis-spring-boot-starter**</artifactId>**

**<version>**\${spring-mybatis-version}**</version>**

**</dependency>**



```
<!--Logback-->

<dependency>

    <groupId>net.logstash.logback</groupId>

    <artifactId>logstash-logback-encoder</artifactId>

    <version>${logback-version}</version>

</dependency>

</dependencies>

</dependencyManagement>

<build>

    <pluginManagement>

        <!--Spring Boot Maven Plugin-->

        <plugins>

            <plugin>

                <groupId>org.springframework.boot</groupId>

                <artifactId>spring-boot-maven-plugin</artifactId>

            </plugin>

        </plugins>

    </pluginManagement>

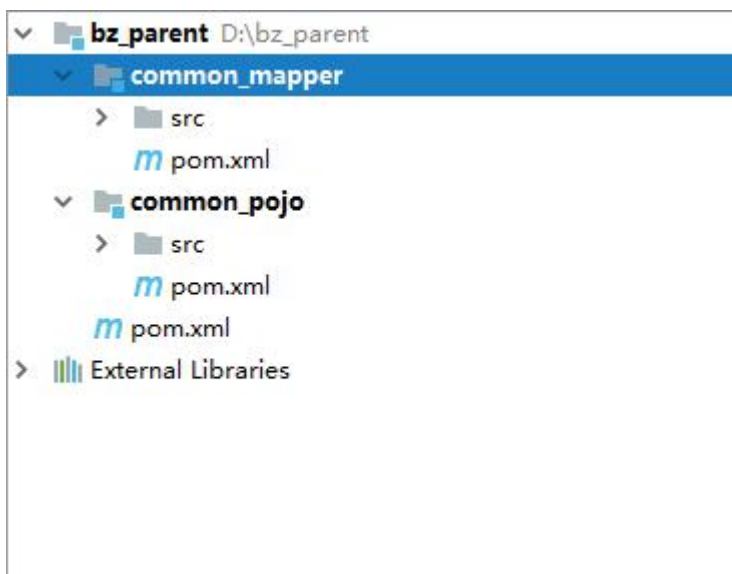
</build>
```

---

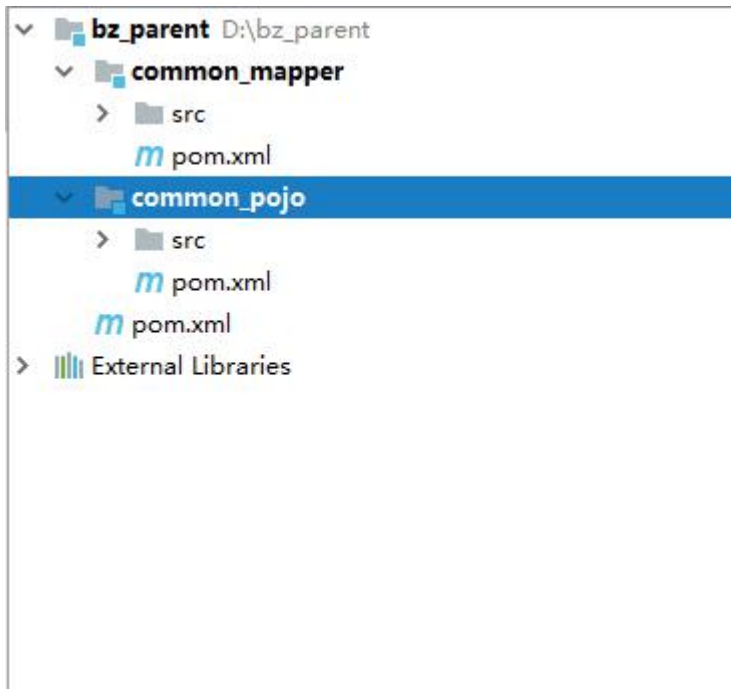
```
</project>
```

## 2.2 创建 Mapper 与 Pojo

### 2.2.1 创建 Mapper 项目



## 2.2.2 创建 Pojo 项目



## 2.3 使用工具生成 Mapper 与 Pojo

### 2.3.1 generatorSqlmapCustom 工具的使用

### 2.3.2 修改 Mapper 的 POM 文件添加依赖坐标

```
<?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
http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <parent>

        <artifactId>bz_parent</artifactId>

        <groupId>com.bjsxt</groupId>
```

```
<version>1.0-SNAPSHOT</version>
```

```
</parent>
```

```
<modelVersion>4.0.0</modelVersion>
```

```
<artifactId>common_mapper</artifactId>
```

```
<dependencies>
```

```
<!--Pojo-->
```

```
<dependency>
```

```
<groupId>com.bjsxt</groupId>
```

```
<artifactId>common_pojo</artifactId>
```

```
<version>1.0-SNAPSHOT</version>
```

```
</dependency>
```

```
<!-- MyBatis -->
```

```
<dependency>
```

```
<groupId>org.mybatis</groupId>
```

```
<artifactId>mybatis</artifactId>
```

```
<version>${mybatis-version}</version>
```

```
</dependency>
```

```
<!-- MySql Driver -->
```

---

```
<dependency>
```

```
    <groupId>mysql</groupId>
```

```
    <artifactId>mysql-connector-java</artifactId>
```

```
    <version>${mysql-connector-java-version}</version>
```

```
</dependency>
```

```
<!--Alibaba DataBase Connection Pool-->
```

```
<dependency>
```

```
    <groupId>com.alibaba</groupId>
```

```
    <artifactId>druid</artifactId>
```

```
    <version>${druid-version}</version>
```

```
</dependency>
```

```
<!--PageHelper-->
```

```
<dependency>
```

```
    <groupId>com.github.pagehelper</groupId>
```

```
    <artifactId>pagehelper-spring-boot-starter</artifactId>
```

```
    <version>${pagehelper-version}</version>
```

```
</dependency>
```

```
<!--MyBatis And Spring Integration Starter-->
```

```
<dependency>
```

```
<groupId>org.mybatis.spring.boot</groupId>

<artifactId>mybatis-spring-boot-starter</artifactId>

<version>${spring-mybats-version}</version>

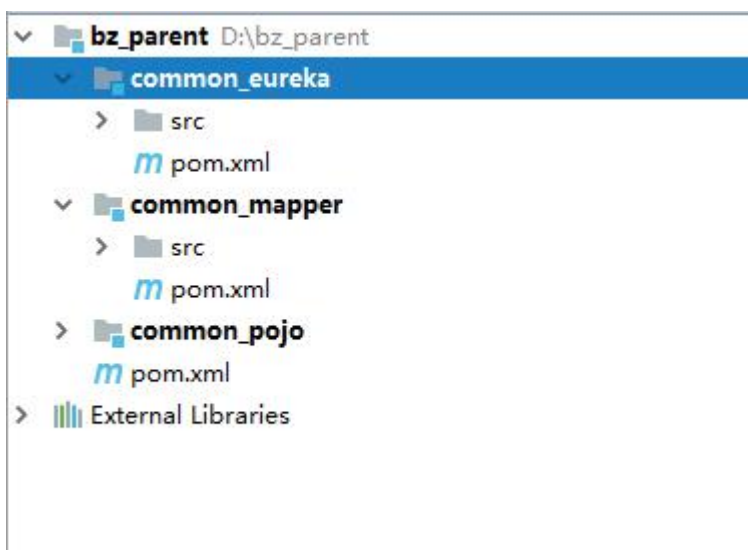
</dependency>

</dependencies>

</project>
```

### 3 搭建 Eureka 注册中心

#### 3.1 创建 Eureka 服务



#### 3.2 修改 POM 文件添加依赖

```
<?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`

`http://maven.apache.org/xsd/maven-4.0.0.xsd">`

`<parent>`

`<artifactId>bz_parent</artifactId>`

`<groupId>com.bjsxt</groupId>`

`<version>1.0-SNAPSHOT</version>`

`</parent>`

`<modelVersion>4.0.0</modelVersion>`

`<artifactId>common_eureka</artifactId>`

`<dependencies>`

`<!--Spring Boot Web Starter-->`

`<dependency>`

`<groupId>org.springframework.boot</groupId>`

`<artifactId>spring-boot-starter-web</artifactId>`

`</dependency>`

`<!--Spring Cloud Eureka Server Starter-->`

`<dependency>`

`<groupId>org.springframework.cloud</groupId>`

`<artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>`

```
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

### 3.3 创建配置文件，添加配置

```
spring:

  application:

    name: eureka-server

server:

  port: 8761
```



---

*#关闭注册*

**eureka:**

**client:**

**register-with-eureka: false**

**fetch-registry: false**

### 3.4 创建启动类，添加相关注解

```
/**  
 * Eureka 注册中心  
 */  
  
@SpringBootApplication  
@EnableEurekaServer  
  
public class EurekaServerApplication {  
  
    public static void main(String[] args){  
  
        SpringApplication.run(EurekaServerApplication.class,args);  
  
    }  
}
```

---

## 3.5 将 Eureka 注册中心部署到 Linux 环境中

### 3.5.1 安装 lrzsz 上传下载工具

安装命令：yum install lrzsz -y

上传命令：rz

下载命令：sz

### 3.5.2 上传注册中心 jar

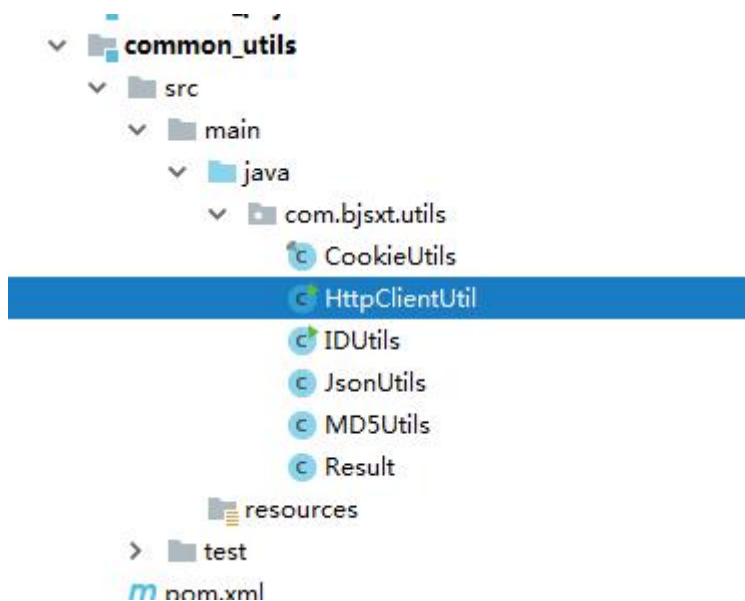
```
total 42676
-rw-r--r--. 1 root root 43689434 Aug 20 23:51 common_eureka.jar
drwxr-xr-x. 2 root root    4096 Aug 20 23:52 logs
-rwxr-xr-x. 1 root root    2194 Aug 20 23:48 server.sh
[root@localhost eureka]#
```

### 3.5.3 上传启动脚本文件(server.sh)

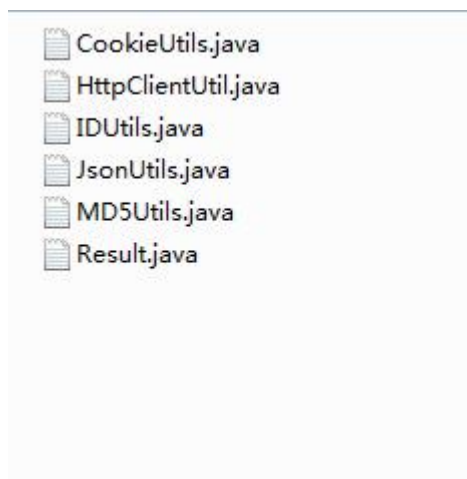
```
total 42676
-rw-r--r--. 1 root root 43689434 Aug 20 23:51 common_eureka.jar
drwxr-xr-x. 2 root root    4096 Aug 20 23:52 logs
-rwxr-xr-x. 1 root root    2194 Aug 20 23:48 server.sh
[root@localhost eureka]#
```

## 3.6 创建 common\_utils 项目(工具集)

### 3.6.1 创建项目



### 3.6.2 添加常用工具类



### 3.6.3 修改 POM 文件添加依赖

```
<?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
```

```
http://maven.apache.org/xsd/maven-4.0.0.xsd">
```

```
<parent>
```

```
<artifactId>bz_parent</artifactId>
```

```
<groupId>com.bjsxt</groupId>
```

```
<version>1.0-SNAPSHOT</version>
```

```
</parent>
```

```
<modelVersion>4.0.0</modelVersion>
```

```
<artifactId>common_utils</artifactId>
```

```
<dependencies>
```

```
<!--Spring Boot Web Starter-->
```

```
<dependency>
```

```
<groupId>org.springframework.boot</groupId>
```

```
<artifactId>spring-boot-starter-web</artifactId>
```

```
</dependency>
```

```
<!--
```

```
https://mvnrepository.com/artifact/org.apache.httpcomponents/httpclient -->
```

```
<dependency>
```

```
<groupId>org.apache.httpcomponents</groupId>
```

```
<artifactId>httpclient</artifactId>

<version>4.5.2</version>

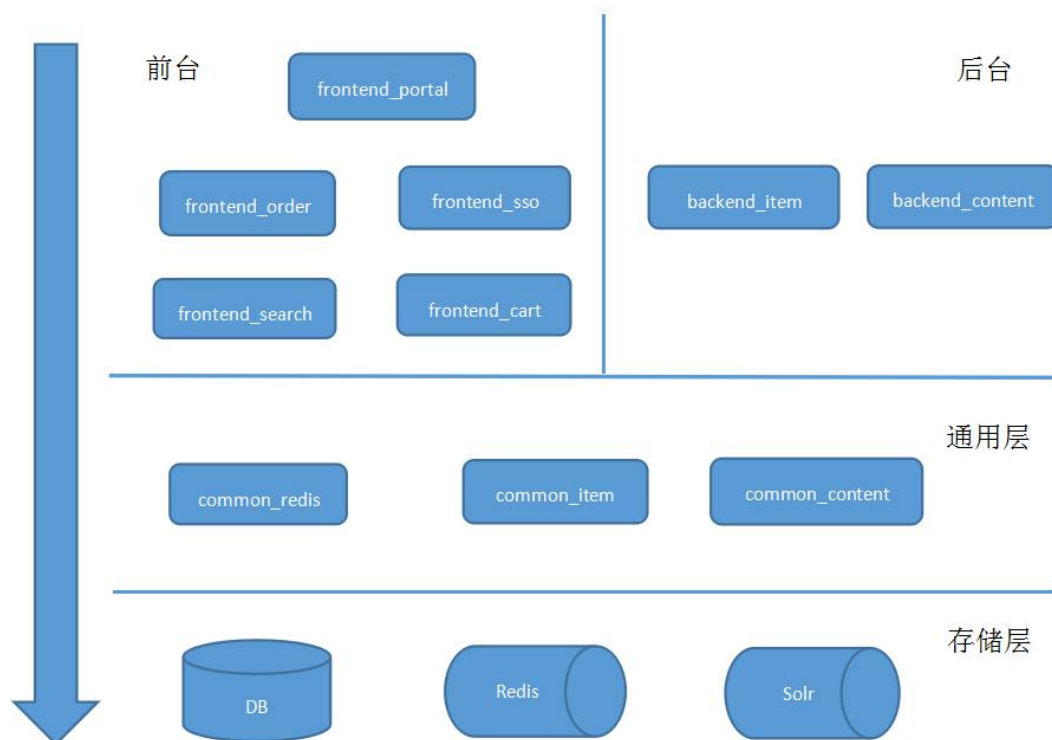
</dependency>

</dependencies>

</project>
```

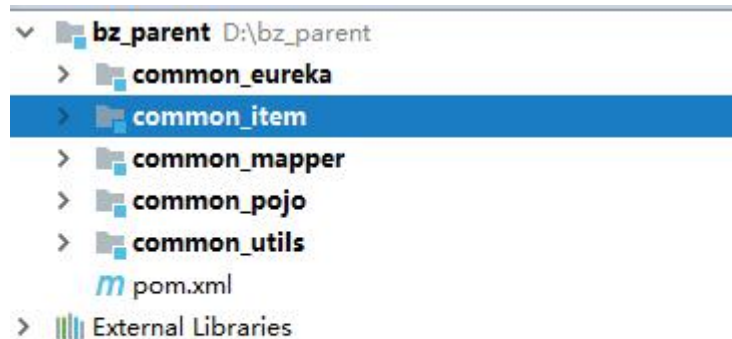
## 五、 开发百战商城后台系统

### 1 百战商城服务设计



## 2 创建 common\_item 服务

### 2.1 创建项目



### 2.2 修改 POM 文件添加依赖坐标

```
<?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

http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <parent>

        <artifactId>bz_parent</artifactId>

        <groupId>com.bjsxt</groupId>

        <version>1.0-SNAPSHOT</version>

    </parent>

    <modelVersion>4.0.0</modelVersion>
```

```
<artifactId>common_item</artifactId>
```

```
<dependencies>
```

```
<!--mapper-->
```

```
<dependency>
```

```
<groupId>com.bjsxt</groupId>
```

```
<artifactId>common_mapper</artifactId>
```

```
<version>1.0-SNAPSHOT</version>
```

```
</dependency>
```

```
<!--utils-->
```

```
<dependency>
```

```
<groupId>com.bjsxt</groupId>
```

```
<artifactId>common_utils</artifactId>
```

```
<version>1.0-SNAPSHOT</version>
```

```
</dependency>
```

```
<!--Spring Boot Web Starter-->
```

```
<dependency>
```

```
<groupId>org.springframework.boot</groupId>
```

```
<artifactId>spring-boot-starter-web</artifactId>
```

```
</dependency>

<!--Spring Cloud Eureka Client Starter-->

<dependency>

    <groupId>org.springframework.cloud</groupId>

    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>

</dependency>

</dependencies>

</project>
```

## 2.3 创建配置文件，添加相关配置

**spring:**

**application:**

**name:** common-item

**datasource:**

**driverClassName:** com.mysql.jdbc.Driver

**url:** jdbc:mysql://localhost:3306/db\_shop?characterEncoding=UTF-8

**username:** root

**password:** root

**type:** com.alibaba.druid.pool.DruidDataSource



**server:**

**port:** 9010

**eureka:**

**client:**

**serviceUrl:**

**defaultZone:** http://eureka-server:8761/eureka/

## 2.4 修改 common\_mapper 项目的 POM 文件 , 添加资源拷贝插件

```
<build>  
  <resources>  
    <resource>  
      <directory>src/main/java</directory>  
      <includes>  
        <include>**/*.xml</include>  
      </includes>  
    </resource>  
    <resource>  
      <directory>src/main/resources</directory>
```

```
<includes>

    <include>**/*.xml</include>

</includes>

</resource>

</resources>

</build>
```

## 2.5 创建启动类，添加相关注解

```
/**
 * 通用服务 Common_item
 */

@SpringBootApplication

@EnableDiscoveryClient

@MapperScan("com.bjsxt.mapper")

public class CommonItemApplication {

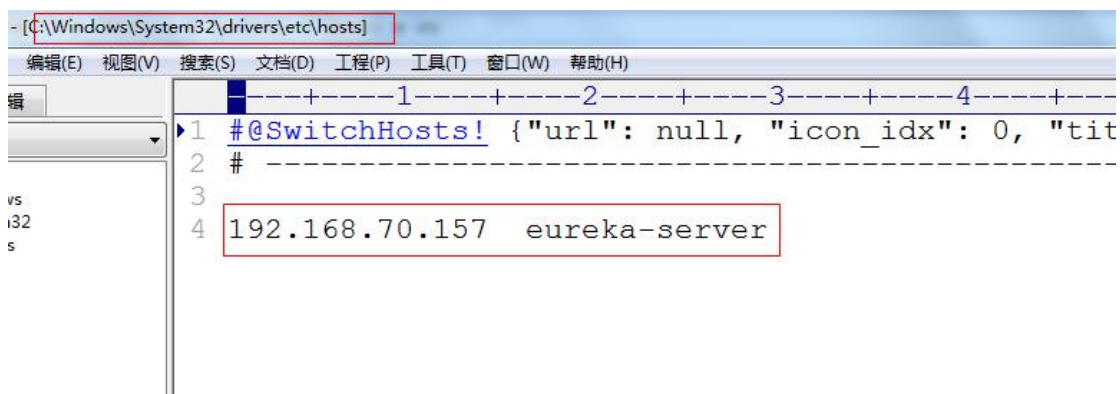
    public static void main(String[] args){

        SpringApplication.run(CommonItemApplication.class,args);

    }

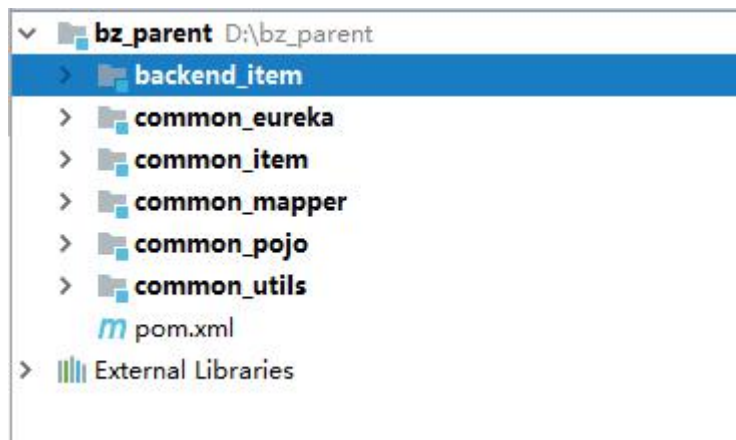
}
```

## 2.6 修改 Host 文件添加注册中心域名与 IP 的映射

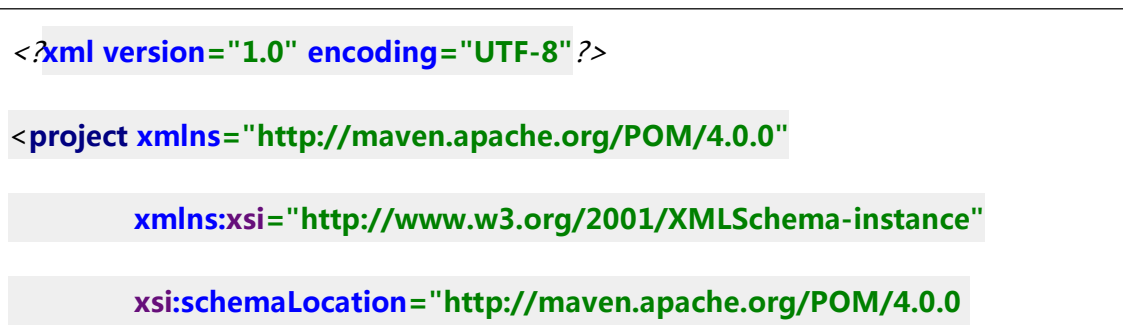


## 3 创建 backend\_item 服务

### 3.1 创建项目



### 3.2 修改 POM 文件添加依赖坐标



---

```
http://maven.apache.org/xsd/maven-4.0.0.xsd">
```

```
<parent>
```

```
<artifactId>bz_parent</artifactId>
```

```
<groupId>com.bjsxt</groupId>
```

```
<version>1.0-SNAPSHOT</version>
```

```
</parent>
```

```
<modelVersion>4.0.0</modelVersion>
```

```
<artifactId>backend_item</artifactId>
```

```
<dependencies>
```

```
<!--pojo-->
```

```
<dependency>
```

```
<groupId>com.bjsxt</groupId>
```

```
<artifactId>common_pojo</artifactId>
```

```
<version>1.0-SNAPSHOT</version>
```

```
</dependency>
```

```
<!--utils-->
```

```
<dependency>
```

```
<groupId>com.bjsxt</groupId>
```

```
<artifactId>common_utils</artifactId>

<version>1.0-SNAPSHOT</version>

</dependency>

<!--Spring Boot Web Starter-->

<dependency>

    <groupId>org.springframework.boot</groupId>

    <artifactId>spring-boot-starter-web</artifactId>

</dependency>

<!--Spring Cloud Eureka Client Starter-->

<dependency>

    <groupId>org.springframework.cloud</groupId>

    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>

</dependency>

<!--Spring Cloud OpenFeign Starter-->

<dependency>

    <groupId>org.springframework.cloud</groupId>

    <artifactId>spring-cloud-starter-openfeign</artifactId>

</dependency>

</dependencies>

</project>
```

---

### 3.3 创建配置文件，添加相关配置

```
spring:

  application:

    name: backend-item

server:

  port: 9020

eureka:

  client:

    serviceUrl:

      defaultZone: http://eureka-server:8761/eureka/
```

### 3.4 创建启动类，添加相关注解

```
/**
 * BackendItem 服务
 */
@SpringBootApplication
@EnableDiscoveryClient
```

---

`@EnableFeignClients`

```
public class BackendItemApplication {  
  
    public static void main(String[] args){  
  
        SpringApplication.run(BackendItemApplication.class,args);  
  
    }  
  
}
```

## 4 开发商品管理接口

### 4.1 实现查询商品接口

#### 4.1.1 在 common\_item 服务中实现查询商品

##### 4.1.1.1 在 common\_utils 项目中添加 PageResult 模型

```
/**  
  
 * 分页模型  
  
 */  
  
public class PageResult implements Serializable{  
  
  
    private Integer pageIndex;//当前页  
  
    private Long totalPage;//总页数  
  
    private List result;//结果集
```

---

```
public Integer getPageIndex() {
```

```
    return pageIndex;
```

```
}
```

```
public Long getTotalPage() {
```

```
    return totalPage;
```

```
}
```

```
public List getResult() {
```

```
    return result;
```

```
}
```

```
public void setPageIndex(Integer pageIndex) {
```

```
    this.pageIndex = pageIndex;
```

```
}
```

```
public void setTotalPage(Long totalPage) {
```

```
    this.totalPage = totalPage;
```

```
}
```

```
public void setResult(List result) {
```

```
    this.result = result;
```



```
}  
  
}
```

#### 4.1.1.2 创建 controller

```
@RestController  
  
@RequestMapping("/service/item")  
  
public class ItemController {  
  
    @Autowired  
  
    private ItemService itemService;  
  
    /**  
     * 查询商品数据  
     */  
  
    @RequestMapping(value="/selectTbItemAllByPage",method =  
RequestMethod.GET)  
  
    public PageResult selectTbItemAllByPage(@RequestParam Integer  
page,@RequestParam Integer rows){  
  
        return this.itemService.selectTbItemAllByPage(page,rows);  
  
    }  
  
}
```

---

#### 4.1.1.3 创建 service

```
public interface ItemService {  
  
    PageResult selectTbItemAllByPage(Integer page,Integer rows);  
  
}
```

#### 4.1.1.4 创建 serviceImpl

```
@Service  
  
public class ItemServiceImpl implements ItemService {  
  
    @Autowired  
  
    private TbItemMapper tbItemMapper;  
  
    /**  
     * 查询所有商品，并分页。  
     * @param page  
     * @param rows  
     * @return
```

---

\*/

@Override

```
public PageResult selectTbItemAllByPage(Integer page, Integer rows) {  
  
    PageHelper.startPage(page, rows);  
  
    TbItemExample example = new TbItemExample();  
  
    TbItemExample.Criteria criteria = example.createCriteria();  
  
    criteria.andStatusEqualTo((byte)1);  
  
    List<TbItem> list = this.tbItemMapper.selectByExample(example);  
  
    PageInfo<TbItem> pageInfo = new PageInfo<>(list);  
  
    PageResult result = new PageResult();  
  
    result.setPageIndex(page);  
  
    result.setTotalPage(pageInfo.getTotal());  
  
    result.setResult(list);  
  
    return result;  
  
}  
}
```

## 4.1.2 在 backend\_item 服务中实现商品查询

### 4.1.2.1 创建 controller

@RestController

```
@RequestMapping("/backend/item")

public class ItemController {

    @Autowired

    private ItemService itemService;

    /**
     * 查询商品并分页处理
     * @return
     */

    @RequestMapping("/selectTbItemAllByPage")

    public Result selectTbItemAllByPage(@RequestParam(defaultValue = "1")
Integer page,@RequestParam(defaultValue = "2") Integer rows){

        try{

            return this.itemService.selectTbItemAllByPage(page,rows);

        }catch (Exception e){

            e.printStackTrace();

        }

        return Result.build(500,"error");

    }

}
```

---

#### 4.1.2.2 创建 service

```
public interface ItemService {  
  
    Result selectTbItemAllByPage(Integer page,Integer rows);  
  
}
```

#### 4.1.2.3 创建 serviceImpl

```
/**  
  
 * 商品管理  
  
 */  
  
@Service  
  
public class ItemServiceImpl implements ItemService {  
  
    @Autowired  
  
    private CommonItemFeignClient commonItemFeignClient;  
  
    @Override  
  
    public Result selectTbItemAllByPage(Integer page, Integer rows) {  
  
        PageResult pageResult =  
  
this.commonItemFeignClient.selectTbItemAllByPage(page,rows);  
  
    }  
  
}
```

```
        if(pageResult != null && pageResult.getResult() != null &&
pageResult.getResult().size() > 0){

            return Result.ok(pageResult);

        }

        return Result.error("查无结果");

    }

}
```

#### 4.1.2.4创建 feignClient

```
@FeignClient(value = "common-item")

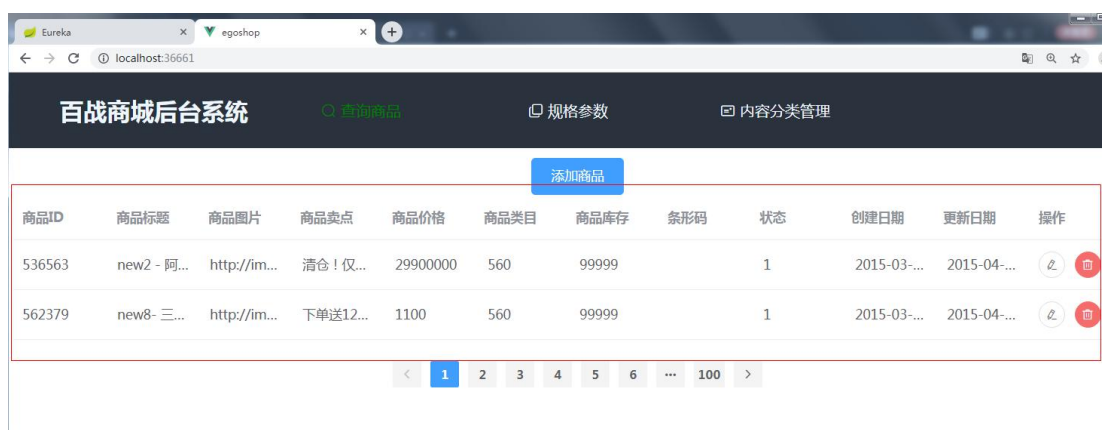
public interface CommonItemFeignClient {





    @GetMapping("/service/item/selectTbItemAllByPage")

    PageResult selectTbItemAllByPage(@RequestParam("page") Integer page,
    @RequestParam("rows") Integer rows);

}
```

### 4.1.2.5 测试



商品ID	商品标题	商品图片	商品卖点	商品价格	商品类目	商品库存	条形码	状态	创建日期	更新日期	操作
536563	new2 - 阿...	http://im...	清仓！仅...	29900000	560	99999		1	2015-03-...	2015-04-...	 
562379	new8- 三...	http://im...	下单送12...	1100	560	99999		1	2015-03-...	2015-04-...	 

## 4.2 实现添加商品接口

### 4.2.1 在 common\_item 服务中实现商品分类查询

#### 4.2.1.1 创建 controller

```
/**  
  
 * 商品类目  
  
 */  
  
@RestController  
@RequestMapping("/service/itemCategory")  
  
public class ItemCategoryController {  
  
    @Autowired  
  
    private ItemCategoryService itemCategoryService;  
  
    /**  
  
     * 根据父节点查询子节点  
  
     */  
}
```

```
@RequestMapping("/selectItemCategoryByParentId")

public List<TbItemCat> selectItemCategoryByParentId(@RequestParam Long
id){

    return this.itemCategoryService.selectItemCategoryByParentId(id);

}

}
```

#### 4.2.1.2 创建 service

```
public interface ItemCategoryService {

    List<TbItemCat> selectItemCategoryByParentId(Long id);

}
```

#### 4.2.1.3 创建 serviceImpl

```
/**

 * 商品分类查询

 */

@Service

public class ItemCategoryServiceImpl implements ItemCategoryService {
```



---

@Autowired

**private** TbItemCatMapper **tbItemCatMapper**;

/\*\*

\* 根据分类 ID 查询子节点

\* @param id

\* @return

\*/

@Override

**public** List<TbItemCat> selectItemCategoryByParentId(Long id) {

TbItemCatExample example = **new** TbItemCatExample();

TbItemCatExample.Criteria criteria = example.createCriteria();

criteria.andParentIdEqualTo(id);

criteria.andStatusEqualTo(1);

List<TbItemCat> list = **this.tbItemCatMapper**.selectByExample(example);

**return** list;

}

}

---

## 4.2.2 在 backend\_item 服务中实现商品分类查询

### 4.2.2.1 创建 controller

```
/**
 * 商品类目 Controller
 */

@RestController

@RequestMapping("/backend/itemCategory")

public class ItemCategoryController {

    @Autowired

    private ItemCategoryService itemCategoryService;

    /**
     * 根据类目 ID 查询当前类目的子节点
     */

    @RequestMapping("/selectItemCategoryByParentId")

    public Result selectItemCategoryByParentId(@RequestParam(value =
    "id",defaultValue = "0") Long id){

        try{

            return this.itemCategoryService.selectItemCategoryByParentId(id);

        }catch (Exception e){

            e.printStackTrace();

        }

    }

}
```

```
    }

    return Result.build(500,"error");

}

}
```

#### 4.2.2.2 创建 service

```
public interface ItemCategoryService {

    Result selectItemCategoryByParentId(Long id);

}
```

#### 4.2.2.3 创建 serviceImpl

```
/**
 * 商品类目 Service
 */
@Service
public class ItemCategoryServiceImpl implements ItemCategoryService {
```

---

**@Autowired**

**private** CommonItemFeignClient **commonItemFeignClient**;

/\*\*

\* 根据类目 ID 查询子类目

\* @param id

\* @return

\*/

**@Override**

**public** Result selectItemCategoryByParentId(Long id) {

List<TbItemCat> list =

**this.commonItemFeignClient**.selectItemCategoryByParentId(id);

**if**(list != **null** && list.size() > 0){

**return** Result.ok(list);

}

**return** Result.error("查无结果");

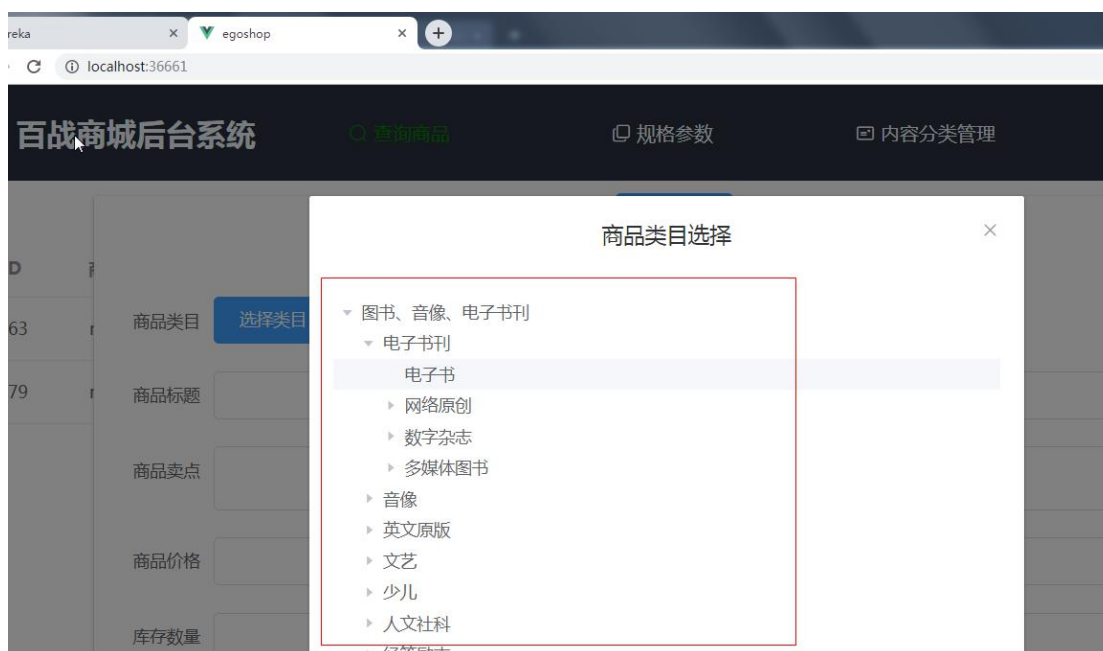
}

}

#### 4.2.2.4修改 feignClient

```
//-----/Service/itemCategory  
  
@PostMapping("/service/itemCategory/selectItemCategoryByParentId")  
  
List< TbItemCat> selectItemCategoryByParentId(@RequestParam("id") Long id);
```

#### 4.2.2.5测试



### 4.2.3 在 common\_item 服务中实现规格参数模板查询

#### 4.2.3.1创建 controller

```
/**  
  
 * ItemParam  
  
 */  
  
@RestController
```

```
@RequestMapping("/service/itemParam")

public class ItemParamController {

    @Autowired

    private ItemParamService itemParamService;

    /**
     * 根据商品分类 ID 查询规格参数模板
     */

    @RequestMapping("/selectItemParamByItemCatId")

    public TbItemParam selectItemParamByItemCatId(@RequestParam Long
itemCatId){

        return this.itemParamService.selectItemParamByItemCatId(itemCatId);

    }

}
```

#### 4.2.3.2 创建 service

```
public interface ItemParamService {

    TbItemParam selectItemParamByItemCatId(Long itemCatId);

}
```

---

```
}
```

#### 4.2.3.3 创建 serviceImpl

```
/**
 * ItemParamService
 */
@Service
public class ItemParamServiceImpl implements ItemParamService {

    @Autowired
    private TbItemParamMapper tbItemParamMapper;

    @Override
    public TbItemParam selectItemParamByItemCatId(Long itemCatId) {
        TbItemParamExample example = new TbItemParamExample();
        TbItemParamExample.Criteria criteria = example.createCriteria();
        criteria.andItemCatIdEqualTo(itemCatId);

        List<TbItemParam> list =
            this.tbItemParamMapper.selectByExampleWithBLOBs(example);

        if(list != null && list.size() > 0){
            return list.get(0);
        }
    }
}
```

```
    }

    return null;

}

}
```

## 4.2.4 在 backend\_item 服务中实现规格参数模板查询

### 4.2.4.1 创建 controller

```
/**
 * ItemParam
 */
@RestController
@RequestMapping("/backend/itemParam")
public class ItemParamController {

    @Autowired
    private ItemParamService itemParamService;

    /**
     * 根据商品分类 ID 查询规格参数模板
     */
    @RequestMapping("/selectItemParamByItemCatId/{itemCatId}")
```



```
public Result selectItemParamByItemCatId(@PathVariable("itemCatId") Long
itemCatId){

    try{

        return

this.itemParamService.selectItemParamByItemCatId(itemCatId);

    }catch (Exception e){

        e.printStackTrace();

    }

    return Result.build(500,"error");

}

}
```

#### 4.2.4.2 创建 service

```
public interface ItemParamService {

    Result selectItemParamByItemCatId(Long itemCatId);

}
```

#### 4.2.4.3 创建 serviceImpl

```
/**
```

```
* ItemParamService

*/

@Service

public class ItemParamServiceImpl implements ItemParamService {

    @Autowired

    private CommonItemFeignClient commonItemFeignClient;

    @Override

    public Result selectItemParamByItemCatId(Long itemCatId) {

        TbItemParam tbItemParam =

this.commonItemFeignClient.selectItemParamByItemCatId(itemCatId);

        if(tbItemParam != null){

            return Result.ok(tbItemParam);

        }

        return Result.error("查无结果");

    }

}
```

#### 4.2.4.4修改 feignClient

```
@FeignClient(value = "common-item")

public interface CommonItemFeignClient {

    //-----/Service/Item

    @GetMapping("/service/item/selectTbItemAllByPage")

    PageResult selectTbItemAllByPage(@RequestParam("page") Integer page,
    @RequestParam("rows") Integer rows);

    //-----/Service/itemCategory

    @PostMapping("/service/itemCategory/selectItemCategoryByParentId")

    List<TbItemCat> selectItemCategoryByParentId(@RequestParam("id") Long
id);

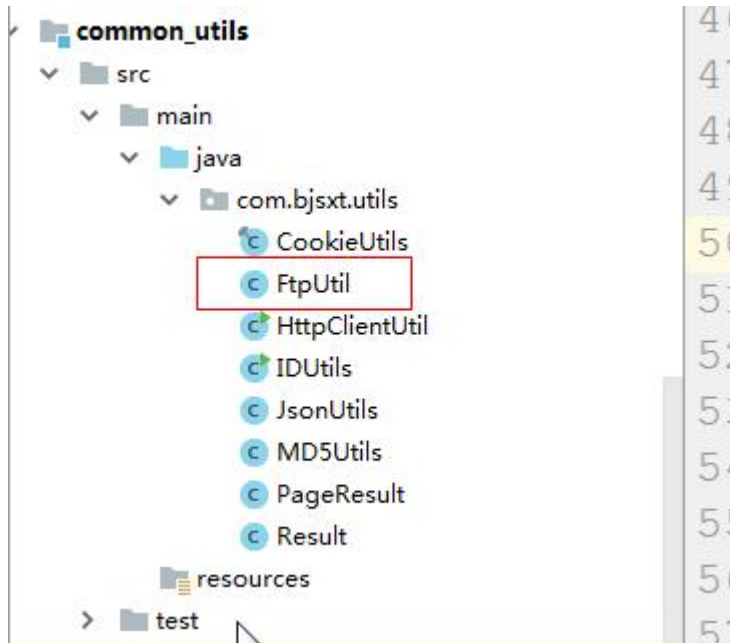
    //-----/Service/itemParam-----

    @PostMapping("/service/itemParam/selectItemParamByItemCatId")

    TbItemParam selectItemParamByItemCatId(@RequestParam("itemCatId")
Long itemCatId);
}
```

## 4.2.5 在 backend\_item 服务中处理图片上传

### 4.2.5.1 添加 FtpUtil 工具类



### 4.2.5.2 创建 controller

```
/**
 * 图片上传
 */

@RestController
@RequestMapping("/file")

public class FileUploadController {

    @Autowired

    private FileUploadService fileUploadService;
```

```
/**
 * 图片上传
 */

@RequestMapping("/upload")

public Result fileUpload(MultipartFile file){

    try{

        return this.fileUploadService.fileUpload(file);

    }catch (Exception e){

        e.printStackTrace();

    }

    return Result.build(500,"error");

}

}
```

#### 4.2.5.3 创建 service

```
public interface FileUploadService {

    Result fileUpload(MultipartFile file);

}
```

---

#### 4.2.5.4创建 serviceImpl

```
/**
 * 上传图片 Service
 */

@Service

public class FileUploadServiceImpl implements FileUploadService {

    /*FTP_HOST: 192.168.70.144

    FTP_PORT: 21

    FTP_USERNAME: ftpuser

    FTP_PASSWORD: ftpuser

    FTP_BASEPATH: /home/ftpuser/*/

    @Value("${FTP_HOST}")

    private String FTP_HOST;

    @Value("${FTP_PORT}")

    private int FTP_PORT;

    @Value("${FTP_USERNAME}")

    private String FTP_USERNAME;

    @Value("${FTP_PASSWORD}")
```

```
private String FTP_PASSWORD;

@Value("${FTP_BASEPATH}")

private String FTP_BASEPATH;

@Override

public Result fileUpload(MultipartFile file) {

    try {

        //定义上传图片的目录结构

        SimpleDateFormat sdf = new SimpleDateFormat("/yyyy/MM/dd/");

        String path = sdf.format(new Date());

        //设置新的文件名

        String newFileName = IDUtils.genImageName() +

file.getOriginalFilename().substring(file.getOriginalFilename().lastIndexOf("."));

        FtpUtil.uploadFile(this.FTP_HOST, this.FTP_PORT,

this.FTP_USERNAME, this.FTP_PASSWORD, this.FTP_BASEPATH, path,

newFileName, file.getInputStream());

        String imageURL = "http://" + this.FTP_HOST + path + newFileName;

        return Result.ok(imageURL);

    } catch (Exception e) {

        e.printStackTrace();

    }

}
```

```
        return null;

    }

}
```

#### 4.2.5.5 在配置文件中添相关配置

```
FTP_HOST: 192.168.70.144

FTP_PORT: 21

FTP_USERNAME: ftpuser

FTP_PASSWORD: ftpuser

FTP_BASEPATH: /home/ftpuser/
```

### 4.2.6 在 common\_item 服务中实现商品添加

#### 4.2.6.1 修改 controller

```
/**

 * 商品的添加

 */

@RequestMapping("/insertTbItem")

public Integer insertTbItem(@RequestBody TbItem tbItem){

    return this.itemService.insertTbItem(tbItem);

}
```



---

#### 4.2.6.2 修改 service

```
Integer insertTbItem(TbItem tbItem);
```

#### 4.2.6.3 修改 serviceImpl

```
/**
 * 持久化 TbItem
 * @param tbItem
 * @return
 */
@Override
public Integer insertTbItem(TbItem tbItem) {
    return this.tbItemMapper.insert(tbItem);
}
```

---

## 4.2.7 在 common\_item 服务中实现添加商品描述

### 4.2.7.1 创建 controller

```
/**
 * commonItem itemDescController
 */
@RestController
@RequestMapping("/service/itemDesc")
public class ItemDescController {

    @Autowired
    private ItemDescService itemDescService;

    /**
     * 添加商品描述
     * @param tbItemDesc
     * @return
     */
    @RequestMapping("/insertItemDesc")
    public Integer insertItemDesc(@RequestBody TbItemDesc tbItemDesc){
        return this.itemDescService.insertTbitemDesc(tbItemDesc);
    }
}
```

---

```
}
```

#### 4.2.7.2 创建 service

```
public interface ItemDescService {  
  
    Integer insertTbitemDesc(TbItemDesc tbItemDesc);  
  
}
```

#### 4.2.7.3 创建 serviceImpl

```
@Service  
  
public class ItemDescServiceImpl implements ItemDescService {  
  
    @Autowired  
  
    private TbItemDescMapper tbItemDescMapper;  
  
    @Override  
  
    public Integer insertTbitemDesc(TbItemDesc tbItemDesc) {  
  
        return this.tbItemDescMapper.insert(tbItemDesc);  
  
    }  
  
}
```

## 4.2.8 在 common\_item 服务中实现商品规格参数的添加

### 4.2.8.1 创建 controller

```
@RestController

@RequestMapping("/service/itemParamItem")

public class ItemParamItemController {

    @Autowired

    private ItemParamItemService itemParamItemService;

    /**
     * 添加商品规格参数
     */

    @RequestMapping("/insertTbItemParamItem")

    public Integer insertTbItemParamItem(@RequestBody TbItemParamItem
tbItemParamItem){

        return

this.itemParamItemService.insertTbItemParamItem(tbItemParamItem);

    }

}
```

---

#### 4.2.8.2 创建 service

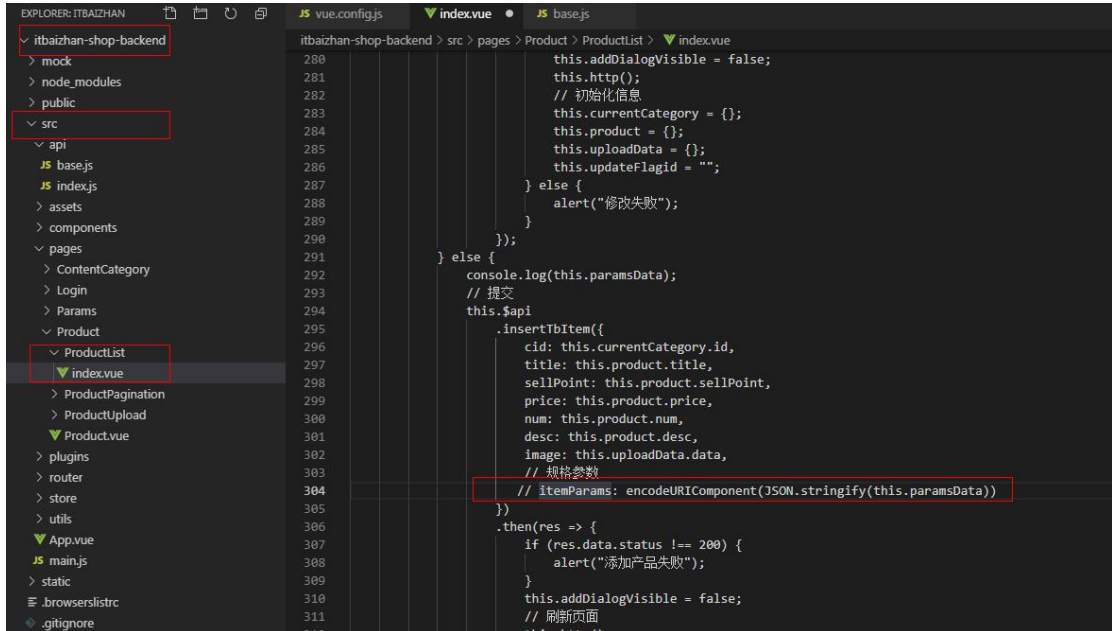
```
public interface ItemParamItemService {  
  
    Integer insertTbItemParamItem(TbItemParamItem tbItemParamItem);  
  
}
```

#### 4.2.8.3 创建 serviceImpl

```
@Service  
  
public class ItemParamItemServiceImpl implements ItemParamItemService {  
  
    @Autowired  
  
    private TbItemParamItemMapper tbItemParamItemMapper;  
  
    @Override  
  
    public Integer insertTbItemParamItem(TbItemParamItem tbItemParamItem) {  
  
        return this.tbItemParamItemMapper.insert(tbItemParamItem);  
  
    }  
  
}
```

## 4.2.9 在 backend\_item 服务中实现商品添加

### 4.2.9.1 修改前台代码注释掉商品规格参数



### 4.2.9.2 修改 controller

```
/**
 * 添加商品
 */

@RequestMapping("/insertTbItem")
public Result insertTbItem(TbItem tbItem,String desc,String itemParams){

    try{

        return this.itemService.insertTbItem(tbItem,desc,itemParams);

    }catch (Exception e){

        e.printStackTrace();

    }
}
```

```
        return Result.build(500,"error");  
    }  
}
```

#### 4.2.9.3 修改 service

```
Result insertTbItem(TbItem tbItem,String desc,String itemParams);
```

#### 4.2.9.4 修改 serviceImpl

```
/**  
 * 添加 TbItem , 添加 TbitemDesc , 添加 TbItemParamItem  
 * @param tbItem  
 * @param desc  
 * @param itemParams  
 * @return  
 */  
  
@Override  
public Result insertTbItem(TbItem tbItem, String desc, String itemParams) {  
    //补齐 Tbitem 数据  
  
    Long itemId = IDUtils.genItemId();  
  
    Date date = new Date();
```

```
tbItem.setId(itemId);

tbItem.setStatus((byte)1);

tbItem.setUpdated(date);

tbItem.setCreated(date);

Integer tbItemNum = this.commonItemFeignClient.insertTbItem(tbItem);

//补齐商品描述对象

TbItemDesc tbItemDesc = new TbItemDesc();

tbItemDesc.setItemId(itemId);

tbItemDesc.setItemDesc(desc);

tbItemDesc.setCreated(date);

tbItemDesc.setUpdated(date);

Integer tbitemDescNum =

this.commonItemFeignClient.insertItemDesc(tbItemDesc);

//补齐商品规格参数

TbItemParamItem tbItemParamItem = new TbItemParamItem();

tbItemParamItem.setItemId(itemId);

tbItemParamItem.setParamData(itemParams);

tbItemParamItem.setUpdated(date);

tbItemParamItem.setCreated(date);

Integer itemParamItmeNum =
```



```
this.commonItemFeignClient.insertTbItemParamItem(tbItemParamItem);

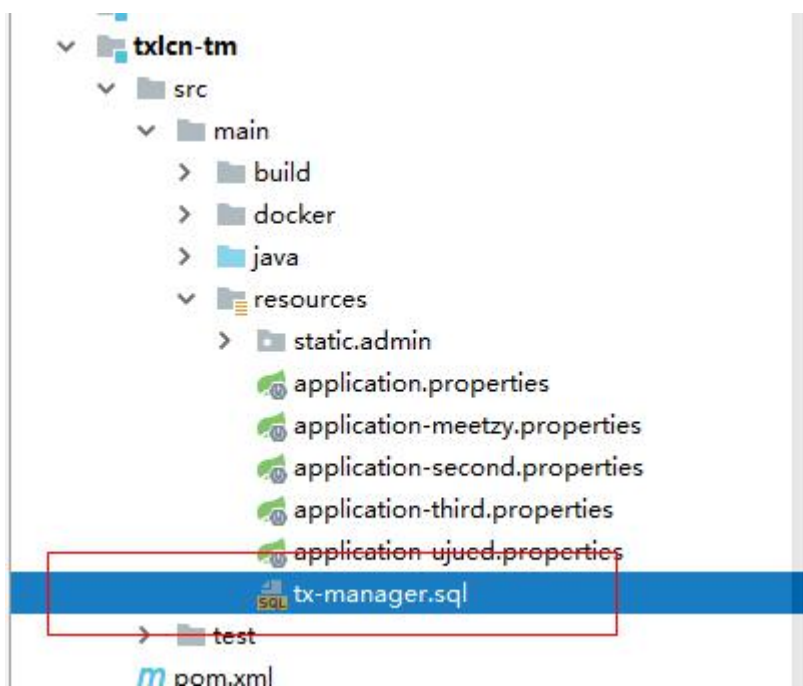
return Result.ok();
}
```

## 4.2.10 添加分布式事务 TX-LCN

### 4.2.10.1 搭建 TX-LCN 服务端

注意：在 Spring Boot2.x 版本中要求 TX-LCN 必须是 5.0 以上

### 4.2.10.2 向数据库中导入 SQL 文件



### 4.2.10.3 修改 application.properties 文件

```
application.properties x tx-manager.sql x
1 spring.application.name=TransactionManager
2 server.port=7970
3 spring.datasource.driver-class-name=com.mysql.jdbc.Driver
4 #spring.datasource.url=jdbc:mysql://192.168.1.115:3306/tx-manager?chara
5 spring.datasource.url=jdbc:mysql://192.168.1.101:3306/tx-manager?chara
6 spring.datasource.username=root
7 spring.datasource.password=root
8 spring.jpa.database-platform=org.hibernate.dialect.MySQL5InnoDBDialect
9 #第一次运行项目，初始化用create创建表，以后用none或者update
10 spring.jpa.hibernate.ddl-auto=none
11
12 #TxManager Ip, 默认为127.0.0.1
13 tx-lcn.manager.host=121.42.14.194
14 #TM监听socket端口，默认为8070
```

### 4.2.10.4 配置 TX-managerID。Tx 的服务端在哪个环境上运行

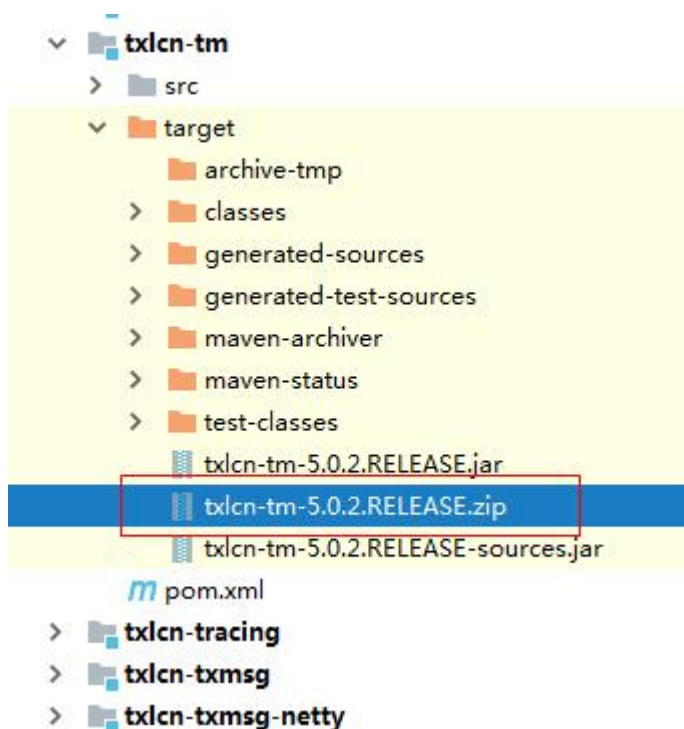
就配置那个环境的 IP

```
#TxManager Ip, 默认为127.0.0.1
tx-lcn.manager.host=192.168.70.157
#TM监听socket端口，默认为8070
```

### 4.2.10.5 配置 Redis 地址

```
#redis地址
#spring.redis.host=192.168.1.122
spring.redis.host=192.168.70.157
#redis端口
spring.redis.port=6379
#redis密码
#spring.redis.password=123456
```

#### 4.2.10.6 将 TX 服务端上传到 Linux 中



#### 4.2.10.7 通过 server.sh 启动脚本启动服务端

```
-rw-r--r--. 1 root root 345 Feb 22 16:12 application-meetzy.properties
-rw-r--r--. 1 root root 1895 Aug 23 2019 application.properties
-rw-r--r--. 1 root root 435 Feb 22 16:12 application-second.properties
-rw-r--r--. 1 root root 400 Feb 22 16:12 application-third.properties
-rw-r--r--. 1 root root 413 Feb 22 16:12 application-ujued.properties
drwxr-xr-x. 2 root root 4096 Aug 23 2019 lib
-rwxr-xr-x. 1 root root 2194 Aug 23 00:39 server.sh
drwxr-xr-x. 3 root root 4096 Aug 23 2019 static
-rw-r--r--. 1 root root 703909 Aug 23 2019 txlcn-tm-5.0.2.RELEASE.jar
-rw-r--r--. 1 root root 676953 Aug 23 2019 txlcn-tm-5.0.2.RELEASE-sources.jar
-rw-r--r--. 1 root root 1439 Feb 22 16:12 tx-manager.sql
[root@localhost txlcn-tm-5.0.2.RELEASE]# vim server.sh
[root@localhost txlcn-tm-5.0.2.RELEASE]# ./server.sh start
Starting the txlcn-tm-5.0.2.RELEASE.jar...
txlcn-tm-5.0.2.RELEASE.jar Started and the PID is 7555.
You can check the log file in /usr/local/tx-lcn/txlcn-tm-5.0.2.RELEASE/logs/txlcn-tm-5.0.2.RELEASE.jar for details.
[root@localhost txlcn-tm-5.0.2.RELEASE]#
```

## 4.2.10.8 访问管理页面

Eureka

egoshop

TxManager系统后台

+

→ ↺ ⚠ 不安全 | 192.168.70.157:7970/admin/index.html#/

TxManager系统后台

🏠 首页

⚙ 异常记录

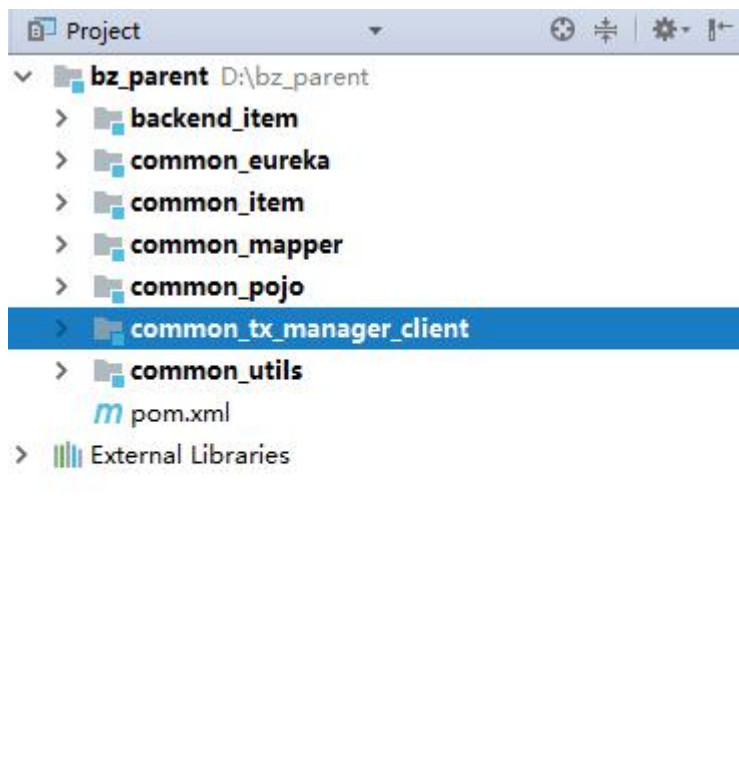
☰ 系统日志

配置信息

名称	值
TM主机IP	192.168.70.157
TM事务消息端口	8070
TM心跳时间	15000 ms
已注册的TC	0 <a href="#">查看详情</a>
事务并发处理等级	150
分布式事务时间	30000 ms
事务异常通知	disabled
允许系统日志	false
TM版本号	5.0.2.RELEASE

## 4.2.11 创建 TX-LCN 客户端

### 4.2.11.1 创建 TX-LCN 客户端项目



### 4.2.11.2 修改 POM 文件添加依赖

```
<?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
http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <parent>

        <artifactId>bz_parent</artifactId>

        <groupId>com.bjsxt</groupId>
```

```
<version>1.0-SNAPSHOT</version>
```

```
</parent>
```

```
<modelVersion>4.0.0</modelVersion>
```

```
<artifactId>common_tx_manager_client</artifactId>
```

```
<dependencies>
```

```
<dependency>
```

```
<groupId>com.codingapi.txlcn</groupId>
```

```
<artifactId>txlcn-tc</artifactId>
```

```
<version>5.0.2.RELEASE</version>
```

```
</dependency>
```

```
<dependency>
```

```
<groupId>com.codingapi.txlcn</groupId>
```

```
<artifactId>txlcn-txmsg-netty</artifactId>
```

```
<version>5.0.2.RELEASE</version>
```

```
</dependency>
```

```
</dependencies>
```

```
</project>
```

---

## 4.2.12 在服务中使用 TX-LCN 做分布式事务处理

### 4.2.12.1 在 common\_item 服务中添加 TX-LCN

```
<dependency>  
  <groupId>com.bjsxt</groupId>  
  <artifactId>common_tx_manager_client</artifactId>  
  <version>1.0-SNAPSHOT</version>  
</dependency>
```

#### 4.2.12.1.1 修改配置文件，添加 TX-LCN 服务端地址

```
tx-lcn:  
  
  client:  
  
    manager-address: 192.168.70.157:8070
```

#### 4.2.12.1.2 修改启动类，开启 TX-LCN

```
@EnableDistributedTransaction
```

#### 4.2.12.1.3 在方法上添加分布式事务处理注解

```
@LcnTransaction
```

#### 4.2.12.2 在 backend\_item 服务中添加 TX-LCN

```
<dependency>  
  <groupId>com.bjsxt</groupId>  
  <artifactId>common_tx_manager_client</artifactId>  
  <version>1.0-SNAPSHOT</version>  
</dependency>  
  
<!--mapper-->  
  
<dependency>  
  <groupId>com.bjsxt</groupId>  
  <artifactId>common_mapper</artifactId>  
  <version>1.0-SNAPSHOT</version>  
</dependency>
```

#### 4.2.12.2.1 修改配置文件，添加 TX-LCN 服务端地址

```
tx-lcn:
```

```
  client:
```



---

**manager-address:** 192.168.70.157:8070

#### 4.2.12.2.2 修改启动类，开启 TX-LCN

`@EnableDistributedTransaction`

#### 4.2.12.2.3 在方法上添加分布式事务处理注解

`@LcnTransaction`

### 4.3 实现删除商品接口

#### 4.3.1 在 common\_item 服务中实现删除商品

##### 4.3.1.1 修改 controller

```
/**
 * 删除商品
 */
@RequestMapping("/deleteItemById")
public Integer deleteItemById(@RequestBody TbItem tbItem){

    return this.itemService.updateItemById(tbItem);
}
```

---

#### 4.3.1.2 修改 service

```
Integer updateItemById(TbItem tbItem);
```

#### 4.3.1.3 修改 serviceImpl

```
/**
 * 更新与删除商品
 * 删除是更新 status 字段的值，修改为 3
 * @param tbItem
 * @return
 */
@Override
@LcnTransaction
public Integer updateItemById(TbItem tbItem) {
    tbItem.setUpdated(new Date());
    return this.tbItemMapper.updateByPrimaryKeySelective(tbItem);
}
```

---

## 4.3.2 在 backend\_item 服务中实现删除商品

### 4.3.2.1 修改 controller

```
/**
 * 删除商品
 */
@RequestMapping("/deleteItemById")
public Result deleteItemById(Long itemId){

    try{

        return this.itemService.deleteItemById(itemId);

    }catch (Exception e){

        e.printStackTrace();

    }

    return Result.build(500,"error");
}
```

### 4.3.2.2 修改 service

```
Result deleteItemById(Long itemId);
```

### 4.3.2.3修改 serviceImpl

```
/**
 * 删除商品
 * @param itemId
 * @return
 */
@Override
@LcnTransaction
public Result deleteItemById(Long itemId) {

    TbItem item = new TbItem();

    item.setId(itemId);

    item.setStatus((byte)3);

    Integer itemNum = this.commonItemFeignClient.deleteItemById(item);

    if(itemNum != null){

        return Result.ok();

    }

    return Result.error("删除失败");
}
```

#### 4.3.2.4修改 feignClient

```
@PostMapping("/service/item/deleteItemById")  
  
Integer deleteItemById(@RequestBody TbItem tbItem);
```

### 4.4 实现预更新商品接口

#### 4.4.1 在 common\_item 服务中实现根据 ID 查询商品

##### 4.4.1.1修改 controller

```
/**  
 * 根据商品 ID 查询商品，商品分类，商品描述，商品规格参数  
 */  
  
@RequestMapping("/preUpdateItem")  
  
public Map<String,Object> preUpdateItem(Long itemId){  
  
    return this.itemService.preUpdateItem(itemId);  
  
}
```

##### 4.4.1.2修改 service

```
Map<String,Object> preUpdateItem(Long itemId);
```

#### 4.4.1.3 修改 serviceImpl

```
/**
 * 根据商品 ID 查询商品所有信息。
 * @param itemId
 * @return
 */
@Override
public Map<String, Object> preUpdateItem(Long itemId) {

    Map<String, Object> map = new HashMap<>();

    //根据商品 ID 查询商品
    TbItem item = this.tbItemMapper.selectByPrimaryKey(itemId);

    map.put("item", item);

    //根据商品 ID 查询商品描述
    TbItemDesc itemDesc = this.tbItemDescMapper.selectByPrimaryKey(itemId);

    map.put("itemDesc", itemDesc);

    //根据商品 ID 查询商品类目
    TbItemCat itemCat =

this.tbItemCatMapper.selectByPrimaryKey(item.getCid());

    map.put("itemCat", itemCat);
}
```

```
//根据商品 ID 查询商品规格参数

TbItemParamItemExample example = new TbItemParamItemExample();

TbItemParamItemExample.Criteria criteria = example.createCriteria();

criteria.andItemIdEqualTo(itemId);

List<TbItemParamItem> list =

this.tbItemParamItemMapper.selectByExampleWithBLOBs(example);

if(list != null && list.size() > 0){

    map.put("itemParamItem",list.get(0));

}

return map;

}
```

## 4.4.2 在 backend\_item 服务中实现商品预更新查询

### 4.4.2.1 修改 controller

```
/**

 * 预更新商品查询

 */

@RequestMapping("/preUpdateItem")

public Result preUpdateItem(Long itemId){
```

```
try{

    return this.itemService.preUpdateItem(itemId);

}catch (Exception e){

    e.printStackTrace();

}

return Result.build(500,"error");

}
```

#### 4.4.2.2修改 service

```
Result preUpdateItem(Long itemId);
```

#### 4.4.2.3修改 serviceImpl

```
/**
 * 预更新商品查询
 * @param itemId
 * @return
 */
@Override
public Result preUpdateItem(Long itemId) {
```



```
Map<String,Object> map =  
  
this.commonItemFeignClient.preUpdateItem(itemId);  
  
if(map != null){  
  
    return Result.ok(map);  
  
}  
  
return Result.error("查无结果");  
  
}
```

#### 4.4.2.4修改 feignClient

```
@PostMapping("/service/item/preUpdateItem")  
  
Map<String,Object> preUpdateItem(@RequestParam("itemId") Long itemId);
```

#### 4.4.2.5解决 itemCat 显示不正确以及无法显示 itemDesc 问题

修改 common\_item 的 serviceImpl 代码

```
/**  
  
* 根据商品 ID 查询商品所有信息。  
  
* @param itemId  
  
* @return
```

---

*\*/*

**@Override**

```
public Map<String, Object> preUpdateItem(Long itemId) {  
  
    Map<String,Object> map = new HashMap<>();  
  
    //根据商品 ID 查询商品  
  
    TbItem item = this.tbItemMapper.selectByPrimaryKey(itemId);  
  
    map.put("item",item);  
  
  
    //根据商品 ID 查询商品描述  
  
    TbItemDesc itemDesc = this.tbItemDescMapper.selectByPrimaryKey(itemId);  
  
    map.put("itemDesc",itemDesc.getItemDesc());  
  
  
    //根据商品 ID 查询商品类目  
  
    TbItemCat itemCat =  
  
this.tbItemCatMapper.selectByPrimaryKey(item.getCid());  
  
    map.put("itemCat",itemCat.getName());  
  
  
    //根据商品 ID 查询商品规格参数  
  
    TbItemParamItemExample example = new TbItemParamItemExample();  
  
    TbItemParamItemExample.Criteria criteria = example.createCriteria();  
  
    criteria.andItemIdEqualTo(itemId);  
  
    List<TbItemParamItem> list =
```

```
this.tbItemParamItemMapper.selectByExampleWithBLOBs(example);

    if(list != null && list.size() > 0){

        map.put("itemParamItem",list.get(0));

    }

    return map;
}
```

## 4.5 实现商品更新接口

### 4.5.1 在 common\_item 服务中实现商品更新

#### 4.5.1.1 修改 ItemController

```
/**
 * 对商品表的更新操作
 *
 */

@RequestMapping("/upateTbitem")

public Integer updateTbitem(@RequestBody TbItem tbItem){

    return this.itemService.updateItemById(tbItem);

}
```

---

#### 4.5.1.2修改 ItemDescController

```
/**
 * 更新商品描述
 */
@RequestMapping("/updateItemDesc")
public Integer updateItemDesc(@RequestBody TbItemDesc tbItemDesc){

    return this.itemDescService.updateItemDesc(tbItemDesc);

}
```

#### 4.5.1.3修改 ItemParamItemController

```
/**
 * 更新商品规格参数
 */
@RequestMapping("/upateItemParamItem")
public Integer upateItemParamItem(@RequestBody TbItemParamItem
tbItemParamItem){

    return this.itemParamItemService.upateItemParamItem(tbItemParamItem);

}
```

---

#### 4.5.1.4修改 ItemDescService

```
Integer updateItemDesc(TbItemDesc tbItemDesc);
```

#### 4.5.1.5修改 ItemParamItemService

```
Integer upateItemParamItem(TbItemParamItem tbItemParamItem);
```

#### 4.5.1.6修改 ItemDescServiceImpl

```
/**
 * 更新商品描述
 * @param tbItemDesc
 * @return
 */
@Override
@LcnTransaction
public Integer updateItemDesc(TbItemDesc tbItemDesc) {
    tbItemDesc.setUpdated(new Date());
    return this.tbItemDescMapper.updateByPrimaryKeySelective(tbItemDesc);
}
```

---

#### 4.5.1.7 修改 ItemParamItemServiceImpl

```
@Override
@LcnTransaction
public Integer upateItemParamItem(TbItemParamItem tbItemParamItem) {
    tbItemParamItem.setUpdated(new Date());
    TbItemParamItemExample example = new TbItemParamItemExample();
    TbItemParamItemExample.Criteria criteria = example.createCriteria();
    criteria.andItemIdEqualTo(tbItemParamItem.getItemId());

    return
    this.tbItemParamItemMapper.updateByExampleSelective(tbItemParamItem,exam
    ple);
}
```

#### 4.5.2 在 backend\_item 服务中实现商品更新

##### 4.5.2.1 修改 controller

```
/**
 * 商品更新
 */
```

```
@RequestMapping("/updateTbItem")

public Result updateTbItem(TbItem tbItem,String desc,String itemParams){

    try{

        return this.itemService.updateTbItem(tbItem,desc,itemParams);

    }catch (Exception e){

        e.printStackTrace();

    }

    return Result.build(500,"error");

}
```

#### 4.5.2.2修改 service

```
Result updateTbItem(TbItem tbItem,String desc,String itemParams);
```

#### 4.5.2.3修改 serviceImpl

```
/**

 * 更新商品：更新 TbItem 表，更新 TbitemDesc 表，更新 TbItemParamItem 表

 * @param tbItem

 * @param desc

 * @param itemParams
```

---

*\* @return*

*\*/*

@Override

@LcnTransaction

**public** Result updateTbItem(TbItem tbItem, String desc, String itemParams) {

*//更新商品*

Integer itemNum = **this.commonItemFeignClient**.updateTbitem(tbItem);

*//更新商品描述*

TbItemDesc tbItemDesc = **new** TbItemDesc();

tbItemDesc.setItemId(tbItem.getId());

tbItemDesc.setItemDesc(desc);

Integer itemDescNum =

**this.commonItemFeignClient**.updateItemDesc(tbItemDesc);

*//更新商品规格参数*

TbItemParamItem tbItemParamItem = **new** TbItemParamItem();

tbItemParamItem.setParamData(itemParams);

tbItemParamItem.setItemId(tbItem.getId());

Integer itemParamItemNum =

**this.commonItemFeignClient**.upateItemParamItem(tbItemParamItem);



```
return Result.ok();
}
```

#### 4.5.2.4 修改 feignClient

```
@PostMapping("/service/item/upateTbitem")

Integer updateTbitem(@RequestBody TbItem tbItem);

@PostMapping("/service/itemDesc/updateItemDesc")

Integer updateItemDesc(@RequestBody TbItemDesc tbItemDesc);

@PostMapping("/service/itemParamItem/upateItemParamItem")

Integer upateItemParamItem(@RequestBody TbItemParamItem tbItemParamItem);
```

## 5 开发规格参数管理接口

### 5.1 实现查询所有规格参数模板接口

### 5.1.1 在 common\_item 服务中实现查询规格参数

#### 5.1.1.1 修改 controller

\* [查询所有商品规格参数模板](#)

```
*/  
  
@RequestMapping("/selectItemParamAll")  
  
public PageResult selectItemParamAll(@RequestParam Integer  
page, @RequestParam Integer rows){  
  
    return this.itemParamService.selectItemParamAll(page, rows);  
  
}
```

#### 5.1.1.2 修改 service

```
PageResult selectItemParamAll(Integer page, Integer rows);
```

#### 5.1.1.3 修改 serviceImpl

```
/**  
  
 * 查询所有规格参数模板  
  
 * @param page  
  
 * @param rows  
  
 * @return  
  
 */  
  
@Override  
  
public PageResult selectItemParamAll(Integer page, Integer rows) {
```

```
PageHelper.startPage(page,rows);

TbItemParamExample example = new TbItemParamExample();

List<TbItemParam> list =

this.tbItemParamMapper.selectByExampleWithBLOBs(example);


PageInfo<TbItemParam> pageInfo = new PageInfo<>(list);


PageResult pageResult = new PageResult();

pageResult.setPageIndex(page);

pageResult.setResult(pageInfo.getList());

pageResult.setTotalPage(pageInfo.getTotal());

return pageResult;

}
```

## 5.1.2 在 backend\_item 服务中实现查询规格参数

### 5.1.2.1 修改 controller

```
/**
 * 查询商品全部规格参数
 */

@RequestMapping("/selectItemParamAll")

public Result selectItemParamAll(@RequestParam(defaultValue = "1") Integer
```

```
page, @RequestParam(defaultValue = "30") Integer rows){  
  
    try{  
  
        return this.itemParamService.selectItemParamAll(page, rows);  
  
    } catch (Exception e){  
  
        e.printStackTrace();  
  
    }  
  
    return Result.build(500, "error");  
}
```

#### 5.1.2.2 修改 service

```
Result selectItemParamAll(Integer page, Integer rows);
```

#### 5.1.2.3 修改 serviceImpl

```
/**  
  
 * 查询所有规格参数模板  
  
 * @param page  
  
 * @param rows  
  
 * @return  
  
 */
```

```
@Override
```

```
public Result selectItemParamAll(Integer page, Integer rows) {  
  
    PageResult pageResult =  
  
    this.commonItemFeignClient.selectItemParamAll(page,rows);  
  
    if(pageResult != null && pageResult.getResult().size() > 0){  
  
        return Result.ok(pageResult);  
  
    }  
  
    return Result.error("查无结果");  
  
}
```

#### 5.1.2.4修改 feignClient

```
@PostMapping("/service/itemParam/selectItemParamAll")  
  
PageResult selectItemParamAll(@RequestParam("page") Integer  
page,@RequestParam("rows") Integer rows);
```

### 5.2 实现为商品分类添加规格参数模板接口

#### 5.2.1 在 common\_item 服务中实现添加规格参数模板

##### 5.2.1.1修改 controller

```
/**
```

---

```
* 根据商品分类添加规格参数模板
```

```
*/
```

```
@RequestMapping("/insertItemParam")
```

```
public Integer insertItemParam(@RequestBody TbItemParam tbItemParam){
```

```
    return this.itemParamService.insertItemParam(tbItemParam);
```

```
}
```

### 5.2.1.2 修改 service

```
Integer insertItemParam(TbItemParam tbItemParam);
```

### 5.2.1.3 修改 serviceImpl

```
/**
```

```
* 添加商品分类规格参数模板
```

```
* @param tbItemParam
```

```
* @return
```

```
*/
```

```
@Override
```

```
@LcnTransaction
```

```
public Integer insertItemParam(TbItemParam tbItemParam) {  
  
    return this.tbItemParamMapper.insertSelective(tbItemParam);  
  
}
```

## 5.2.2 在 backend\_item 服务中实现添加规格参数模板

### 5.2.2.1 修改 controller

```
/**  
 * 添加商品分类规格参数模板  
 */  
  
@RequestMapping("/insertItemParam")  
  
public Result insertItemParam(Long itemCatId ,String paramData){  
  
    try{  
  
        return this.itemParamService.insertItemParam(itemCatId,paramData);  
  
    }catch (Exception e){  
  
        e.printStackTrace();  
  
    }  
  
    return Result.build(500,"error");  
  
}
```

---

### 5.2.2.2修改 service

```
Result insertItemParam(Long itemCatId ,String paramData);
```

### 5.2.2.3修改 serviceImpl

```
/**
 * 添加商品分类规格参数模板
 * @param itemCatId
 * @param paramData
 * @return
 */
@Override
@LcnTransaction
public Result insertItemParam(Long itemCatId, String paramData) {
    //创建 TbItemParam 对象并补齐数据
    TbItemParam tbItemParam = new TbItemParam();
    tbItemParam.setItemCatId(itemCatId);
    tbItemParam.setParamData(paramData);
    tbItemParam.setCreated(new Date());
    tbItemParam.setUpdated(new Date());
}
```



```
Integer itemParamNum =  
  
this.commonItemFeignClient.insertItemParam(tbItemParam);  
  
if(itemParamNum != null){  
    return Result.ok();  
}  
  
return Result.error("添加失败");  
  
}
```

#### 5.2.2.4修改 feignClient

```
@RequestMapping("/service/itemParam/insertItemParam")  
  
Integer insertItemParam(@RequestBody TbItemParam tbItemParam);
```

### 5.3 实现删除规格参数模板接口

#### 5.3.1 在 common\_item 服务中实现删除规格参数模板

##### 5.3.1.1修改 controller

```
/**  
 * 根据规格参数模板的 ID 删除数据  
 */
```

```
@RequestMapping("/deleteItemParamById")

public Integer deleteItemParamById(@RequestParam Long id){

    return this.itemParamService.deleteItemParamById(id);

}
```

### 5.3.1.2 修改 service

```
Integer deleteItemParamById(Long id);
```

### 5.3.1.3 修改 serviceImpl

```
/**
 * 根据 ID 删除规格参数模板
 * @param id
 * @return
 */

@Override

public Integer deleteItemParamById(Long id) {

    return this.tbItemParamMapper.deleteByPrimaryKey(id);

}
```

---

## 5.3.2 在 backend\_item 服务中实现删除规格参数模板

### 5.3.2.1修改 controller

```
/**
 * 删除规格参数模板
 */
@RequestMapping("/deleteItemParamById")
public Result deleteItemParamById(Long id){
    try{
        return this.itemParamService.deleteItemParamById(id);
    }catch (Exception e){
        e.printStackTrace();
    }
    return Result.build(500,"error");
}
```

### 5.3.2.2修改 service

```
Result deleteItemParamById(Long id);
```

### 5.3.2.3修改 serviceImpl

```
/**
 * 根据规格参数模板的 ID 删除规格参数模板
 *
 * @param id
 *
 * @return
 */
@Override
public Result deleteItemParamById(Long id) {

    Integer itemParamNum =

    this.commonItemFeignClient.deleteItemParamById(id);

    if(itemParamNum != null && itemParamNum > 0){

        return Result.ok();

    }

    return Result.error("删除失败");

}
```

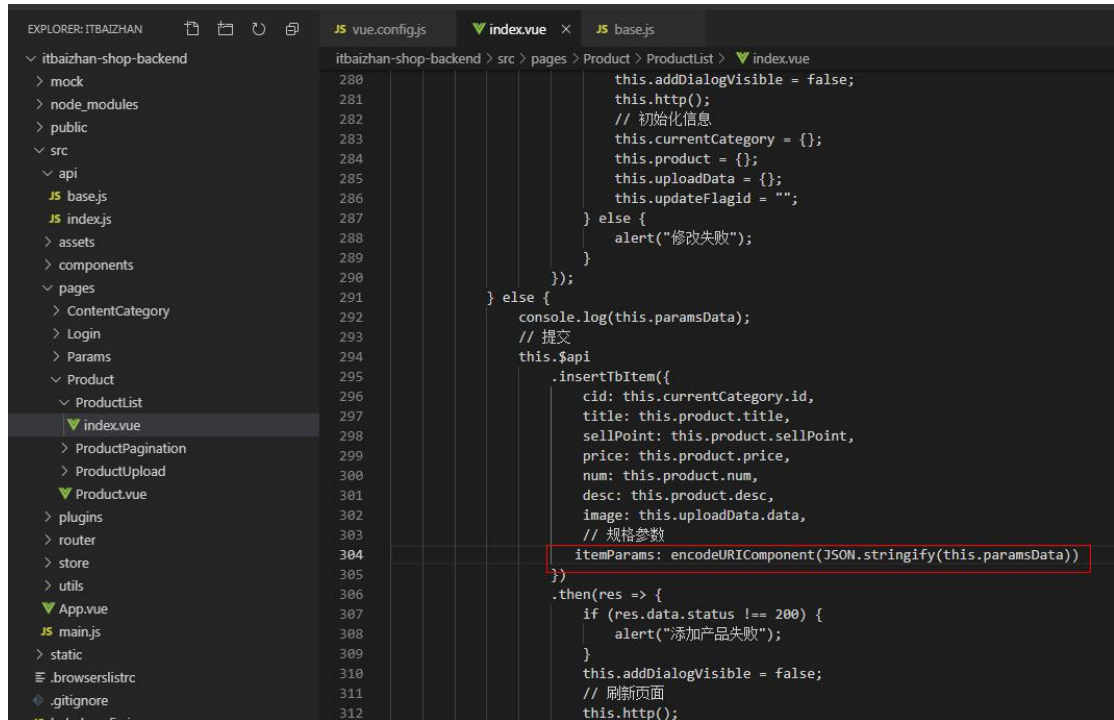
### 5.3.2.4修改 feignClient

```
@RequestMapping("/service/itemParam/deleteItemParamById")

Integer deleteItemParamById(@RequestParam("id") Long id);
```

## 5.3.3 测试添加商品与更新商品完整流程

### 5.3.3.1 修改前端页面去掉注释

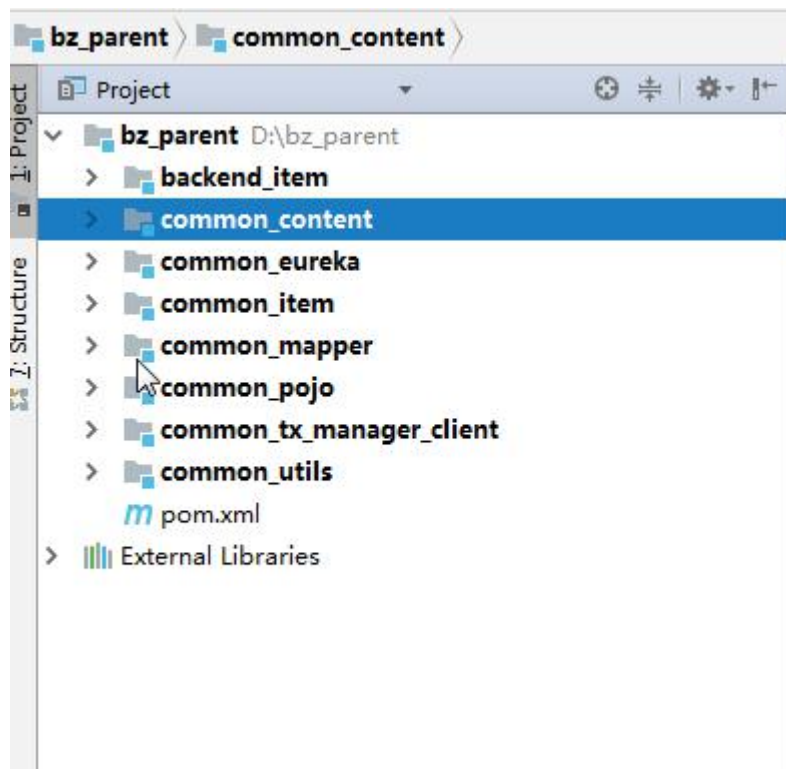


```
EXPLOSER: ITBAIZHAN  vue.config.js  index.vue  base.js
itbaizhan-shop-backend
├── mock
├── node_modules
├── public
├── src
│   ├── api
│   ├── base.js
│   ├── index.js
│   ├── assets
│   ├── components
│   ├── pages
│   │   ├── ContentCategory
│   │   ├── Login
│   │   ├── Params
│   │   ├── Product
│   │   │   ├── ProductList
│   │   │   │   └── index.vue
│   │   ├── ProductPagination
│   │   ├── ProductUpload
│   │   └── Product.vue
│   ├── plugins
│   ├── router
│   ├── store
│   ├── utils
│   ├── App.vue
│   ├── main.js
│   ├── static
│   ├── .browserslistrc
│   ├── .gitignore
│   └── babel.config.js
└── ...

280 this.addDialogVisible = false;
281 this.http();
282 // 初始化信息
283 this.currentCategory = {};
284 this.product = {};
285 this.uploadData = {};
286 this.updateFlagid = "";
287 } else {
288     alert("修改失败");
289 }
290 });
291 } else {
292     console.log(this.paramsData);
293     // 提交
294     this.$api
295     .insertTbItem({
296         cid: this.currentCategory.id,
297         title: this.product.title,
298         sellPoint: this.product.sellPoint,
299         price: this.product.price,
300         num: this.product.num,
301         desc: this.product.desc,
302         image: this.uploadData.data,
303         // 规格参数
304         itemParams: encodeURIComponent(JSON.stringify(this.paramsData))
305     })
306     .then(res => {
307         if (res.data.status !== 200) {
308             alert("添加产品失败");
309         }
310         this.addDialogVisible = false;
311         // 刷新页面
312         this.http();
```

## 6 创建 common\_content 服务

### 6.1 创建项目



### 6.2 修改 POM 文件，添加依赖坐标

```
<?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
http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <parent>

        <artifactId>bz_parent</artifactId>

        <groupId>com.bjsxt</groupId>

        <version>1.0-SNAPSHOT</version>
```

---

```
</parent>
```

```
<modelVersion>4.0.0</modelVersion>
```

```
<artifactId>common_content</artifactId>
```

```
<dependencies>
```

```
  <!--mapper-->
```

```
  <dependency>
```

```
    <groupId>com.bjsxt</groupId>
```

```
    <artifactId>common_mapper</artifactId>
```

```
    <version>1.0-SNAPSHOT</version>
```

```
  </dependency>
```

```
  <!--utils-->
```

```
  <dependency>
```

```
    <groupId>com.bjsxt</groupId>
```

```
    <artifactId>common_utils</artifactId>
```

```
    <version>1.0-SNAPSHOT</version>
```

```
  </dependency>
```

```
  <!--Spring Boot Web Starter-->
```

```
  <dependency>
```

```
    <groupId>org.springframework.boot</groupId>
```

```
<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<!--Spring Cloud Eureka Client Starter-->

<dependency>

    <groupId>org.springframework.cloud</groupId>

    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>

</dependency>

<dependency>

    <groupId>com.bjsxt</groupId>

    <artifactId>common_tx_manager_client</artifactId>

    <version>1.0-SNAPSHOT</version>

</dependency>

</dependencies>

</project>
```

### 6.3 创建配置文件，添加相关配置

**spring:**

**application:**

**name:** common-content



---

**datasource:**

**driverClassName:** com.mysql.jdbc.Driver

**url:** jdbc:mysql://localhost:3306/bz\_shop?characterEncoding=UTF-8

**username:** root

**password:** root

**type:** com.alibaba.druid.pool.DruidDataSource

**server:**

**port:** 9020

**eureka:**

**client:**

**serviceUrl:**

**defaultZone:** http://eureka-server:8761/eureka/

**tx-lcn:**

**client:**

**manager-address:** 192.168.70.157:8070

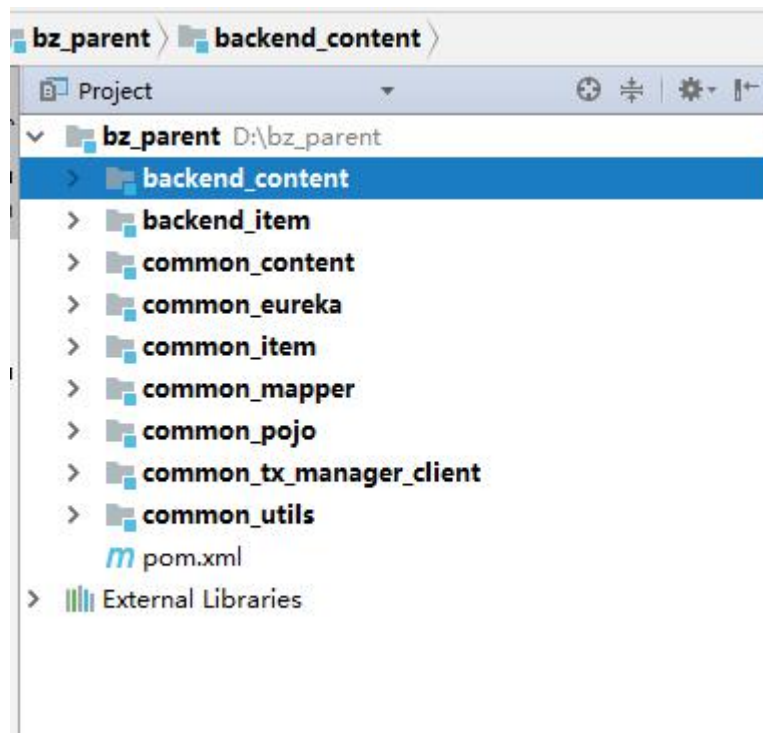
---

## 6.4 创建启动类，添加相关注解

```
/**  
 * CommonContent 启动类  
 */  
  
@SpringBootApplication  
@EnableDiscoveryClient  
@EnableDistributedTransaction  
@MapperScan("com.bjsxt.mapper")  
  
public class CommonCotentApplication {  
    public static void main(String[] args){  
        SpringApplication.run(CommonCotentApplication.class,args);  
    }  
}
```

## 7 创建 backend\_content 服务

### 7.1 创建项目



### 7.2 修改 POM 文件，添加依赖坐标

```
<?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
http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <parent>

    <artifactId>bz_parent</artifactId>

    <groupId>com.bjsxt</groupId>

    <version>1.0-SNAPSHOT</version>
```

---

```
</parent>
```

```
<modelVersion>4.0.0</modelVersion>
```

```
<artifactId>backend_content</artifactId>
```

```
<dependencies>
```

```
<!--pojo-->
```

```
<dependency>
```

```
<groupId>com.bjsxt</groupId>
```

```
<artifactId>common_pojo</artifactId>
```

```
<version>1.0-SNAPSHOT</version>
```

```
</dependency>
```

```
<!--utils-->
```

```
<dependency>
```

```
<groupId>com.bjsxt</groupId>
```

```
<artifactId>common_utils</artifactId>
```

```
<version>1.0-SNAPSHOT</version>
```

```
</dependency>
```

```
<!--Spring Boot Web Starter-->
```

```
<dependency>
```

```
<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<!--Spring Cloud Eureka Client Starter-->

<dependency>

    <groupId>org.springframework.cloud</groupId>

    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>

</dependency>

<!--Spring Cloud OpenFeign Starter-->

<dependency>

    <groupId>org.springframework.cloud</groupId>

    <artifactId>spring-cloud-starter-openfeign</artifactId>

</dependency>

<dependency>

    <groupId>com.bjsxt</groupId>

    <artifactId>common_tx_manager_client</artifactId>

    <version>1.0-SNAPSHOT</version>

</dependency>

<!--mapper-->

<dependency>
```

```
<groupId>com.bjsxt</groupId>

<artifactId>common_mapper</artifactId>

<version>1.0-SNAPSHOT</version>

</dependency>

</dependencies>

</project>
```

### 7.3 创建配置文件，添加相关配置

**spring:**

**application:**

**name:** backend-content

**server:**

**port:** 9021

**eureka:**

**client:**

**serviceUrl:**

**defaultZone:** http://eureka-server:8761/eureka/

---

**tx-lcn:**

**client:**

**manager-address:** 192.168.70.157:8070

## 7.4 创建启动类，添加相关注解

```
/**
 * BackendContent 启动类
 */
@SpringBootApplication
@EnableDiscoveryClient
@EnableFeignClients
@EnableDistributedTransaction

public class BackendContentApplication {

    public static void main(String[] args){

        SpringApplication.run(BackendContentApplication.class,args);

    }

}
```

## 8 实现内容分类管理接口

### 8.1 实现内容分类查询接口

#### 8.1.1 在 common\_content 服务中实现查询内容分类

##### 8.1.1.1 添加 controller

```
/**
 * ContentCategoryController
 */
@RestController
@RequestMapping("/service/contentCategory")
public class ContentCategoryController {

    @Autowired
    private ContentCategoryService contentCategoryService;

    /**
     * 根据父节点 ID 查询子节点
     */
    @RequestMapping("/selectContentCategoryByParentId")
    public List<TbContentCategory>
selectContentCategoryByParentId(@RequestParam Long parentId ){

        return
```



```
this.contentCategoryService.selectContentCategoryByParentId(parentId);

    }

}
```

### 8.1.1.2 添加 service

```
public interface ContentCategoryService {

    List<TbContentCategory> selectContentCategoryByParentId(Long parentId );

}
```

### 8.1.1.3 添加 serviceImpl

```
/**
 * 内容分类业务层
 */

@Service

public class ContentCategoryServiceImpl implements ContentCategoryService {

    @Autowired
```

```
private TbContentCategoryMapper tbContentCategoryMapper;

/**
 * 根据父节点 Id 查询子节点
 * @param parentId
 * @return
 */
@Override
public List<TbContentCategory> selectContentCategoryByParentId(Long
parentId) {
    TbContentCategoryExample tbContentCategoryExample = new
TbContentCategoryExample();
    TbContentCategoryExample.Criteria criteria =
tbContentCategoryExample.createCriteria();
    criteria.andParentIdEqualTo(parentId);
    List<TbContentCategory> list =
this.tbContentCategoryMapper.selectByExample(tbContentCategoryExample);
    return list;
}
}
```

## 8.1.2 在 backend\_content 服务中实现查询内容分类

### 8.1.2.1 添加 controller

```
@RestController

@RequestMapping("/content")

public class ContentCategoryController {

    @Autowired

    private ContentCategoryService contentCategoryService;

    /**
     * 根据当前节点 ID 查询子节点
     */

    @RequestMapping("/selectContentCategoryByParentId")

    public Result selectContentCategoryByParentId(@RequestParam(defaultValue
= "0") Long id){

        try{

            return

this.contentCategoryService.selectContentCategoryByParentId(id);

        }catch (Exception e){

            e.printStackTrace();

        }

    }

}
```

```
    }

    return Result.build(500,"error");

}

}
```

### 8.1.2.2 添加 service

```
public interface ContentCategoryService {

    Result selectContentCategoryByParentId(Long id);

}
```

### 8.1.2.3 添加 serviceImpl

```
@Service

public class ContentCategoryServiceImpl implements ContentCategoryService {

    @Autowired

    private CommonContentFeignClient commonContentFeignClient;

    /**

    * 查询内容分类

    */

}
```

```

    * @param id

    * @return

    */

    @Override

    public Result selectContentCategoryByParentId(Long id) {

        List<TbContentCategory> list =

this.commonContentFeignClient.selectContentCategoryByParentId(id);

        if(list != null && list.size() > 0){

            return Result.ok(list);

        }

        return Result.error("查无结果");

    }

}

```

#### 8.1.2.4 添加 feignClient

```

@FeignClient(value = "common-content")

public interface CommonContentFeignClient {

    //-----/service/contentCategory-----

    @PostMapping("/service/contentCategory/selectContentCategoryByParentId")

```

```
List<TbContentCategory>  
  
selectContentCategoryByParentId(@RequestParam("parentId") Long parentId );  
  
}
```

### 8.1.3 在 common\_content 服务中实现添加内容分类

#### 8.1.3.1 修改 controller

```
/**  
 * 添加内容分类  
 */  
  
@RequestMapping("/insertContentCategory")  
  
public Integer insertContentCategory(@RequestBody TbContentCategory  
tbContentCategory){  
  
    return  
  
this.contentCategoryService.insertContentCategory(tbContentCategory);  
}
```

#### 8.1.3.2 修改 service

```
Integer insertContentCategory(TbContentCategory tbContentCategory);
```

### 8.1.3.3 修改 serviceImpl

```
/**
 * 添加内容分类
 * @param tbContentCategory
 * @return
 */
@Override
@LcnTransaction
public Integer insertContentCategory(TbContentCategory tbContentCategory) {

    //补齐数据

    tbContentCategory.setUpdated(new Date());

    tbContentCategory.setCreated(new Date());

    tbContentCategory.setIsParent(false);

    tbContentCategory.setSortOrder(1);

    tbContentCategory.setStatus(1);

    //插入数据

    Integer contentCategoryNum =

this.tbContentCategoryMapper.insert(tbContentCategory);

    //查询当前新节点的父节点

    TbContentCategory contentCategory =
```

```
this.tbContentCategoryMapper.selectByPrimaryKey(tbContentCategory.getParentId());

//判断当前父节点是否是叶子节点

if(!contentCategory.getIsParent()){

    contentCategory.setIsParent(true);

    contentCategory.setUpdated(new Date());

    this.tbContentCategoryMapper.updateByPrimaryKey(contentCategory);

}

return contentCategoryNum;

}
```

## 8.1.4 在 backend\_content 服务中实现添加内容分类

### 8.1.4.1修改 controller

```
/**
 * 添加内容分类
 */
@RequestMapping("/insertContentCategory")
public Result insertContentCategory(TbContentCategory tbContentCategory){

    try{

        return
```



```
this.contentCategoryService.insertContentCategory(tbContentCategory);

    }catch (Exception e){

        e.printStackTrace();

    }

    return Result.build(500,"error");
}
```

#### 8.1.4.2修改 service

```
Result insertContentCategory(TbContentCategory tbContentCategory);
```

#### 8.1.4.3修改 serviceImpl

```
/**
 * 添加内容分类
 * @param tbContentCategory
 * @return
 */
@Override
@LcnTransaction
public Result insertContentCategory(TbContentCategory tbContentCategory) {
```

```
Integer contentCategoryNum =  
this.commonContentFeignClient.insertContentCategory(tbContentCategory);  
  
if(contentCategoryNum != null){  
    return Result.ok();  
}  
  
return Result.error("添加失败");  
}
```

#### 8.1.4.4修改 feignClient

```
@PostMapping("/service/contentCategory/insertContentCategory")  
  
Integer insertContentCategory(@RequestBody TbContentCategory  
tbContentCategory);
```

### 8.1.5 在 common\_content 服务中实现删除内容分类

#### 8.1.5.1修改 controller

```
/**  
 * 删除内容分类  
 */  
  
@RequestMapping("/deleteContentCategoryById")  
  
public Integer deleteContentCategoryById(@RequestParam Long categoryId){
```

```
return this.contentCategoryService.deleteContentCategoryId(categoryId);  
}
```

#### 8.1.5.2 修改 service

```
Integer deleteContentCategoryId(Long categoryId);
```

#### 8.1.5.3 修改 serviceImpl

```
@Override  
  
public Integer deleteContentCategoryId(Long categoryId) {  
  
    //查询当前节点  
  
    TbContentCategory currentCategory =  
    this.tbContentCategoryMapper.selectByPrimaryKey(categoryId);  
  
    //删除当前节点的子节点  
  
    Integer status = this.deleteNode(currentCategory);  
  
    //查询当前节点的父节点
```

```
TbContentCategory parentCategory =  
this.tbContentCategoryMapper.selectByPrimaryKey(currentCategory.getParentId  
());  
  
//查看当前节点是否有兄弟节点，决定是否修改父节点的状态。  
  
TbContentCategoryExample example = new TbContentCategoryExample();  
TbContentCategoryExample.Criteria criteria = example.createCriteria();  
criteria.andParentIdEqualTo(parentCategory.getId());  
  
List<TbContentCategory> list =  
this.tbContentCategoryMapper.selectByExample(example);  
  
if(list.size() == 0){  
    parentCategory.setIsParent(false);  
    parentCategory.setUpdated(new Date());  
    this.tbContentCategoryMapper.updateByPrimaryKey(parentCategory);  
}  
  
return 200;  
}  
  
/**  
 * 删除当前节点与子节点  
 */
```

```
private Integer deleteNode(TbContentCategory currentCategory){

    if(currentCategory.getIsParent()){

        //是父节点

        //查询当前节点所有的子节点

        TbContentCategoryExample example = new
TbContentCategoryExample();

        TbContentCategoryExample.Criteria criteria = example.createCriteria();

        criteria.andParentIdEqualTo(currentCategory.getId());

        List<TbContentCategory> list =

this.tbContentCategoryMapper.selectByExample(example);

        for(TbContentCategory children :list){

            this.deleteNode(children);

this.tbContentCategoryMapper.deleteByPrimaryKey(currentCategory.getId());

        }

    }else{

        //不是父节点

this.tbContentCategoryMapper.deleteByPrimaryKey(currentCategory.getId());

    }

    return 200;
}
```

```
}
```

## 8.1.6 在 backend\_content 服务中实现删除内容分类

### 8.1.6.1 修改 controller

```
/**
 * 删除内容分类
 *
 */
@RequestMapping("/deleteContentCategoryId")
public Result deleteContentCategoryId(Long categoryId){

    try{

        return

        this.contentCategoryService.deleteContentCategoryId(categoryId);

    }catch (Exception e){

        e.printStackTrace();

    }

    return Result.build(500,"error");
}
```

---

### 8.1.6.2修改 service

```
Result deleteContentCategoryId(Long categoryId);
```

### 8.1.6.3修改 serviceImpl

```
/**
 * 删除内容分类
 * @param categoryId
 * @return
 */
@Override
public Result deleteContentCategoryId(Long categoryId) {

    Integer status =

    this.commonContentFeignClient.deleteContentCategoryId(categoryId);

    if(status == 200){

        return Result.ok();

    }

    return Result.error("删除失败");

}
```

---

#### 8.1.6.4修改 feignClient

```
@PostMapping("/service/contentCategory/deleteContentCategoryById")  
  
public Integer deleteContentCategoryById(@RequestParam("categoryId") Long  
categoryId);
```

### 8.1.7 在 common\_content 中实现修改内容分类

#### 8.1.7.1修改 controller

```
/**  
 * 修改内容分类名称  
 */  
  
@RequestMapping("/updateContentCategory")  
  
public Integer updateContentCategory(@RequestBody TbContentCategory  
tbContentCategory){  
  
    return  
  
    this.contentCategoryService.updateContentCategory(tbContentCategory);  
}
```



---

### 8.1.7.2修改 service

```
Integer updateContentCategory(TbContentCategory tbContentCategory);
```

### 8.1.7.3修改 serviceImpl

```
/**
 * 修改内容分类名称
 * @param tbContentCategory
 * @return
 */
@Override
@LcnTransaction
public Integer updateContentCategory(TbContentCategory tbContentCategory) {
    tbContentCategory.setUpdated(new Date());

    return
    this.tbContentCategoryMapper.updateByPrimaryKeySelective(tbContentCategory);
}
```

---

## 8.1.8 在 backend\_content 服务中实现修改内容分类

### 8.1.8.1 修改 controller

```
/**
 * 修改内容分类
 */
@RequestMapping("/updateContentCategory")
public Result updateContentCategory(TbContentCategory tbContentCategory){
    try{
        return
this.contentCategoryService.updateContentCategory(tbContentCategory);
    }catch (Exception e){
        e.printStackTrace();
    }
    return Result.build(500,"error");
}
```

### 8.1.8.2 修改 service

```
Result updateContentCategory(TbContentCategory tbContentCategory);
```

### 8.1.8.3修改 serviceImpl

```
/**
 * 修改内容分类
 * @param tbContentCategory
 * @return
 */
@Override
@LcnTransaction
public Result updateContentCategory(TbContentCategory tbContentCategory) {
    Integer num =
    this.commonContentFeignClient.updateContentCategory(tbContentCategory);

    if(num != null){
        return Result.ok();
    }

    return Result.error("更新失败");
}
```

### 8.1.8.4修改 feignClient

```
@PostMapping("/service/contentCategory/updateContentCategory")
Integer updateContentCategory(@RequestBody TbContentCategory
```

```
tbContentCategory);
```

## 8.1.9 在 common\_content 服务中实现根据分类查询内容

### 8.1.9.1 创建 controller

```
@RestController
@RequestMapping("/service/content")

public class ContentController {

    @Autowired
    private ContentService contentService;

    /**
     * 根据分类查询内容
     */
    @RequestMapping("/selectTbContentAllByCategoryId")
    public PageResult selectTbContentAllByCategoryId(@RequestParam Integer
page, @RequestParam Integer rows, @RequestParam Long categoryId){

        return
this.contentService.selectTbContentAllByCategoryId(page, rows, categoryId);
    }
}
```

---

```
}
```

### 8.1.9.2 创建 service

```
public interface ContentService {  
  
    PageResult selectTbContentAllByCategoryId(Integer page, Integer rows, Long  
    categoryId);  
  
}
```

### 8.1.9.3 创建 serviceImpl

```
@Service  
  
public class ContentServiceImpl implements ContentService {  
  
    @Autowired  
  
    private TbContentMapper tbContentMapper;  
  
    @Override  
  
    public PageResult selectTbContentAllByCategoryId(Integer page, Integer rows,  
    Long categoryId) {  
  
        PageHelper.startPage(page,rows);
```

```
TbContentExample example = new TbContentExample();

TbContentExample.Criteria criteria = example.createCriteria();

criteria.andCategoryIdEqualTo(categoryId);

List<TbContent> list = this.tbContentMapper.selectByExample(example);

PageInfo<TbContent> pageInfo = new PageInfo<>(list);

PageResult result = new PageResult();

result.setPageIndex(page);

result.setTotalPage(pageInfo.getTotal());

result.setResult(pageInfo.getList());

return result;
}
}
```

## 8.1.10 在 backend\_content 服务中实现根据分类查询内容

### 8.1.10.1 创建 controller

```
@RestController

@RequestMapping("/content")

public class ContentController {
```

```
@Autowired

private ContentService contentService;

/**
 * 根据分类 ID 查询内容
 */

@RequestMapping("/selectTbContentAllByCategoryId")

public Result selectTbContentAllByCategoryId(@RequestParam(defaultValue
= "1") Integer page,@RequestParam(defaultValue = "30") Integer rows,Long
categoryId){

    try{

        return

this.contentService.selectTbContentAllByCategoryId(page,rows,categoryId);

    }catch (Exception e){

        e.printStackTrace();

    }

    return Result.build(500,"error");

}

}
```

---

### 8.1.10.2 创建 service

```
public interface ContentService {  
  
    Result selectTbContentAllByCategoryId(Integer page, Integer rows, Long  
categoryId);  
  
}
```

### 8.1.10.3 创建 serviceImpl

```
@Service  
  
public class ContentServiceImpl implements ContentService {  
  
    @Autowired  
  
    private CommonContentFeignClient commonContentFeignClient;  
  
    /**  
    * 根据分类 ID 查询分类内容  
    * @param page  
    * @param rows  
    * @param categoryId  
    * @return
```



```

    */

    @Override

    public Result selectTbContentAllByCategoryId(Integer page, Integer rows,
Long categoryId) {

        PageResult pageResult =

this.commonContentFeignClient.selectTbContentAllByCategoryId(page,rows,cate
goryId);

        if(pageResult != null && pageResult.getResult().size() > 0){

            return Result.ok(pageResult);

        }

        return Result.error("查无结果");

    }
}

```

#### 8.1.10.4 修改 feignClient

```

@PostMapping("/service/content/selectTbContentAllByCategoryId")

PageResult selectTbContentAllByCategoryId(@RequestParam("page") Integer
page, @RequestParam("rows") Integer rows, @RequestParam("categoryId") Long
categoryId);

```

---

## 8.1.11 在 common\_content 中实现根据分类添加内容

### 8.1.11.1 修改 controller

```
/**  
 * 根据分类添加内容  
 */  
  
@RequestMapping("/insertTbContent")  
public Integer insertTbContent(@RequestBody TbContent tbContent){  
    return this.contentService.insertTbContent(tbContent);  
}
```

### 8.1.11.2 修改 service

```
Integer insertTbContent(TbContent tbContent);
```

### 8.1.11.3 修改 serviceImpl

```
/**  
 * 根据分类添加内容  
 * @param tbContent  
 * @return
```

```
*/  
  
@Override  
  
@LcnTransaction  
  
public Integer insertTbContent(TbContent tbContent) {  
  
    tbContent.setUpdated(new Date());  
  
    tbContent.setCreated(new Date());  
  
    return this.tbContentMapper.insertSelective(tbContent);  
  
}
```

## 8.1.12 在 backend\_content 中实现根据分类添加内容

### 8.1.12.1 修改 controller

```
/**  
 * 根据分类添加内容  
 */  
  
@RequestMapping("/insertTbContent")  
  
public Result insertTbContent(TbContent tbContent){  
  
    try{  
  
        return this.contentService.insertTbContent(tbContent);  
  
    }catch (Exception e){  
  
        e.printStackTrace();  
  
    }  
  
}
```

```
}

    return Result.build(500,"error");
}
```

#### 8.1.12.2 修改 service

```
Result insertTbContent(TbContent tbContent);
```

#### 8.1.12.3 修改 serviceImpl

```
/**
 * 根据分类添加内容
 * @param tbContent
 * @return
 */
@Override
@LcnTransaction
public Result insertTbContent(TbContent tbContent) {

    Integer num = this.commonContentFeignClient.insertTbContent(tbContent);

    if(num != null){

        return Result.ok();
    }
}
```

```
}  
  
    return Result.error("添加失败");  
  
}
```

#### 8.1.12.4 修改 feignClient

```
@PostMapping("/service/content/insertTbContent")  
  
Integer insertTbContent(@RequestBody TbContent tbContent);
```

### 8.1.13 在 common\_content 服务中实现删除分类下的内容

#### 8.1.13.1 修改 controller

```
/**  
 * 删除分类下的内容  
 */  
  
@RequestMapping("/deleteContentByIds")  
  
public Integer deleteContentByIds(@RequestParam Long id){  
  
    return this.contentService.deleteContentByIds(id);  
  
}
```

---

### 8.1.13.2 修改 service

```
Integer deleteContentByIds(Long id);
```

### 8.1.13.3 修改 serviceImpl

```
/**
 * 删除分类内容
 * @param id
 * @return
 */
@Override
@LcnTransaction
public Integer deleteContentByIds(Long id) {
    return this.tbContentMapper.deleteByPrimaryKey(id);
}
```

---

## 8.1.14 在 backend\_content 服务中实现删除分类下的内容

### 8.1.14.1 修改 controller

```
/**
 * 删除分类的内容
 */
@RequestMapping("/deleteContentByIds")
public Result deleteContentByIds(Long ids){
    try{
        return this.contentService.deleteContentByIds(ids);
    }catch (Exception e){
        e.printStackTrace();
    }
    return Result.build(500,"error");
}
```

### 8.1.14.2 修改 service

```
Result deleteContentByIds(Long ids);
```

#### 8.1.14.3 修改 serviceImpl

```
/**
 * 删除分类下的内容
 * @param ids
 * @return
 */
@Override
@LcnTransaction
public Result deleteContentByIds(Long ids) {

    Integer num = this.commonContentFeignClient.deleteContentByIds(ids);

    if(num != null){

        return Result.ok();

    }

    return Result.error("删除失败");

}
```

#### 8.1.14.4 修改 feignClient

```
@RequestMapping("/service/content/deleteContentByIds")

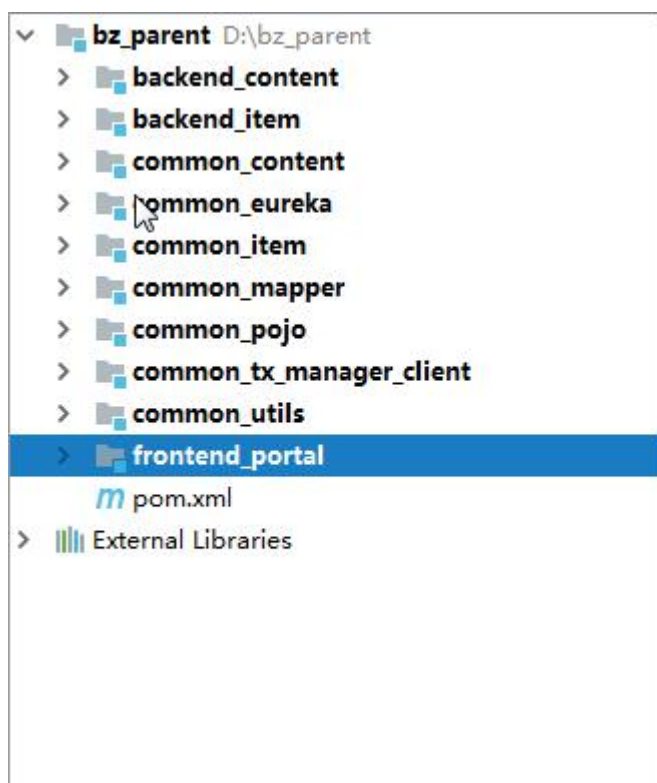
Integer deleteContentByIds(@RequestParam("id") Long id);
```



## 六、 开发百战商城前台系统

### 1 创建 frontend\_item 项目

#### 1.1 创建项目



#### 1.2 修改 POM 文件 , 添加依赖

```
<?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  
http://maven.apache.org/xsd/maven-4.0.0.xsd">
```

```
<parent>
```

```
  <artifactId>bz_parent</artifactId>
```

```
  <groupId>com.bjsxt</groupId>
```

```
  <version>1.0-SNAPSHOT</version>
```

```
</parent>
```

```
<modelVersion>4.0.0</modelVersion>
```

```
<artifactId>frontend_portal</artifactId>
```

```
<dependencies>
```

```
  <!--pojo-->
```

```
  <dependency>
```

```
    <groupId>com.bjsxt</groupId>
```

```
    <artifactId>common_pojo</artifactId>
```

```
    <version>1.0-SNAPSHOT</version>
```

```
  </dependency>
```

```
  <!--utils-->
```

```
  <dependency>
```

```
    <groupId>com.bjsxt</groupId>
```

```
    <artifactId>common_utils</artifactId>
```

```
    <version>1.0-SNAPSHOT</version>
```

---

```
</dependency>
```

```
<!--Spring Boot Web Starter-->
```

```
<dependency>
```

```
<groupId>org.springframework.boot</groupId>
```

```
<artifactId>spring-boot-starter-web</artifactId>
```

```
</dependency>
```

```
<!--Spring Cloud Eureka Client Starter-->
```

```
<dependency>
```

```
<groupId>org.springframework.cloud</groupId>
```

```
<artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
```

```
</dependency>
```

```
<!--Spring Cloud OpenFeign Starter-->
```

```
<dependency>
```

```
<groupId>org.springframework.cloud</groupId>
```

```
<artifactId>spring-cloud-starter-openfeign</artifactId>
```

```
</dependency>
```

```
</dependencies>
```

```
</project>
```

---

### 1.3 创建配置文件，添加相关配置

```
spring:

  application:

    name: frontend-portal

server:

  port: 9030

eureka:

  client:

    serviceUrl:

      defaultZone: http://eureka-server:8761/eureka/
```

### 1.4 创建启动类，添加相关注解

```
/**
 * Frontend_portal 服务启动类
 */

@SpringBootApplication

@EnableDiscoveryClient

@EnableFeignClients
```

```
public class FrontendPortalApplication {  
  
    public static void main(String[] args){  
  
        SpringApplication.run(FrontendPortalApplication.class,args);  
  
    }  
  
}
```

## 2 查询首页内容

### 2.1 实现首页商品分类查询接口

#### 2.1.1 在 common\_item 服务中实现商品分类查询

##### 2.1.1.1 创建首页显示商品分类模型

CatNode

```
public class CatNode implements Serializable {  
  
    @JsonProperty("n")  
    private String name;  
  
    @JsonProperty("i")  
    private List<?> item;  
  
    public void setName(String name) {  
  
        this.name = name;  
  
    }  
  
}
```

```
}

public String getName() {

    return name;

}

public List<?> getItem() {

    return item;

}

public void setItem(List<?> item) {

    this.item = item;

}

}
```

CatResult

```
public class CatResult implements Serializable {

    private List<?> data;

    public List<?> getData() {
```

```
        return data;

    }

    public void setData(List<?> data) {

        this.data = data;

    }

}
```

#### 2.1.1.2修改 controller

```
/**
 * 查询首页商品分类
 */
@RequestMapping("/selectItemCategoryAll")
public CatResult selectItemCategoryAll(){

    return this.itemCategoryService.selectItemCategoryAll();

}
```

#### 2.1.1.3修改 service

```
CatResult selectItemCategoryAll();
```

#### 2.1.1.4修改 serviceImpl

```
/**
 * 查询首页商品分类
 * @return
 */
@Override
public CatResult selectItemCategoryAll() {
    CatResult catResult = new CatResult();

    //查询商品分类
    catResult.setData(getCatList(0L));

    System.out.println(catResult.getData().size());

    return catResult;
}

/**
 * 私有方法，查询商品分类
 */
private List<?> getCatList(Long parentId){

    //创建查询条件

    TbItemCatExample example = new TbItemCatExample();
```



```
TbItemCatExample.Criteria criteria = example.createCriteria();

criteria.andParentIdEqualTo(parentId);

List<TbItemCat> list = this.tbItemCatMapper.selectByExample(example);

List resultList = new ArrayList();

int count = 0;

for(TbItemCat tbItemCat:list){

    //判断是否是父节点

    if(tbItemCat.getIsParent()){

        CatNode catNode = new CatNode();

        catNode.setName(tbItemCat.getName());

        catNode.setItem(getCatList(tbItemCat.getId()));

        resultList.add(catNode);

        count++;

        //只取商品分类中的 18 条数据

        if (count == 18){

            break;

        }

    }else{

        resultList.add(tbItemCat.getName());

    }

}

return resultList;
```

```
}
```

## 2.1.2 在 frontend\_portal 服务中实现商品分类查询

### 2.1.2.1 创建 controller

```
/**  
 * 首页商品分类  
 */  
  
@RestController  
@RequestMapping("/frontend/itemCategory")  
  
public class ItemCategoryController {  
  
    @Autowired  
    private ItemCategoryService itemCategoryService;  
  
    /**  
     * 查询首页商品分类  
     */  
    @RequestMapping("/selectItemCategoryAll")
```

```
public Result selectItemCategoryAll(){

    try{

        return this.itemCategoryService.selectItemCategoryAll();

    }catch(Exception e){

        e.printStackTrace();

    }

    return Result.build(500,"error");

}

}
```

### 2.1.2.2创建 service

```
public interface ItemCategoryService {

    Result selectItemCategoryAll();

}
```

### 2.1.2.3创建 serviceImpl

```
/**

 * 查询首页商品分类

 * @return
```

```
*/

@Override

public Result selectItemCategoryAll() {

    CatResult catResult =

this.commonItemFeignClient.selectItemCategoryAll();

    if(catResult != null ){

        return Result.ok(catResult);

    }

    return Result.error("查无结果");

}

}
```

#### 2.1.2.4创建 feignClient

```
@FeignClient(value="common-item")

public interface CommonItemFeignClient {

    //-----/service/ItemCategory

    @PostMapping("/service/itemCategory/selectItemCategoryAll")

    CatResult selectItemCategoryAll();

}
```

---

## 2.2 实现首页大广告位查询接口

### 2.2.1 在 common\_content 服务中实现首页大广告位查询接口

#### 2.2.1.1修改 controller

```
/**
 * 查询首页大广告位
 */
@RequestMapping("/selectFrontendContentByAD")
public List<Map> selectFrontendContentByAD(){
    return this.contentService.selectFrontendContentByAD();
}
```

#### 2.2.1.2修改 serviceImpl

```
/**
 * 查询首页大广告位
 * @return
 */
@Override
public List<Map> selectFrontendContentByAD() {
```

```
TbContentExample example = new TbContentExample();

TbContentExample.Criteria criteria = example.createCriteria();

criteria.andCategoryIdEqualTo(this.AD);

List<TbContent> list = this.tbContentMapper.selectByExample(example);

List<Map> result = new ArrayList<>();

//模型转换

for(TbContent content:list){

    Map map = new HashMap();

    map.put("heightB",240);

    map.put("src",content.getPic());

    map.put("width",670);

    map.put("alt",content.getSubTitle());

    map.put("srcB",null);

    map.put("widthB",550);

    map.put("href",content.getUrl());

    map.put("height",240);

    result.add(map);

}

return result;

}
```

---

### 2.2.1.3 修改 service

```
List<Map> selectFrontendContentByAD();
```

## 2.2.2 在 frontend\_portal 服务中实现查询首页大广告位

### 2.2.2.1 创建 controller

```
/**
 * 首页内容管理 Controller
 */
@RestController
@RequestMapping("/frontend/content")
public class ContentController {

    @Autowired
    private ContentService contentService;

    /**
     * 查询首页大广告位
     */
    @RequestMapping("/selectFrontendContentByAD")
```

```
public Result selectFrontendContentByAD(){

    try{

        return this.contentService.selectFrontendContentByAD();

    }catch(Exception e){

        e.printStackTrace();

    }

    return Result.build(500,"error");

}
```

#### 2.2.2.2 创建 service

```
public interface ContentService {

    Result selectFrontendContentByAD();

}
```

#### 2.2.2.3 创建 serviceImpl

```
/**

 * 首页内容管理 Service
```



```
*/  
  
@Service  
  
public class ContentServiceImpl implements ContentService {  
  
    @Autowired  
  
    private CommonContentFeignClient commonContentFeignClient;  
  
    /**  
     * 查询首页大广告位  
     * @return  
     */  
  
    @Override  
  
    public Result selectFrontendContentByAD() {  
  
        List<Map> list =  
  
this.commonContentFeignClient.selectFrontendContentByAD();  
  
        if(list != null && list.size() > 0){  
  
            return Result.ok(list);  
  
        }  
  
        return Result.error("查无结果");  
  
    }  
  
}
```

#### 2.2.2.4 创建 feignClient

```
@FeignClient(value = "common-content")

public interface CommonContentFeignClient {

    @PostMapping("/service/content/selectFrontendContentByAD")

    List<Map> selectFrontendContentByAD();

}
```

#### 2.2.3 测试首页大广告位

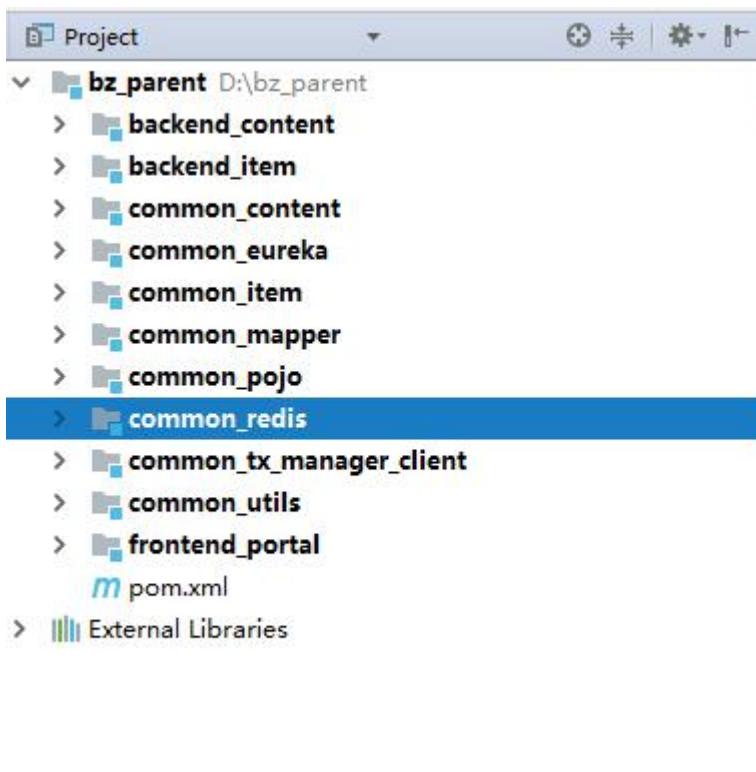


## 3 缓存系统

### 3.1 创建缓存服务

#### 3.1.1 创建 common\_redis 服务

##### 3.1.1.1 创建项目



##### 3.1.1.2 修改 POM 文件，添加坐标依赖

```
<?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

http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <parent>
```

```
<artifactId>bz_parent</artifactId>
```

```
<groupId>com.bjsxt</groupId>
```

```
<version>1.0-SNAPSHOT</version>
```

```
</parent>
```

```
<modelVersion>4.0.0</modelVersion>
```

```
<artifactId>common_redis</artifactId>
```

```
<dependencies>
```

```
<!--pojo-->
```

```
<dependency>
```

```
<groupId>com.bjsxt</groupId>
```

```
<artifactId>common_pojo</artifactId>
```

```
<version>1.0-SNAPSHOT</version>
```

```
</dependency>
```

```
<!--utils-->
```

```
<dependency>
```

```
<groupId>com.bjsxt</groupId>
```

```
<artifactId>common_utils</artifactId>
```

```
<version>1.0-SNAPSHOT</version>
```

```
</dependency>
```

---

*<!--Spring Boot Web Starter-->*

**<dependency>**

**<groupId>**org.springframework.boot**</groupId>**

**<artifactId>**spring-boot-starter-web**</artifactId>**

**</dependency>**

*<!--Spring Cloud Eureka Client Starter-->*

**<dependency>**

**<groupId>**org.springframework.cloud**</groupId>**

**<artifactId>**spring-cloud-starter-netflix-eureka-client**</artifactId>**

**</dependency>**

*<!--Spring Cloud OpenFeign Starter-->*

**<dependency>**

**<groupId>**org.springframework.cloud**</groupId>**

**<artifactId>**spring-cloud-starter-openfeign**</artifactId>**

**</dependency>**

*<!--Spring Boot Data Redis Starter-->*

**<dependency>**

**<groupId>**org.springframework.boot**</groupId>**

**<artifactId>**spring-boot-starter-data-redis**</artifactId>**

**</dependency>**

```
</dependencies>

<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
</project>
```

### 3.1.1.3 创建配置文件

```
spring:
  application:
    name: common-redis
  redis:
    database: 1
    host: 192.168.70.157
```

---

**port:** 6379

**jedis:**

**pool:**

**max-active:** 200 #最大连接数

**max-wait:** -1 #连接池最大等待时间(负数表示没有限制)

**max-idle:** 10 #连接池最大空闲数

**min-idle:** 0 #连接池最小空闲数

**timeout:** 2000 #连接超时时间

**server:**

**port:** 9080

**eureka:**

**client:**

**serviceUrl:**

**defaultZone:** http://eureka-server:8761/eureka/

### 3.1.1.4 创建启动类

/\*\*

---

```
* CommonRedis 启动类
```

```
*/
```

```
@SpringBootApplication
```

```
@EnableDiscoveryClient
```

```
public class CommonRedisApplication {
```

```
    public static void main(String[] args){
```

```
        SpringApplication.run(CommonRedisApplication.class,args);
```

```
    }
```

```
}
```

### 3.1.1.5 创建配置类，配置序列化器

```
/**
```

```
* 配置序列化器
```

```
*/
```

```
@Configuration
```

```
public class RedisConfig {
```

```
/**
```

```
* 配置 RedisTemplate 序列化器
```



```
*/

@Bean

public RedisTemplate<String,Object>

setRedisTemplate(RedisConnectionFactory factory){

    //创建 RedisTemplate

    RedisTemplate<String,Object> redisTemplate = new RedisTemplate<>();

    redisTemplate.setConnectionFactory(factory);

    //设置序列化器

    //创建 Redis 中的 value 的序列化器

    Jackson2JsonRedisSerializer jackson2JsonRedisSerializer = new

Jackson2JsonRedisSerializer(Object.class);

    ObjectMapper objectMapper = new ObjectMapper();

    objectMapper.setVisibility(PropertyAccessor.ALL,

JsonAutoDetect.Visibility.ANY);

    objectMapper.enableDefaultTyping(ObjectMapper.DefaultTyping.NON_FINAL);

    jackson2JsonRedisSerializer.setObjectMapper(objectMapper);

    //创建 Redis 中的 key 的序列化器

    StringRedisSerializer stringRedisSerializer = new StringRedisSerializer();
```

```
//设置 Redis 中的 String 类型的 value 的序列化器
redisTemplate.setValueSerializer(jackson2JsonRedisSerializer);

//设置 Redis 中的 Hash 类型的 value 的序列化器
redisTemplate.setHashValueSerializer(jackson2JsonRedisSerializer);

//设置 Redis 中的 String 类型的 key 的序列化器
redisTemplate.setKeySerializer(stringRedisSerializer);

//设置 Redis 中的 Hash 类型的 key 的序列化器
redisTemplate.setKeySerializer(stringRedisSerializer);

redisTemplate.afterPropertiesSet();

return redisTemplate;
}
}
```

## 3.2 将首页商品分类数据添加到缓存中

### 3.2.1 在 common\_redis 服务中添加缓存首页商品分类实现

#### 3.2.1.1 创建 controller

```
/**
```

---

*\* 缓存首页商品分类*

*\*/*

**@RestController**

**@RequestMapping("/redis/itemCategory")**

**public class** ItemCategoryController {

**@Autowired**

**private** ItemCategoryService **itemCategoryService**;

*/\*\**

*\* 向 Redis 中添加缓存数据*

*\*/*

**@RequestMapping("/insertItemCategory")**

**public void** insertItemCategory(**@RequestBody** CatResult catResult){

**this.itemCategoryService.insertItemCategory**(catResult);

}

*/\*\**

*\* 查询 Redis 中缓存的首页商品分类*

*\*/*

**@RequestMapping("/selectItemCategory")**

```
public CatResult selectItemCategory(){  
  
    return this.itemCategoryService.selectItemCategory();  
  
}  
}
```

### 3.2.1.2 创建 service

```
public interface ItemCategoryService {  
  
    void insertItemCategory(CatResult catResult);  
  
    CatResult selectItemCategory();  
  
}
```

### 3.2.1.3 创建 serviceImpl

```
/**  
  
 * 缓存首页商品分类业务层  
  
 */  
  
@Service  
  
public class ItemCategoryServiceImpl implements ItemCategoryService {
```

@Autowired

**private** RedisTemplate<String,Object> **redisTemplate**;

@Value("\${frontend\_catresult\_redis\_key}")

**private** String **frontend\_catresult\_redis\_key**;

/\*\*

\* 向缓存中添加首页商品分类

\* @param *catResult*

\*/

@Override

**public void** insertItemCategory(CatResult catResult) {

**this.redisTemplate.opsForValue().set(this.frontend\_catresult\_redis\_key,catResult)**

;

}

/\*\*

\* 查询缓存中的首页商品分类

\* @return

\*/

@Override

```
public CatResult selectItemCategory() {  
  
    return  
  
(CatResult)this.redisTemplate.opsForValue().get(this.frontend_catresult_redis_key);  
  
}  
  
}
```

#### 3.2.1.4 修改配置文件添加缓存首页商品分类的 key

```
#配置缓存首页商品分类的 key  
  
frontend_catresult_redis_key: frontend:catresult:redis:key
```

### 3.2.2 在 frontend\_portal 服务中添加查询缓存业务

#### 3.2.2.1 修改 serviceImpl

```
/**  
  
 * 查询首页商品分类  
  
 * @return  
  
 */  
  
@Override  
  
public Result selectItemCategoryAll() {
```

```
//查询缓存

try{

    CatResult catResult =

this.commonRedisFeignClient.selectItemCategory();

    //判断缓存中是否命中

    if(catResult != null && catResult.getData() != null &&

catResult.getData().size() > 0){

        return Result.ok(catResult);

    }

}catch(Exception e){

    e.printStackTrace();

}

//查询数据库

CatResult catResult = this.commonItemFeignClient.selectItemCategoryAll();

//添加到缓存

try{

    if(catResult != null && catResult.getData() != null &&

catResult.getData().size() > 0){

        this.commonRedisFeignClient.insertItemCategory(catResult);

    }

}catch(Exception e){

    e.printStackTrace();

}
```

```
}

    if(catResult != null && catResult.getData() != null &&
catResult.getData().size() > 0){

        return Result.ok(catResult);

    }

    return Result.error("查无结果");

}
```

### 3.2.2.2创建 feignClient

```
/**
 * 请求 common-redis 服务
 */

@FeignClient(value = "common-redis")

public interface CommonRedisFeignClient {

    //-----/redis/itemCategory

    @PostMapping("/redis/itemCategory/insertItemCategory")

    void insertItemCategory(@RequestBody CatResult catResult);

}
```



```
@PostMapping("/redis/itemCategory/selectItemCategory")

CatResult selectItemCategory();

}
```

### 3.3 将首页大广告位数据添加到缓存中

#### 3.3.1 在 common\_redis 服务中添加缓存首页大广告位数据

##### 3.3.1.1 创建 controller

```
/**
 * 缓存首页大广告位
 */

@RestController

@RequestMapping("/redis/content")

public class ContentController {

    @Autowired

    private ContentService contentService;

    /**
     * 将大广告位的数据添加到缓存中
     */
}
```

```

    */

    @RequestMapping("/insertContentAD")

    public void insertContentAD(@RequestBody List<Map> list){

        this.contentService.insertContentAD(list);

    }

    /**

     * 查询缓存中首页大广告位的数据

     */

    @RequestMapping("/selectContentAD")

    public List<Map> selectContentAD(){

        return this.contentService.selectContentAD();

    }

}

```

### 3.3.1.2 创建 service

```

public interface ContentService {

    void insertContentAD(List<Map> list);

    List<Map> selectContentAD();

}

```

### 3.3.1.3 创建 serviceImpl

```
/**
 * 缓存首页大广告位业务层
 */

@Service

public class ContentServiceImpl implements ContentService {

    @Autowired

    private RedisTemplate<String,Object> redisTemplate;

    @Value("${frontend_ad_redis_key}")

    private String frontend_ad_redis_key;

    /**
     * 向缓存中添加首页大广告位
     * @param list
     */

    @Override

    public void insertContentAD(List<Map> list) {
```

```
        this.redisTemplate.opsForValue().set(this.frontend_ad_redis_key,list);

    }

    /**
     * 查询缓存中的首页大广告位
     * @return
     */
    @Override
    public List<Map> selectContentAD() {

        return (List<Map>)
this.redisTemplate.opsForValue().get(this.frontend_ad_redis_key);

    }

}
```

#### 3.3.1.4修改配置文件添加缓存大广告位的 key

```
#配置缓存首页大广告位的 key

frontend_ad_redis_key: frontend:ad:redis:key:89
```

### 3.3.2 在 frontend\_portal 服务中添加首页大广告位查询缓存业务

#### 3.3.2.1 修改 serviceImpl

```
/**
 * 查询首页大广告位
 * @return
 */
@Override
public Result selectFrontendContentByAD() {

    //查询缓存

    try{

        List<Map> list = this.commonRedisFeignClient.selectContentAD();

        if(list != null && list.size() > 0){

            return Result.ok(list);

        }

    }catch(Exception e){

        e.printStackTrace();

    }

    //查询数据库

    List<Map> list =

    this.commonContentFeignClient.selectFrontendContentByAD();

    if(list != null && list.size() > 0){
```

---

```
//将查询到数据添加到缓存中
```

```
this.commonRedisFeignClient.insertContentAD(list);

}

if(list != null && list.size() > 0){

    return Result.ok(list);

}

return Result.error("查无结果");

}
```

### 3.3.2.2 修改 feignClient

```
@PostMapping("/redis/content/insertContentAD")

void insertContentAD(@RequestBody List<Map> list);

@PostMapping("/redis/content/selectContentAD")

List<Map> selectContentAD();
```

---

## 4 搜索服务

### 4.1 修改 solr 的 schema.xml 文件

#### 4.1.1 配置业务字段

```
<field name="item_title" type="text_ik" indexed="true" stored="true"/>

<field name="item_sell_point" type="text_ik" indexed="true" stored="true"/>

<field name="item_price" type="long" indexed="true" stored="true"/>

<field name="item_image" type="string" indexed="false" stored="true" />

<field name="item_category_name" type="string" indexed="true"
stored="true" />

<field name="item_desc" type="text_ik" indexed="true" stored="false" />

<field name="item_keywords" type="text_ik" indexed="true" stored="false"
multiValued="true"/>

<copyField source="item_title" dest="item_keywords"/>

<copyField source="item_sell_point" dest="item_keywords"/>

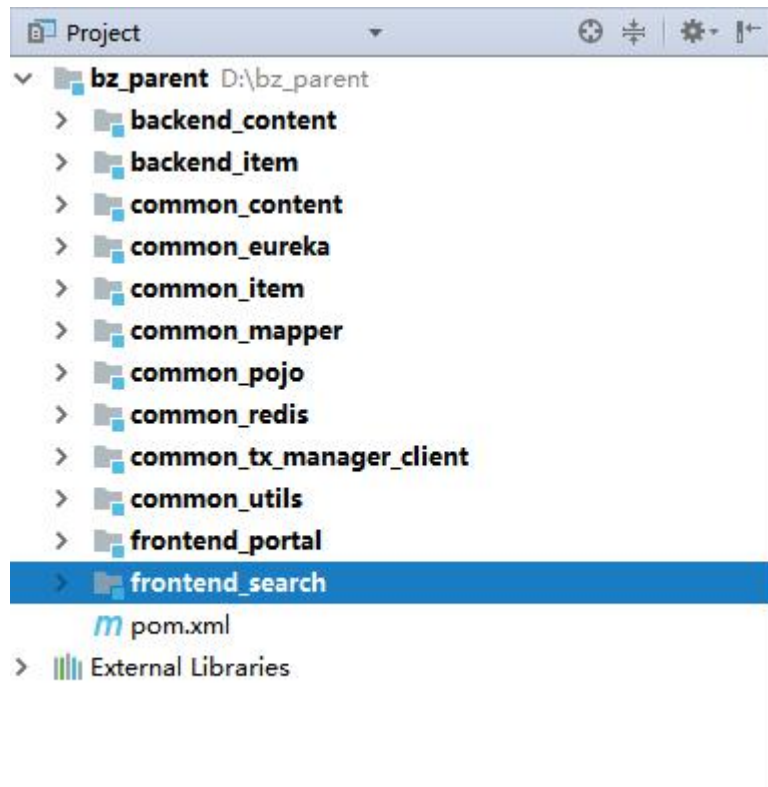
<copyField source="item_category_name" dest="item_keywords"/>

<copyField source="item_desc" dest="item_keywords"/>
```

## 4.2 创建搜索服务

### 4.2.1 创建 frontend\_search 服务

#### 4.2.1.1 创建项目



#### 4.2.1.2 修改 POM 文件，添加坐标依赖

```
<?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
http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <parent>

        <artifactId>bz_parent</artifactId>
```



```
<groupId>com.bjsxt</groupId>
```

```
<version>1.0-SNAPSHOT</version>
```

```
</parent>
```

```
<modelVersion>4.0.0</modelVersion>
```

```
<artifactId>frontend_search</artifactId>
```

```
<dependencies>
```

```
<!--pojo-->
```

```
<dependency>
```

```
<groupId>com.bjsxt</groupId>
```

```
<artifactId>common_mapper</artifactId>
```

```
<version>1.0-SNAPSHOT</version>
```

```
</dependency>
```

```
<!--utils-->
```

```
<dependency>
```

```
<groupId>com.bjsxt</groupId>
```

```
<artifactId>common_utils</artifactId>
```

```
<version>1.0-SNAPSHOT</version>
```

```
</dependency>
```

---

```
<!--Spring Boot Web Starter-->
```

```
<dependency>
```

```
    <groupId>org.springframework.boot</groupId>
```

```
    <artifactId>spring-boot-starter-web</artifactId>
```

```
</dependency>
```

```
<!--Spring Cloud Eureka Client Starter-->
```

```
<dependency>
```

```
    <groupId>org.springframework.cloud</groupId>
```

```
    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
```

```
</dependency>
```

```
<!--Spring Boot Data Solr Starter-->
```

```
<dependency>
```

```
    <groupId>org.springframework.boot</groupId>
```

```
    <artifactId>spring-boot-starter-data-solr</artifactId>
```

```
</dependency>
```

```
</dependencies>
```

```
<build>
```

```
    <plugins>
```

```
<plugin>

  <groupId>org.springframework.boot</groupId>

  <artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>
```

#### 4.2.1.3 创建配置文件

**spring:**

**application:**

**name:** frontend-search

**datasource:**

**driverClassName:** com.mysql.jdbc.Driver

**url:** jdbc:mysql://localhost:3306/bz\_shop?characterEncoding=UTF-8

**username:** root

**password:** root

**type:** com.alibaba.druid.pool.DruidDataSource

**data:**

**solr:**

---

**host:** http://192.168.70.158:8080/solr

**core:** collection1

**server:**

**port:** 9100

**eureka:**

**client:**

**serviceUrl:**

**defaultZone:** http://eureka-server:8761/eureka/

#### 4.2.1.4 创建启动类

```
/**  
 * 搜索服务启动类  
 */  
  
@SpringBootApplication  
@MapperScan("com.bjsxt.mapper")  
  
public class FrontendSearchApplication {  
  
    public static void main(String[] args){
```

```
        SpringApplication.run(FrontendSearchApplication.class,args);

    }

}
```

#### 4.2.1.5 创建配置类，配置 SolrTemplate

```
/**
 * SolrTemplate 配置类
 */
@Configuration
public class SolrConfig {

    @Autowired

    private SolrClient solrClient;

    @Bean

    public SolrTemplate getSolrTemplate(){

        return new SolrTemplate(solrClient);

    }

}
```

---

## 4.3 从关系型数据库中查询业务数据

### 4.3.1 编写查询数据的 SQL 语句

```
SELECT

    item.id,

    item.title,

    item.sell_point,

    item.price,

    item.image,

    cat.`name`,

    idesc.item_desc

FROM

    tb_item item

LEFT JOIN tb_item_cat cat ON item.cid = cat.id

LEFT JOIN tb_item_desc idesc ON item.id = idesc.item_id
```

### 4.3.2 创建 SolrItem 实体

```
package com.bjsxt.pojo;

import java.io.Serializable;

/**
```

---

*\* Solr 实体*

*\*/*

**public class** SolrItem **implements** Serializable {

**private** Long **id**;

**private** String **title**;

**private** String **sell\_point**;

**private** Long **price**;

**private** String **image**;

**private** String **name**;

**private** String **item\_desc**;

**public void** setId(Long id) {

**this.id** = id;

    }

**public void** setTitle(String title) {

**this.title** = title;

    }

**public void** setSell\_point(String sell\_point) {

**this.sell\_point** = sell\_point;

    }

---

```
public void setPrice(Long price) {
```

```
    this.price = price;
```

```
}
```

```
public void setImage(String image) {
```

```
    this.image = image;
```

```
}
```

```
public void setName(String name) {
```

```
    this.name = name;
```

```
}
```

```
public void setItem_desc(String item_desc) {
```

```
    this.item_desc = item_desc;
```

```
}
```

```
public Long getId() {
```

```
    return id;
```

```
}
```

```
public String getTitle() {
```



---

```
        return title;
    }

    public String getSell_point() {

        return sell_point;
    }

    public Long getPrice() {

        return price;
    }

    public String getImage() {

        return image;
    }

    public String getName() {

        return name;
    }

    public String getItem_desc() {

        return item_desc;
    }
```

```
}
```

### 4.3.3 添加 SolrItemMapper.xml 文件

```
<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd" >

<mapper namespace="com.bjsxt.mapper.SolrItemMapper" >

    <select id="getItemList" resultType="com.bjsxt.pojo.SolrItem">

        SELECT

        item.id,

        item.title,

        item.sell_point,

        item.price,

        item.image,

        cat.`name`,

        idesc.item_desc

        FROM

        tb_item item

        LEFT JOIN tb_item_cat cat ON item.cid = cat.id

        LEFT JOIN tb_item_desc idesc ON item.id = idesc.item_id
```

```
</select>
```

```
</mapper>
```

### 4.3.4 添加 SolrItemMapper 接口

```
public interface SolrItemMapper {  
  
    List<SolrItem> getItemList();  
  
}
```

## 4.4 在 frontend\_search 服务中添加数据导入接口

### 4.4.1 创建 controller

```
/**  
  
 * 搜索服务 Controller  
  
 */  
  
@RestController  
@RequestMapping("/search")  
  
public class SolrController {  
  
  
  
  
  
  
  
  
    @Autowired
```

```
private SolrService solrService;

/**
 * 向 Solr 索引库中导入数据
 */

@RequestMapping("/importAll")

public Result importAll(){

    return this.solrService.importAll();

}

}
```

#### 4.4.2 创建 service

```
public interface SolrService{

    Result importAll();

}
```

#### 4.4.3 创建 serviceImpl

```
/**
 * 搜索服务业务层
 */
```

---

@Service

**public class** SolrServiceImpl **implements** SolrService {

@Autowired

**private** SolrItemMapper **solrItemMapper**;

@Autowired

**private** SolrTemplate **solrTemplate**;

@Value("\${spring.data.solr.core}")

**private** String **collection**;

/\*\*

\* 向索引库中导入数据

\* @return

\*/

@Override

**public** Result importAll() {

**try**{

//查询数据

List<SolrItem> list = **this.solrItemMapper**.getItemList();

---

*//将数据添加索引库中*

**for**(SolrItem item:list){

*//创建 SolrInputDocument 对象*

SolrInputDocument document = **new** SolrInputDocument();

document.setField("**id**",item.getId());

document.setField("**item\_title**",item.getTitle());

document.setField("**item\_sell\_point**",item.getSell\_point());

document.setField("**item\_price**",item.getPrice());

document.setField("**item\_image**",item.getImage());

document.setField("**item\_category\_name**",item.getName());

document.setField("**item\_desc**",item.getItem\_desc());

*//写入索引库*

**this.solrTemplate**.saveDocument(**this.collection**,document);

}

**this.solrTemplate**.commit(**this.collection**);

**return** Result.ok();

}**catch**(Exception e){

e.printStackTrace();

}

**return** Result.error("导入失败");

}

```
}
```

## 4.5 在 frontend\_search 服务中添加搜索接口

### 4.5.1 在工具项目中添加为页面返回数据的实体

```
/**  
 * Solr 返回给前端页面的数据模型  
 */  
  
public class SolrDocument implements Serializable {  
  
    private String id;  
  
    private String item_title;  
  
    private String item_sell_point;  
  
    private String item_price;  
  
    private String item_image;  
  
    private String item_category_name;  
  
    private String item_desc;  
  
    public void setId(String id) {  
  
        this.id = id;  
  
    }  
}
```

---

```
public void setItem_title(String item_title) {
```

```
    this.item_title = item_title;
```

```
}
```

```
public void setItem_sell_point(String item_sell_point) {
```

```
    this.item_sell_point = item_sell_point;
```

```
}
```

```
public void setItem_price(String item_price) {
```

```
    this.item_price = item_price;
```

```
}
```

```
public void setItem_image(String item_image) {
```

```
    this.item_image = item_image;
```

```
}
```

```
public void setItem_category_name(String item_category_name) {
```

```
    this.item_category_name = item_category_name;
```

```
}
```

```
public void setItem_desc(String item_desc) {
```

```
    this.item_desc = item_desc;
```



---

```
}
```

```
public String getId() {
```

```
    return id;
```

```
}
```

```
public String getItem_title() {
```

```
    return item_title;
```

```
}
```

```
public String getItem_sell_point() {
```

```
    return item_sell_point;
```

```
}
```

```
public String getItem_price() {
```

```
    return item_price;
```

```
}
```

```
public String getItem_image() {
```

```
    return item_image;
```

```
}
```

```
public String getItem_category_name() {  
  
    return item_category_name;  
  
}  
  
public String getItem_desc() {  
  
    return item_desc;  
  
}  
}
```

#### 4.5.2 修改 controller

```
/**  
 * 搜索数据  
 */  
  
@RequestMapping("/list")  
public List<SolrDocument> selectByq(String q, @RequestParam(defaultValue =  
"1") Long page, @RequestParam(defaultValue = "10") Integer pageSize){  
  
    try{  
  
        return this.solrService.selectByq(q,page,pageSize);  
  
    }catch(Exception e){  
  
        e.printStackTrace();  
  
    }  
}
```

```
}  
  
    return null;  
  
}
```

### 4.5.3 修改 service

```
List<SolrDocument> selectByq(String q,Long page,Integer pageSize);
```

### 4.5.4 修改 serviceImpl

```
/**  
 * 搜索 Solr 索引库  
 * @param q  
 * @param page  
 * @param pageSize  
 * @return  
 */  
  
@Override  
public List<SolrDocument> selectByq(String q, Long page, Integer pageSize) {  
  
    //设置高亮查询条件
```

```
HighlightQuery query = new SimpleHighlightQuery();

Criteria criteria = new Criteria("item_keywords");

criteria.is(q);

query.addCriteria(criteria);


//设置高亮属性

HighlightOptions highlightOptions = new HighlightOptions();

highlightOptions.addField("item_title");//设置高亮显示的域

highlightOptions.setSimplePrefix("<em style='color:red'>");//设置高亮的样式的前缀

highlightOptions.setSimplePostfix("</em>");

query.setHighlightOptions(highlightOptions);


//分页

query.setOffset((page-1)*pageSize);

query.setRows(pageSize);


HighlightPage<SolrDocument> highlightPage =

this.solrTemplate.queryForHighlightPage(this.collection,query,SolrDocument.class);


List<HighlightEntry<SolrDocument>> highlighted =
```

```
highlightPage.getHighlighted();

    for(HighlightEntry<SolrDocument> tbItemHighlightEntry:highlighted){

        SolrDocument entity = tbItemHighlightEntry.getEntity();//实体对象，原始的
        实体对象

        List<HighlightEntry.Highlight> highlights =
        tbItemHighlightEntry.getHighlights();

        //如果有高亮，就取高亮

        if(highlights != null && highlights.size() > 0 &&
        highlights.get(0).getSnippets().size() > 0){

            entity.setItem_title(highlights.get(0).getSnippets().get(0));

        }

    }

    List<SolrDocument> list = highlightPage.getContent();

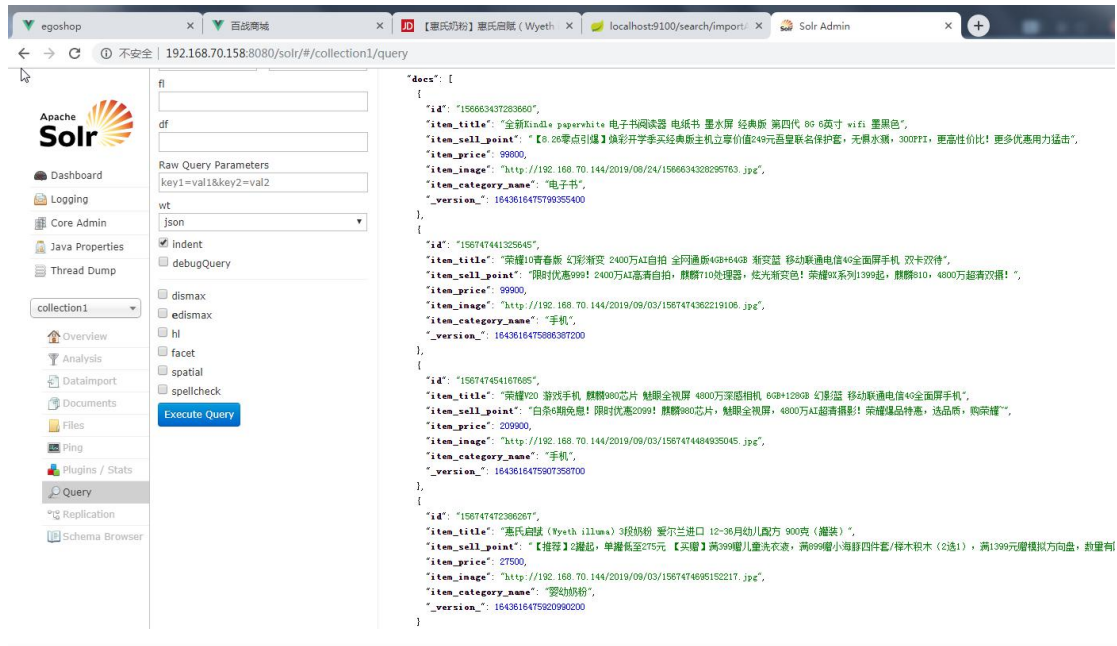
    return list;

}
```

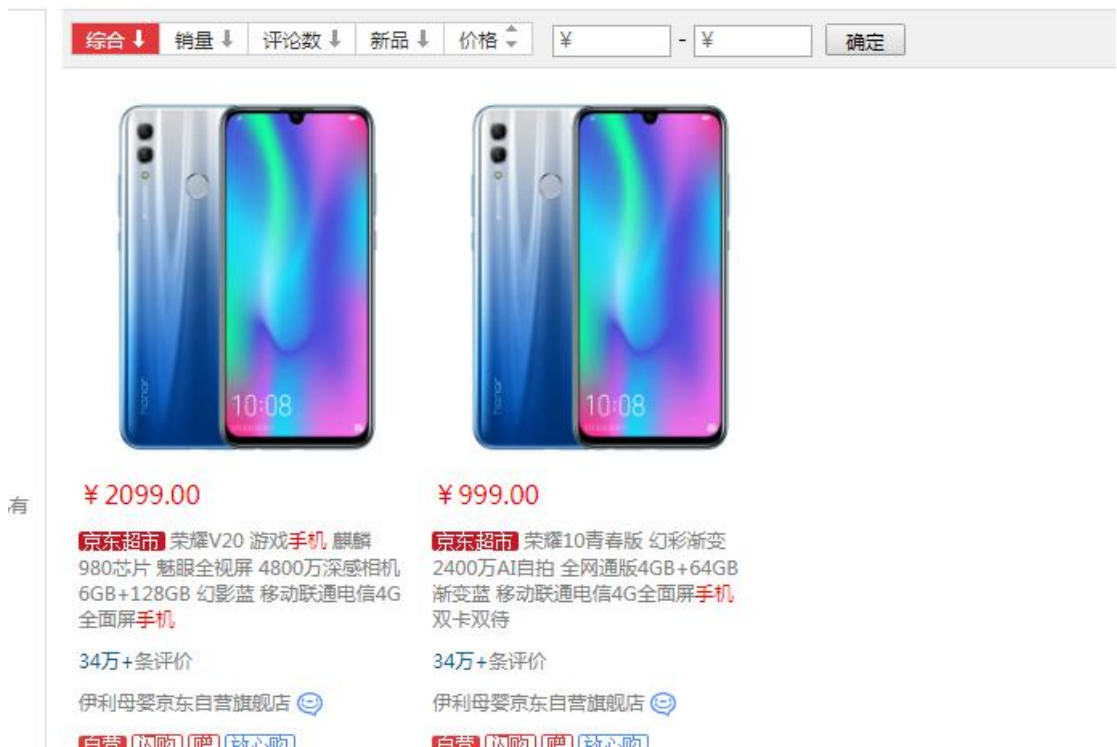
## 4.6 添加搜索测试数据

### 4.6.1 添加数据

### 4.6.2 将数据导入到 solr 中



The screenshot shows the Solr Admin interface. On the left, the 'collection1' dropdown is selected. The 'Raw Query Parameters' section shows 'key1=val1&key2=val2'. The 'Execute Query' button is visible. The main area displays the JSON response of the query, which is a list of documents. Each document contains fields such as 'id', 'item\_title', 'item\_sell\_point', 'item\_price', 'item\_image', 'item\_category\_name', and '\_version\_'.



The screenshot shows a mobile app interface with two product listings. The top bar has a search bar and a '确定' (Confirm) button. The first listing is for the 'Honor V20 游戏手机' (Honor V20 Gaming Phone) with a price of ¥2099.00. The second listing is for the 'Honor 10 青春版' (Honor 10 Youth Edition) with a price of ¥999.00. Both listings include a brief description of the phone's features and a '34万+条评价' (340,000+ reviews) badge.

---

## 5 查看商品详情

### 5.1 查看商品基本信息

#### 5.1.1 在 common\_item 服务中实现查询商品基本信息接口

##### 5.1.1.1 修改 controller

```
/**  
 * 根据商品 ID 查询商品  
 */  
  
@RequestMapping("/selectItemInfo")  
public TbItem selectItemInfo(@RequestParam Long itemId){  
    return this.itemService.selectItemInfo(itemId);  
}
```

##### 5.1.1.2 修改 service

```
TbItem selectItemInfo(Long itemId);
```

##### 5.1.1.3 修改 serviceImpl

```
/**  
 * 根据商品 ID 查询商品
```

```

    * @param itemId

    * @return

    */

@Override

public TbItem selectItemInfo(Long itemId) {

    return this.tbItemMapper.selectByPrimaryKey(itemId);

}

```

## 5.1.2 在 frontend\_portal 服务中实现查询商品基本信息

### 5.1.2.1 创建 controller

```

/**

 * 查询商品 Controller

 */

@RestController

@RequestMapping("/frontend/item")

public class ItemController {

    @Autowired

    private ItemService itemService;

}

```



```
* 查询商品基本信息

*/

@RequestMapping("/selectItemInfo")

public Result selectItemInfo(Long itemId){

    try{

        return this.itemService.selectItemInfo(itemId);

    }catch(Exception e){

        e.printStackTrace();

    }

    return Result.build(500,"error");

}

}
```

### 5.1.2.2创建 service

```
public interface ItemService {

    Result selectItemInfo(Long itemId);

}
```

### 5.1.2.3 创建 serviceImpl

```
/**
 * 查询商品业务层
 */

@Service

public class ItemServiceImpl implements ItemService {

    @Autowired

    private CommonItemFeignClient commonItemFeignClient;

    @Override

    public Result selectItemInfo(Long itemId) {

        TbItem tbItem = this.commonItemFeignClient.selectItemInfo(itemId);

        if(tbItem != null){

            return Result.ok(tbItem);

        }

        return Result.error("查无结果");

    }

}
```

#### 5.1.2.4修改 feignClient

```
//-----/service/item  
  
@PostMapping("/service/item/selectItemInfo")  
  
TbItem selectItemInfo(@RequestParam("itemId") Long itemId);
```

### 5.1.3 在 common\_item 服务中实现查询商品介绍接口

#### 5.1.3.1修改 controller

```
/**  
 * 根据商品 ID 查询商品描述  
 */  
  
@RequestMapping("/selectItemDescByItemId")  
  
public TbItemDesc selectItemDescByItemId(@RequestParam Long itemId){  
    return this.itemDescService.selectItemDescByItemId(itemId);  
}
```

#### 5.1.3.2修改 service

```
TbItemDesc selectItemDescByItemId(Long itemId);
```

### 5.1.3.3 修改 serviceImpl

```
/**
 * 根据商品 ID 查询商品介绍
 * @param itemId
 * @return
 */
@Override
public TbItemDesc selectItemDescById(Long itemId) {
    TbItemDescExample example = new TbItemDescExample();
    TbItemDescExample.Criteria criteria = example.createCriteria();
    criteria.andItemIdEqualTo(itemId);
    List<TbItemDesc> list =
    this.tbItemDescMapper.selectByExampleWithBLOBs(example);
    return list.get(0);
}
```

### 5.1.4 在 frontend\_portal 服务中实现查询商品介绍接口

#### 5.1.4.1 修改 controller

```
/**
 * 查询商品介绍
```

```
*/  
  
@RequestMapping("/selectItemDescById")  
  
public Result selectItemDescById(Long itemId){  
  
    try{  
  
        return this.itemService.selectItemDescById(itemId);  
  
    }catch(Exception e){  
  
        e.printStackTrace();  
  
    }  
  
    return Result.build(500,"error");  
  
}
```

#### 5.1.4.2 修改 service

```
Result selectItemDescById(Long itemId);
```

#### 5.1.4.3 修改 serviceImpl

```
/**  
  
 * 查询商品介绍  
  
 * @param itemId  
  
 * @return
```

```

*/

@Override

public Result selectItemDescByItemId(Long itemId) {

    TbItemDesc tbItemDesc =

    this.commonItemFeignClient.selectItemDescByItemId(itemId);

    if(tbItemDesc != null){

        return Result.ok(tbItemDesc);

    }

    return Result.error("查无结果");

}

```

#### 5.1.4.4 修改 feignClient

```

//-----/service/itemDesc

@PostMapping("/service/itemDesc/selectItemDescByItemId")

TbItemDesc selectItemDescByItemId(@RequestParam("itemId") Long itemId);

```

### 5.1.5 在 common\_item 服务中实现查询商品规格参数接口

#### 5.1.5.1 修改 controller

```

/**

```

---

```
* 根据商品 ID 查询商品规格参数
```

```
*/
```

```
@RequestMapping("/selectTbItemParamItemById")
```

```
public TbItemParamItem selectTbItemParamItemById(@RequestParam Long  
itemId){  
  
    return this.itemParamItemService.selectTbItemParamItemById(itemId);  
}
```

#### 5.1.5.2 修改 service

```
TbItemParamItem selectTbItemParamItemById(Long itemId);
```

#### 5.1.5.3 修改 serviceImpl

```
/**
```

```
* 根据商品 ID 查询商品规格参数
```

```
* @param itemId
```

```
* @return
```

```
*/
```

```
@Override
```

```
public TbItemParamItem selectTbItemParamItemById(Long itemId) {
```

```
TbItemParamItemExample example = new TbItemParamItemExample();

TbItemParamItemExample.Criteria criteria = example.createCriteria();

criteria.andItemIdEqualTo(itemId);

List<TbItemParamItem> list =

this.tbItemParamItemMapper.selectByExampleWithBLOBs(example);

return list.get(0);
}
```

## 5.1.6 在 frontend\_portal 服务中实现查询商品规格参数接



### 5.1.6.1 修改 controller

```
/**
 * 根据商品 ID 查询商品规格参数
 */
@RequestMapping("/selectTbItemParamItemById")
public Result selectTbItemParamItemById(Long itemId){

    try{

        return this.itemService.selectTbItemParamItemById(itemId);

    }catch(Exception e){

        e.printStackTrace();
    }
}
```



```
}  
  
    return Result.build(500,"error");  
  
}
```

### 5.1.6.2 修改 service

```
Result selectTbItemParamItemById(Long itemId);
```

### 5.1.6.3 修改 serviceImpl

```
/**  
 * 根据商品 ID 查询商品规格参数  
 */  
  
@Override  
  
    public Result selectTbItemParamItemById(Long itemId) {  
  
        TbItemParamItem tbItemParamItem =  
  
this.commonItemFeignClient.selectTbItemParamItemById(itemId);  
  
        if(tbItemParamItem != null){  
  
            return Result.ok(tbItemParamItem);  
  
        }  
  
        return Result.error("查无结果");  
    }
```

```
}  
  
}
```

## 5.2 添加缓存

### 5.2.1 在查询商品基本信息中添加查询缓存

#### 5.2.1.1 在 common\_redis 服务中添加缓存商品基本信息接口

##### 5.2.1.1.1 创建 controller

```
/**  
  
 * 缓存商品信息  
  
 */  
  
@RestController  
@RequestMapping("/redis/item")  
  
public class ItemController {  
  
    @Autowired  
  
    private ItemService itemService;  
  
    /**  
  
     * 缓存商品基本信息  
  
     */  
  
}
```

```
@RequestMapping("/insertItemBasicInfo")

public void insertItemBasicInfo(@RequestBody TbItem tbItem){

    this.itemService.insertItemBasicInfo(tbItem);

}

/**
 * 查询缓存中的商品基本信息
 */

@RequestMapping("/selectItemBasicInfo")

public TbItem selectItemBasicInfo(@RequestParam Long tbItemId){

    return this.itemService.selectItemBasicInfo(tbItemId);

}

}
```

#### 5.2.1.1.2 创建 service

```
public interface ItemService {

    void insertItemBasicInfo(TbItem tbItem);

    TbItem selectItemBasicInfo(Long tbItemId);

}
```

### 5.2.1.1.3 创建 serviceImpl

```
/**
 * 缓存商品业务层
 */
@Service
public class ItemServiceImpl implements ItemService {

    @Autowired
    private RedisTemplate<String,Object> redisTemplate;

    @Value("${frontend_item_basic_info_key}")
    private String frontend_item_basic_info_key;

    /**
     * 缓存商品基本信息
     * @param tbItem
     */
    @Override
    public void insertItemBasicInfo(TbItem tbItem) {

        this.redisTemplate.opsForValue().set(this.frontend_item_basic_info_key+":"+tbIt
```

```
em.getId(),tbItem);

    }

    /**
     * 查询缓存中的商品基本信息
     */
    @Override
    public TbItem selectItemBasicInfo(Long tbItemId) {

        return (TbItem)
this.redisTemplate.opsForValue().get(this.frontend_item_basic_info_key+":"+tbI
temId);

    }

}
```

#### 5.2.1.1.4 在配置文件中添加缓存商品基本信息的 key

```
#缓存商品基本信息的 key

frontend_item_basic_info_key: frontend:item:basic:info:key
```

## 5.2.1.2 在 frontend\_portal 服务中添加缓存商品基本信息业务

### 5.2.1.2.1 修改 serviceImpl

```
/**
 * 查询商品基本信息
 * @param itemId
 * @return
 */
@Override
public Result selectItemInfo(Long itemId) {

    try{

        TbItem tbItem=
this.commonRedisFeignClient.selectItemBasicInfo(itemId);

        if(tbItem != null){

            return Result.ok(tbItem);

        }

    }catch(Exception e) {

        e.printStackTrace();

    }

    TbItem tbItem = this.commonItemFeignClient.selectItemInfo(itemId);

    try{

        if(tbItem != null){

            this.commonRedisFeignClient.insertItemBasicInfo(tbItem);
```

```
    }

    }catch (Exception e){

        e.printStackTrace();

    }

    if(tbItem != null){

        return Result.ok(tbItem);

    }

    return Result.error("查无结果");

}
```

#### 5.2.1.2.2 修改 feignClient

```
//-----/redis/item

@PostMapping("/redis/item/insertItemBasicInfo")

void insertItemBasicInfo(@RequestBody TbItem tbItem);

@PostMapping("/redis/item/selectItemBasicInfo")

TbItem selectItemBasicInfo(@RequestParam("tbItemId") Long tbItemId);
```

---

### 5.2.1.3 在 common\_redis 服务中添加缓存商品介绍接口

#### 5.2.1.3.1 修改 controller

```
/**
 * 缓存商品介绍信息
 */
@RequestMapping("/insertItemDesc")
public void insertItemDesc(@RequestBody TbItemDesc tbItemDesc){
    this.itemService.insertItemDesc(tbItemDesc);
}

/**
 * 查询缓存中的商品介绍
 */
@RequestMapping("/selectItemDesc")
public TbItemDesc selectItemDesc(@RequestParam Long tbItemId){
    return this.itemService.selectItemDesc(tbItemId);
}
```

#### 5.2.1.3.2 修改 service

```
void insertItemDesc(TbItemDesc tbItemDesc);
```



```
TbItemDesc selectItemDesc(Long tbItemId);
```

### 5.2.1.3.3 修改 serviceImpl

```
/**
 * 缓存商品介绍信息
 */
@Override
public void insertItemDesc(TbItemDesc tbItemDesc) {

    this.redisTemplate.opsForValue().set(this.frontend_item_desc_key+":"+tbItemD
esc.getItemId(),tbItemDesc);
}

/**
 * 查询缓存中的商品介绍
 */
@Override
public TbItemDesc selectItemDesc(Long tbItemId) {

    return (TbItemDesc)

    this.redisTemplate.opsForValue().get(this.frontend_item_desc_key+":"+tbItemId
);
}
```

```
}
```

#### 5.2.1.3.4 在配置文件中添加缓存商品介绍的 key

*#缓存商品介绍的 key*

**frontend\_item\_desc\_key:** frontend:item:desc:key

#### 5.2.1.4 在 frontend\_portal 服务中添加查询缓存商品介绍信息业务

##### 5.2.1.4.1 修改 serviceImpl

```
/**
```

```
 * 查询商品介绍
```

```
 * @param itemId
```

```
 * @return
```

```
 */
```

```
@Override
```

```
public Result selectItemDescById(Long itemId) {
```

```
    try{
```

```
        TbItemDesc tbItemDesc =
```

```
this.commonRedisFeignClient.selectItemDesc(itemId);
```

```
    if(tbItemDesc != null){
```

```
        return Result.ok(tbItemDesc);
```

```
    }
```

```
    }catch(Exception e){
```

```
        e.printStackTrace();
```

```
    }
```

```
TbItemDesc tbItemDesc =
```

```
this.commonItemFeignClient.selectItemDescById(itemId);
```

```
    try{
```

```
        if(tbItemDesc != null){
```

```
            this.commonRedisFeignClient.insertItemDesc(tbItemDesc);
```

```
        }
```

```
    }catch(Exception e){
```

```
        e.printStackTrace();
```

```
    }
```

```
    if(tbItemDesc != null){
```

```
        return Result.ok(tbItemDesc);
```

```
    }
```

```
    return Result.error("查无结果");
```

```
}
```

#### 5.2.1.4.2 修改 feignClient

```
@PostMapping("/redis/item/insertItemDesc")

void insertItemDesc(@RequestBody TbItemDesc tbItemDesc);


@PostMapping("/redis/item/selectItemDesc")

TbItemDesc selectItemDesc(@RequestParam("tbItemId") Long tbItemId);
```

#### 5.2.1.5 在 common\_redis 服务中添加缓存商品规格参数接口

##### 5.2.1.5.1 修改 controller

```
/**
 * 缓存商品的规格参数
 */

@RequestMapping("/insertItemParamItem")

public void insertItemParamItem(@RequestBody TbItemParamItem
tbItemParamItem){

    this.itemService.insertItemParamItem(tbItemParamItem);
```

```
}

/**
 * 查询缓存中的商品规格参数
 */

@RequestMapping("/selectItemParamItem")

public TbItemParamItem selectItemParamItem(@RequestParam Long tbItemId){

    return this.itemService.selectItemParamItem(tbItemId);

}
```

#### 5.2.1.5.2 修改 service

```
void insertItemParamItem(TbItemParamItem tbItemParamItem);

TbItemParamItem selectItemParamItem(Long tbItemId);
```

#### 5.2.1.5.3 修改 serviceImpl

```
/**
 * 缓存商品的规格参数
 */

@Override
```

```
public void insertItemParamItem(TbItemParamItem tbItemParamItem) {

    this.redisTemplate.opsForValue().set(this.frontend_item_param_key+":"+tbItem
    ParamItem.getItemId(),tbItemParamItem);

}

/**
 * 查询缓存中的商品规格参数
 */

@Override

public TbItemParamItem selectItemParamItem(Long tbItemId) {

    return (TbItemParamItem)

    this.redisTemplate.opsForValue().get(this.frontend_item_param_key+":"+tbItem
    Id);

}
```

#### 5.2.1.5.4 在配置文件中添加缓存商品规格参数的 key

#缓存商品规格参数的 key

frontend\_item\_param\_key: frontend:item:param:key

## 5.2.1.6 在 frontend\_portal 服务中添加查询缓存商品规格参数业务

### 5.2.1.6.1 修改 serviceImpl

```
/**
 * 根据商品 ID 查询商品规格参数
 */
@Override
public Result selectTbItemParamItemById(Long itemId) {
    try{
        TbItemParamItem tbItemParamItem =
            this.commonRedisFeignClient.selectItemParamItem(itemId);

        if(tbItemParamItem != null){
            return Result.ok(tbItemParamItem);
        }
    }catch(Exception e){
        e.printStackTrace();
    }

    TbItemParamItem tbItemParamItem =
        this.commonItemFeignClient.selectTbItemParamItemById(itemId);

    try{
```

```
        if(tbItemParamItem != null){

this.commonRedisFeignClient.insertItemParamItem(tbItemParamItem);

        }

    }catch(Exception e){

        e.printStackTrace();

    }

    if(tbItemParamItem != null){

        return Result.ok(tbItemParamItem);

    }

    return Result.error("查无结果");

}
```

#### 5.2.1.6.2 修改 feignClient

```
@PostMapping("/redis/item/insertItemParamItem")

void insertItemParamItem(@RequestBody TbItemParamItem tbItemParamItem);

@PostMapping("/redis/item/selectItemParamItem")

TbItemParamItem selectItemParamItem(@RequestParam("tbItemId") Long
```

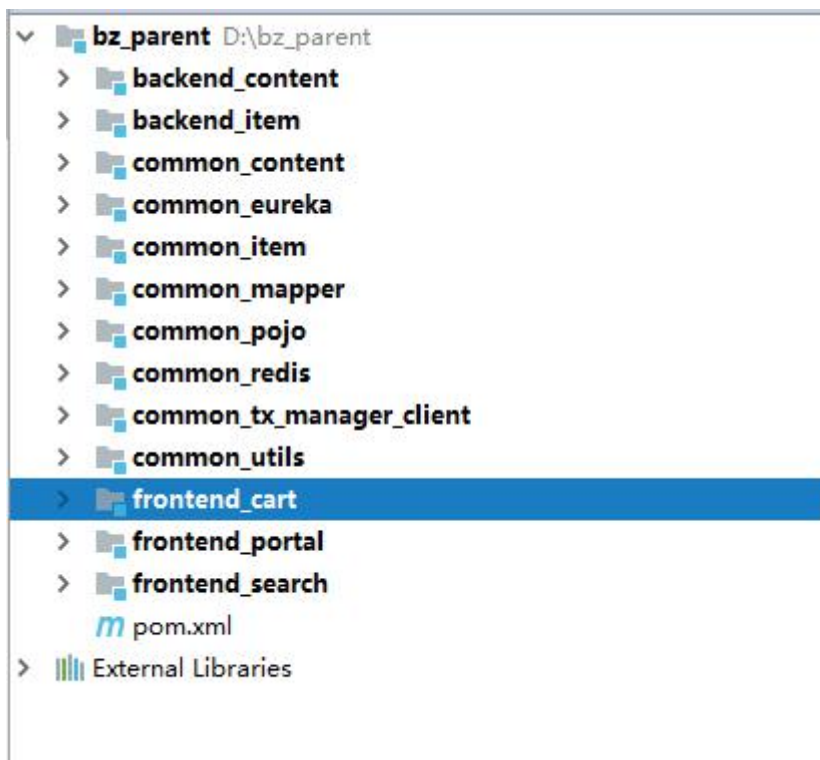


```
tbItemId);
```

## 6 购物车服务

### 6.1 创建购物车服务

#### 6.1.1 创建项目



#### 6.1.2 修改 POM 文件添加依赖坐标

```
<?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
```

---

<http://maven.apache.org/xsd/maven-4.0.0.xsd>">

<parent>

<artifactId>bz\_parent</artifactId>

<groupId>com.bjsxt</groupId>

<version>1.0-SNAPSHOT</version>

</parent>

<modelVersion>4.0.0</modelVersion>

<artifactId>frontend\_cart</artifactId>

<dependencies>

<!--pojo-->

<dependency>

<groupId>com.bjsxt</groupId>

<artifactId>common\_pojo</artifactId>

<version>1.0-SNAPSHOT</version>

</dependency>

<!--utils-->

<dependency>

<groupId>com.bjsxt</groupId>

<artifactId>common\_utils</artifactId>

<version>1.0-SNAPSHOT</version>

```
</dependency>
```

```
<!--Spring Boot Web Starter-->
```

```
<dependency>
```

```
<groupId>org.springframework.boot</groupId>
```

```
<artifactId>spring-boot-starter-web</artifactId>
```

```
</dependency>
```

```
<!--Spring Cloud Eureka Client Starter-->
```

```
<dependency>
```

```
<groupId>org.springframework.cloud</groupId>
```

```
<artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
```

```
</dependency>
```

```
<!--Spring Cloud OpenFeign Starter-->
```

```
<dependency>
```

```
<groupId>org.springframework.cloud</groupId>
```

```
<artifactId>spring-cloud-starter-openfeign</artifactId>
```

```
</dependency>
```

```
</dependencies>
```

```
<build>
```

```
<plugins>
```

```
<plugin>
```

```
<groupId>org.springframework.boot</groupId>
```

```
        <artifactId>spring-boot-maven-plugin</artifactId>

    </plugin>

</plugins>

</build>

</project>
```

### 6.1.3 创建配置文件，添加相关配置

```
spring:

  application:

    name: frontend-cart

server:

  port: 9040

eureka:

  client:

    serviceUrl:

      defaultZone: http://eureka-server:8761/eureka/
```

---

### 6.1.4 创建启动类，添加相关注解

```
/**  
 * 购物车服务  
 */  
  
@SpringBootApplication  
@EnableDiscoveryClient  
@EnableFeignClients  
  
public class FrontendCartApplication {  
    public static void main(String[] args){  
        SpringApplication.run(FrontendCartApplication.class,args);  
    }  
}
```

---

## 6.2 购物车设计

### 6.2.1 用户未登录状态下

#### 6.2.1.1 保存到客户端浏览器的 Cookie 中

### 6.2.2 用户已登录状态下

#### 6.2.2.1 保存到服务端的 Redis 中

## 6.3 在未登录的状态下操作购物车

### 6.3.1 将商品添加到购物车中

#### 6.3.1.1 创建购物车中商品模型

```
/**  
 * 购物车商品模型  
 */  
  
public class CartItem {  
    private Long id;  
    private String title;  
    private String sellPoint;  
    private String image;  
    private int num;  
    private Long price;  
  
    public void setId(Long id) {
```

```
        this.id = id;

    }

    public void setTitle(String title) {

        this.title = title;

    }

    public void setSellPoint(String sellPoint) {

        this.sellPoint = sellPoint;

    }

    public void setImage(String image) {

        this.image = image;

    }

    public void setNum(int num) {

        this.num = num;

    }

    public void setPrice(Long price) {

        this.price = price;

    }
```

---

```
public Long getId() {
```

```
    return id;
```

```
}
```

```
public String getTitle() {
```

```
    return title;
```

```
}
```

```
public String getSellPoint() {
```

```
    return sellPoint;
```

```
}
```

```
public String getImage() {
```

```
    return image;
```

```
}
```

```
public int getNum() {
```

```
    return num;
```

```
}
```

```
public Long getPrice() {
```



```
        return price;

    }

}
```

### 6.3.1.2 创建 controller

```
/**
 * 购物车 Controller
 */
@RestController
@RequestMapping("/cart")
public class CartController {

    @Autowired
    private CookieCartService cookieCartService;

    /**
     * 将商品加入购物车
     */
    @RequestMapping("/addItem")
    public Result addItem(Long itemId, String userId,
        @RequestParam(defaultValue = "1") Integer num, HttpServletRequest request,
        HttpServletResponse response){
```

```
try{

    if(StringUtils.isBlank(userId)){

        //在用户未登录的状态下

        return

this.cookieCartService.addItem(itemId,num,request,response);

    }else{

        // 在用户已登录的状态

    }

}catch(Exception e){

    e.printStackTrace();

}

return Result.build(500,"error");

}

}
```

### 6.3.1.3 创建 service

```
public interface CookieCartService {
```

```
Result addItem(Long itemId,Integer num, HttpServletRequest request,
HttpServletResponse response);
}
```

#### 6.3.1.4创建 serviceImpl

```
/**
 * 用户未登录状态下的购物车操作业务
 */
@Service
public class CookieCartServiceImpl implements CookieCartService {

    @Value("${cart_cookie_name}")
    private String cart_cookie_name;

    @Autowired
    private CommonItemFeignClient commonItemFeignClient;

    /**
     * 将商品添加到购物车中
     * @param itemId
     * @param num
     */
}
```

---

*\* @param request*

*\* @param response*

*\* @return*

*\*/*

@Override

**public** Result addItem(Long itemId,Integer num, HttpServletRequest request,  
HttpServletResponse response) {

*//1,获取临时购物车*

Map<String,CartItem> cart = **this**.getCart(request);

*//2,查询商品*

TbItem item = **this**.selectItemById(itemId);

*//3,向购物车中添加商品*

**this**.addItemToCart(cart,item,num,itemId);

*//4,将购物车通过 Cookie 写回给客户端浏览器*

**this**.addClientCookie(request,response,card);

**return** Result.ok();

}

*/\*\**

*\* 1 获取购物车*

*\*/*

**private** Map<String,CartItem> getCart(HttpServletRequest request){

```
//临时购物车已存在

String cartJson=
CookieUtils.getCookieValue(request,this.cart_cookie_name,true);

    if(StringUtils.isBlank(cartJson)){

        //临时购物车不存在

        return new HashMap<String,CartItem>();

    }

    try{

        //如果含有临时购物车，那么需要做 json 转换

        Map<String,CartItem> map =

JsonUtils.jsonToMap(cartJson,CartItem.class);

        return map;

    }catch(Exception e){

        e.printStackTrace();

    }

    return new HashMap<String,CartItem>();

}

/**

 * 2,根据商品 ID 查询商品

 */

private TbItem selectItemById(Long itemId){
```

```
TbItem tbItem = this.commonItemFeignClient.selectItemInfo(itemId);

return tbItem;

}

/**
 * 将商品添加到购物车中
 */

private void addItemToCart(Map<String,CartItem> cart,TbItem item,Integer
num,Long itemId){

    //从购物车中取商品

    CartItem cItem = cart.get(itemId.toString());

    if(cItem == null){

        //没有相同的商品

        CartItem cartItem = new CartItem();

        cartItem.setId(item.getId());

        cartItem.setImage(item.getImage());

        cartItem.setNum(num);

        cartItem.setPrice(item.getPrice());

        cartItem.setSellPoint(item.getSellPoint());

        cartItem.setTitle(item.getTitle());

        cart.put(itemId.toString(),cartItem);

    }else{
```

```
cItem.setNum(cItem.getNum()+num);

    }

}

/**
 * 4,将购物车通过 Cookie 写回给客户端浏览器
 */

private void addClientCookie(HttpServletRequest
request,HttpServletResponse response,Map<String,CartItem> cart){

    String cartJson = JsonUtils.objectToJson(cart);

    CookieUtils.setCookie(request,response,this.cart_cookie_name,cartJson,true);

}

}
```

#### 6.3.1.5在配置文件中添加购物车的 Cookie 的 key

```
#临时购物车的 key

cart_cookie_name: CART_COOKIE_NAME
```

---

## 6.3.2 查看购物车

### 6.3.2.1 修改 controller

```
/**  
 * 查看购物车  
 */  
  
@RequestMapping("/showCart")  
  
public Result showCart(String userId,HttpServletRequest  
request,HttpServletResponse response){  
  
    try{  
  
        if(StringUtils.isBlank(userId)){  
  
            //在用户未登录的状态下  
  
            return this.cookieCartService.showCart(request,response);  
  
        }else{  
  
            // 在用户已登录的状态  
  
        }  
  
    }catch(Exception e){  
  
        e.printStackTrace();  
  
    }  
  
    return Result.build(500,"error");  
  
}
```



---

### 6.3.2.2 修改 service

```
Result showCart(HttpServletRequest request, HttpServletResponse response);
```

### 6.3.2.3 修改 serviceImpl

```
/**
 * 查看购物车
 * @param request
 * @param response
 * @return
 */
@Override
public Result showCart(HttpServletRequest request, HttpServletResponse
response) {
    List<CartItem> list = new ArrayList<>();

    Map<String, CartItem> cart = this.getCart(request);

    Set<String> keys = cart.keySet();

    for(String key :keys){
        list.add(cart.get(key));
    }
}
```

```
}

    return Result.ok(list);
}
```

### 6.3.3 更新购物车中商品的数量

#### 6.3.3.1 修改 controller

```
/**
 * 修改购物车中商品的数量
 */
@RequestMapping("/updateItemNum")
public Result updateItemNum(Long itemId,String userId,Integer
num,HttpServletRequest request,HttpServletResponse response){

    try{

        if(StringUtils.isBlank(userId)){

            //在用户未登录的状态下

            return

this.cookieCartService.updateItemNum(itemId,num,request,response);

        }else{

            // 在用户已登录的状态

        }

    }
```

```
}catch(Exception e){  
    e.printStackTrace();  
}  
  
return Result.build(500,"error");  
}
```

### 6.3.3.2 修改 service

```
Result updateItemNum(Long itemId,Integer num,HttpServletRequest  
request,HttpServletResponse response);
```

### 6.3.3.3 修改 serviceImpl

```
/**  
 * 修改购物车中商品的数量  
 */  
  
@Override  
public Result updateItemNum(Long itemId, Integer num, HttpServletRequest  
request, HttpServletResponse response) {
```

```
Map<String,CartItem> cart = this.getCart(request);

CartItem item = cart.get(itemId.toString());

if(item != null){

    item.setNum(num);

}

this.addClientCookie(request,response,card);

return Result.ok();

}
```

## 6.3.4 删除购物车中的商品

### 6.3.4.1 修改 controller

```
/**
 * 删除购物车中的商品
 */

@RequestMapping("/deleteItemFromCart")

public Result deleteItemFromCart(Long itemId,String userId,HttpServletRequest
request,HttpServletResponse response){

    try{

        if(StringUtil.isBlank(userId)){

            //在用户未登录的状态下
```

```
        return

        this.cookieCartService.deleteItemFromCart(itemId,request,response);

    }else{

        // 在用户已登录的状态

    }

}

}catch(Exception e){

    e.printStackTrace();

}

return Result.build(500,"error");

}
```

#### 6.3.4.2修改 service

```
Result deleteItemFromCart(Long itemId,HttpServletRequest
request,HttpServletResponse response);
```

#### 6.3.4.3修改 serviceImpl

```
/**
 * 删除购物车中的商品
 */
```

---

@Override

```
public Result deleteItemFromCart(Long itemId, HttpServletRequest request,
HttpServletResponse response) {

    Map<String,CartItem> cart = this.getCart(request);

    cart.remove(itemId.toString());

    this.addClientCookie(request,response,card);

    return Result.ok();
}
```

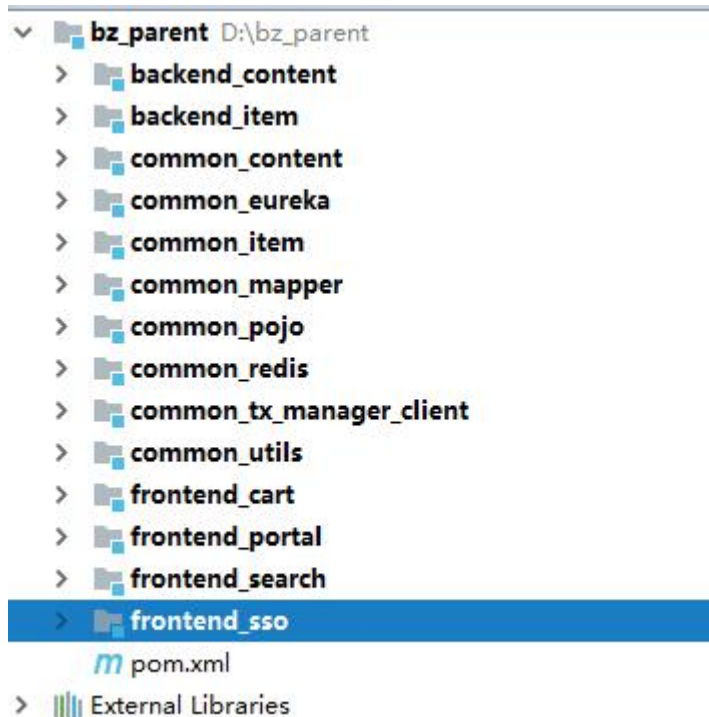
## 7 用户注册与登录服务

### 7.1 什么是单点登录

单点登录(SingleSignOn , SSO) , 就是通过用户的一次性鉴别登录。当用户在身份认证服务器上登录一次以后 , 即可获得访问单点登录系统中其他联邦系统和应用程序的权限 , 同时这种实现是不需要管理员对用户的登录状态或其他信息进行修改的 , 这意味着在多个应用系统中 , 用户只需一次登录就可以访问所有相互信任的应用系统。这种方式减少了由登录产生的时间消耗 , 辅助了用户管理 , 是目前比较流行的。

## 7.2 创建用户注册与登录服务

### 7.2.1 创建项目



### 7.2.2 修改 POM 文件，添加相关依赖

```
<?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
http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <parent>

        <artifactId>bz_parent</artifactId>

        <groupId>com.bjsxt</groupId>
```

```
<version>1.0-SNAPSHOT</version>
```

```
</parent>
```

```
<modelVersion>4.0.0</modelVersion>
```

```
<artifactId>frontend_sso</artifactId>
```

```
<dependencies>
```

```
<!--mapper-->
```

```
<dependency>
```

```
<groupId>com.bjsxt</groupId>
```

```
<artifactId>common_mapper</artifactId>
```

```
<version>1.0-SNAPSHOT</version>
```

```
</dependency>
```

```
<!--utils-->
```

```
<dependency>
```

```
<groupId>com.bjsxt</groupId>
```

```
<artifactId>common_utils</artifactId>
```

```
<version>1.0-SNAPSHOT</version>
```

```
</dependency>
```

```
<!--Spring Boot Web Starter-->
```



```
<dependency>
```

```
    <groupId>org.springframework.boot</groupId>
```

```
    <artifactId>spring-boot-starter-web</artifactId>
```

```
</dependency>
```

```
<!--Spring Cloud Eureka Client Starter-->
```

```
<dependency>
```

```
    <groupId>org.springframework.cloud</groupId>
```

```
    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
```

```
</dependency>
```

```
<!--Spring Cloud OpenFeign Starter-->
```

```
<dependency>
```

```
    <groupId>org.springframework.cloud</groupId>
```

```
    <artifactId>spring-cloud-starter-openfeign</artifactId>
```

```
</dependency>
```

```
<dependency>
```

```
    <groupId>com.bjsxt</groupId>
```

```
    <artifactId>common_tx_manager_client</artifactId>
```

```
    <version>1.0-SNAPSHOT</version>
```

```
</dependency>
```

```
</dependencies>

<build>

  <plugins>

    <plugin>

      <groupId>org.springframework.boot</groupId>

      <artifactId>spring-boot-maven-plugin</artifactId>

    </plugin>

  </plugins>

</build>

</project>
```

### 7.2.3 修改配置文件，添加相关配置

**spring:**

**application:**

**name:** frontend-sso

**datasource:**

**driverClassName:** com.mysql.jdbc.Driver

**url:** jdbc:mysql://localhost:3306/bz\_shop?characterEncoding=UTF-8

**username:** root

**password:** root

---

**type:** com.alibaba.druid.pool.DruidDataSource

**server:**

**port:** 9090

**eureka:**

**client:**

**serviceUrl:**

**defaultZone:** http://eureka-server:8761/eureka/

**tx-lcn:**

**client:**

**manager-address:** 192.168.70.157:8070

## 7.2.4 创建启动类，添加相关注解

```
/**
```

```
 * 用户注册于登录服务
```

```
 */
```

```
@SpringBootApplication
```

```
@EnableDiscoveryClient
```

```
@EnableFeignClients

@MapperScan("com.bjsxt.mapper")

public class FrontendSSOApplication {

    public static void main(String[] args){

        SpringApplication.run(FrontendSSOApplication.class,args);

    }

}
```

## 7.3 注册信息认证接口

### 7.3.1 创建 controller

```
/**

 * 用户注册于登录

 */

@RestController

@RequestMapping("/sso")

public class SSOController {

    @Autowired

    private SSOService ssoService;

}
```

```
* 对用户的注册信息(用户名与电话号码)做数据校验
*/

@RequestMapping("/checkUserInfo/{checkValue}/{checkFlag}")

public Result checkUserInfo(@PathVariable String checkValue,@PathVariable
Integer checkFlag){

    try{

        return this.ssoService.checkUserInfo(checkValue,checkFlag);

    }catch(Exception e){

        e.printStackTrace();

    }

    return Result.build(500,"error");

}

}
```

### 7.3.2 创建 service

```
public interface SSOService {

    Result checkUserInfo(String checkValue,Integer checkFlag);

}
```

### 7.3.3 创建 serviceImpl

```
/**
 * 用户注册与登录业务层
 */

@Service

public class SSOServiceImpl implements SSOService {

    @Autowired

    private TbUserMapper tbUserMapper;

    /**
     * 对用户的注册信息(用户名与电话号码)做数据校验
     */

    @Override

    public Result checkUserInfo(String checkValue, Integer checkFlag) {

        TbUserExample example = new TbUserExample();

        TbUserExample.Criteria criteria = example.createCriteria();

        if(checkFlag == 1){

            criteria.andUsernameEqualTo(checkValue);

        }else if(checkFlag == 2){

            criteria.andPhoneEqualTo(checkValue);

        }
    }
}
```

```
    }

    Integer rows = this.tbUserMapper.countByExample(example);

    if(rows > 0){

        return Result.error("数据不可用");

    }

    return Result.ok(checkValue);

}

}
```

## 7.4 用户注册接口

### 7.4.1 修改 controller

```
/**
 * 用户注册
 */
@RequestMapping("/userRegister")
public Result userRegister(TbUser user){

    try{

        return this.ssoService.userRegister(user);

    }catch(Exception e){

        e.printStackTrace();

    }

}
```

```
}

    return Result.build(500,"error");
}
```

### 7.4.2 修改 service

```
Result userRegister(TbUser user);
```

### 7.4.3 修改 serviceImpl

```
/**
 * 用户注册
 */
@Override
@LcnTransaction
public Result userRegister(TbUser user) {

    //将密码做加密处理。

    String pwd = MD5Utils.digest(user.getPassword());

    user.setPassword(pwd);

    //补齐数据

    user.setCreated(new Date());
```



```
user.setUpdated(new Date());

this.tbUserMapper.insert(user);

return Result.ok();
}
```

## 7.5 用户登录接口

### 7.5.1 在 common\_redis 服务中实现将用户添加到缓存接口

#### 7.5.1.1 创建 controller

```
/**
 * 缓存用户 Controller
 */
@RestController
@RequestMapping("/sso/redis")
public class SSOController {

    @Autowired

    private SSOService ssoService;

}
```

```
    * 将用于缓存到 Redis 中

    */

    @RequestMapping("/insertUser")

    public void insertUser(@RequestBody TbUser tbUser,@RequestParam String
userToken){

        this.ssoService.insertUser(tbUser,userToken);

    }

}
```

#### 7.5.1.2创建 service

```
public interface SSOService {

    void insertUser(TbUser tbUser, String userToken);

}
```

#### 7.5.1.3创建 serviceImpl

```
/**

    * 缓存用户业务层

    */

@Service
```

```
public class SSOServiceImpl implements SSOService {

    @Value("${user_session_redis_key}")

    private String user_session_redis_key;

    @Autowired

    private RedisTemplate<String,Object> redisTemplate;

    @Override

    public void insertUser(TbUser tbUser, String userToken) {

        //将密码置空，处于安全角度考虑

        tbUser.setPassword("");

        this.redisTemplate.opsForValue().set(this.user_session_redis_key+":"+userToken
        ,tbUser,1, TimeUnit.DAYS);

    }

}
```

---

## 7.5.2 在 frontend\_sso 服务中实现用户登录接口

### 7.5.2.1 需改 controller

```
/**
 * 用户登录
 */
@RequestMapping("/userLogin")
public Result userLogin(String username,String password){

    try{

        return this.ssoService.userLogin(username,password);

    }catch(Exception e){

        e.printStackTrace();

    }

    return Result.build(500,"error");
}
```

### 7.5.2.2 修改 service

```
Result userLogin(String username,String password);
```

### 7.5.2.3 修改 serviceImpl

```
/**
 * 用户登录
 * @param username
 * @param password
 * @return
 */
@Override
public Result userLogin(String username, String password) {
    //根据用户名密码查询数据库

    TbUser tbUser = this.login(username,password);

    if(tbUser == null){
        return Result.error("用户名或密码有误，请重新输入");
    }

    //将用户添加到 Redis 中

    String userToken = UUID.randomUUID().toString();

    Integer flag = this.insertUserToRedis(tbUser,userToken);

    if(flag == 500){
        return Result.error("登录失败");
    }

    Map<String,String> map = new HashMap<>();

    map.put("token",userToken);
}
```

```
map.put("userid",tbUser.getId().toString());

map.put("username",tbUser.getUsername());

return Result.ok(map);
}

/**
 * 用户登录业务处理
 */

private TbUser login(String username,String password){

    String pwd = MD5Utils.digest(password);

    TbUserExample example = new TbUserExample();

    TbUserExample.Criteria criteria = example.createCriteria();

    criteria.andUsernameEqualTo(username);

    criteria.andPasswordEqualTo(pwd);

    List<TbUser> list = this.tbUserMapper.selectByExample(example);

    if(list == null || list.size() <= 0){

        return null;

    }

    return list.get(0);

}

/**
 * 将用户添加到 Redis 中
 */
```

```
private Integer insertUserToRedis(TbUser tbUser,String userToken){

    try{

        this.commonRedisFeignClient.insertUser(tbUser,userToken);

        return 200;

    }catch(Exception e){

        e.printStackTrace();

    }

    return 500;

}
```

#### 7.5.2.4创建 feignClient

```
@FeignClient(value = "common-redis")

public interface CommonRedisFeignClient {

    @PostMapping("/sso/redis/insertUser")

    void insertUser(@RequestBody TbUser tbUser, @RequestParam("userToken")

String userToken);

}
```

---

## 7.6 用户退出登录接口

### 7.6.1 在 common\_redis 服务中实现用户退出登录接口

#### 7.6.1.1 修改 controller

```
/**  
 * 用户退出登录  
 */  
  
@RequestMapping("/logOut")  
  
public void logOut(@RequestParam String userToken){  
    this.ssoService.logOut(userToken);  
}
```

#### 7.6.1.2 修改 service

```
void logOut(@RequestParam String userToken);
```

#### 7.6.1.3 修改 serviceImpl

```
/**  
 * 用户退出登录  
 * @param userToken
```



```
*/  
  
@Override  
  
public void logOut(String userToken) {  
  
    this.redisTemplate.delete(this.user_session_redis_key+":"+userToken);  
  
}
```

## 7.6.2 在 frontend\_sso 服务中实现用户退出登录接口

### 7.6.2.1 修改 controller

```
/**  
 * 用户退出登录  
 */  
  
@RequestMapping("/logOut")  
  
public Result logOut(String token){  
  
    try{  
  
        return this.ssoService.logOut(token);  
  
    }catch(Exception e){  
  
        e.printStackTrace();  
  
    }  
  
    return Result.build(500,"error");  
}
```

---

```
}
```

### 7.6.2.2 修改 service

```
Result logOut(String token);
```

### 7.6.2.3 修改 serviceImpl

```
/**  
 * 用户退出登录  
 * @param token  
 * @return  
 */  
  
@Override  
public Result logOut(String token) {  
    this.commonRedisFeignClient.logOut(token);  
    return Result.ok();  
}
```

---

#### 7.6.2.4修改 feignClient

```
@PostMapping("/sso/redis/logOut")  
  
void logOut(@RequestParam("userToken") String userToken);
```

## 8 用户登录状态下的购物车操作

### 8.1 将商品添加到购物车

#### 8.1.1 在 common\_redis 服务中实现缓存购物车接口

##### 8.1.1.1创建 controller

```
/**  
  
 * 购物车操作  
  
 */  
  
@RestController  
  
@RequestMapping("/redis/cart")  
  
public class CartController {  
  
    @Autowired  
  
    private CartService cartService;  
  
    /**  
  
     * 将购物车缓存到 redis 中
```

```
*/

@RequestMapping("/insertCart")

public void insertCart(@RequestBody Map<String,Object> map){

    this.cartService.insertCart(map);

}

}
```

#### 8.1.1.2 创建 service

```
public interface CartService {

    void insertCart(Map<String,Object> map);

}
```

#### 8.1.1.3 创建 serviceImpl

```
/**

 * 购物车操作业务层

 */

@Service

public class CartServiceImpl implements CartService {
```

@Autowired

**private** RedisTemplate<String,Object> **redisTemplate**;

@Value("\${frontend\_cart\_redis\_key}")

**private** String **frontend\_cart\_redis\_key**;

/\*\*

\* 缓存购物车

\* @param map

\*/

@Override

**public void** insertCart(Map<String, Object> map) {

String userId = (String) map.get("userId");

Map<String,CartItem> cart = (Map<String, CartItem>) map.get("cart");

**this.redisTemplate.opsForHash().put(this.frontend\_cart\_redis\_key,userId,card);**

}

}

#### 8.1.1.4修改配置文件，添加缓存购物的 key

#缓存购物车的 key

---

**frontend\_cart\_redis\_key:** frontend:cart:redis:key

## 8.1.2 在 common\_redis 服务中实现查询购物车接口

### 8.1.2.1修改 controller

```
/**
 * 根据用户 ID 查询用户购物车
 */
@RequestMapping("/selectCartByUserId")
public Map<String,CartItem> selectCartByUserId(@RequestParam String userId){
    return this.cartService.selectCartByUserId(userId);
}
```

### 8.1.2.2修改 service

```
Map<String,CartItem> selectCartByUserId(String userId);
```

### 8.1.2.3修改 serviceImpl

```
/**
 * 根据用户 ID 查询用户购物车
```

```
*/  
  
@Override  
  
public Map<String, CartItem> selectCartByUserId(String userId) {  
  
    return (Map<String, CartItem>)  
  
this.redisTemplate.opsForHash().get(this.frontend_cart_redis_key,userId);  
  
}
```

### 8.1.3 在 frontend\_cart 服务中实现用户登录状态下向购物车中添加商品

#### 8.1.3.1 修改 controller

```
/**  
  
 * 将商品加入购物车  
  
*/  
  
@RequestMapping("/addItem")  
  
public Result addItem(Long itemId, String userId, @RequestParam(defaultValue =  
"1") Integer num, HttpServletRequest request, HttpServletResponse response){  
  
    try{  
  
        if(StringUtils.isBlank(userId)){  
  
            //在用户未登录的状态下
```

```
        return this.cookieCartService.addItem(itemId,num,request,response);

    }else{

        // 在用户已登录的状态

        return this.redisCartService.addItem(itemId,num,userId);

    }

}

}catch(Exception e){

    e.printStackTrace();

}

return Result.build(500,"error");

}
```

### 8.1.3.2 创建 service

```
public interface RedisCartService {

    Result addItem(Long itemId, Integer num,String userId);

}
```

### 8.1.3.3 创建 serviceImpl

```
/**
```



---

*\* 用户已登录状态下的购物车操作业务*

*\*/*

**@Service**

**public class** RedisCartServiceImpl **implements** RedisCartService {

**@Autowired**

**private** CommonItemFeignClient **commonItemFeignClient**;

**@Autowired**

**private** CommonRedisFeignClient **commonRedisFeignClient**;

**@Override**

**public** Result addItem(Long itemId, Integer num, String userId) {

*//1,查询商品*

TbItem tbItem = **this**.selectItemById(itemId);

*//2,获取购物车*

Map<String,CartItem> cart = **this**.getCart(userId);

*//3,将商品添加大购物车中*

**this**.addItemToCart(cart,tbItem,num,itemId);

*//4,将购物车缓存到 redis 中*

**this**.addCartToRedis(userId,card);

**return** Result.ok();

```
}

/**
 * 1 根据商品 Id 查询商品
 */

private TbItem selectItemById(Long itemId){

    return this.commonItemFeignClient.selectItemInfo(itemId);

}

/**
 * 2 根据用户 ID 获取 Redis 中的购物车
 */

private Map<String,CartItem> getCart(String userId){

    try{

        Map<String,CartItem> cart =

this.commonRedisFeignClient.selectCartByUserId(userId);

        if(cart == null){

            cart = new HashMap<String,CartItem>();

        }

        return cart;

    }catch(Exception e){

        e.printStackTrace();

    }

}
```

```
        return new HashMap<String,CartItem>();

    }

    /**
     * 3 将商品添加到购物车中
     */

    private void addItemToCart(Map<String,CartItem> cart,TbItem item,Integer
num,Long itemId){

        CartItem cItem = cart.get(itemId.toString());

        if(cItem == null){

            //没有相同的商品

            CartItem cartItem = new CartItem();

            cartItem.setId(item.getId());

            cartItem.setImage(item.getImage());

            cartItem.setNum(num);

            cartItem.setPrice(item.getPrice());

            cartItem.setSellPoint(item.getSellPoint());

            cartItem.setTitle(item.getTitle());

            cart.put(item.getId().toString(),cartItem);

        }else{

            cItem.setNum(cItem.getNum()+num);

        }

    }

}
```

```
/**
 * 4,将购物车缓存到 redis 中
 */

private void addCartToRedis(String userId,Map<String,CartItem> cart){

    Map<String,Object> map = new HashMap<>();

    map.put("userId",userId);

    map.put("cart",cart);

    this.commonRedisFeignClient.insertCart(map);

}

}
```

#### 8.1.3.4创建 feignClient

```
@FeignClient(value = "common-redis")

public interface CommonRedisFeignClient {

    @PostMapping("/redis/cart/selectCartByUserId")

    Map<String,CartItem> selectCartByUserId(@RequestParam("userId") String

userId);

    @PostMapping("/redis/cart/insertCart")
```

```
void insertCart(@RequestBody Map<String,Object> map);  
}
```

## 8.1.4 在 frontend\_cart 服务中实现用户登录状态下查看购物车中商品

### 8.1.4.1 修改 controller

```
/**  
 * 查看购物车  
 */  
@RequestMapping("/showCart")  
public Result showCart(String userId,HttpServletRequest  
request,HttpServletResponse response){  
    try{  
        if(StringUtils.isBlank(userId)){  
            //在用户未登录的状态下  
            return this.cookieCartService.showCart(request,response);  
        }else{  
            // 在用户已登录的状态  
            return this.redisCartService.showCart(userId);  
        }  
    }  
}
```

```
}catch(Exception e){  
    e.printStackTrace();  
}  
  
return Result.build(500,"error");  
}
```

#### 8.1.4.2 修改 service

```
Result showCart(String userId);
```

#### 8.1.4.3 修改 serviceImpl

```
/**  
 * 查看购物车  
 * @param userId  
 * @return  
 */  
  
@Override  
public Result showCart(String userId) {  
    List<CartItem> list = new ArrayList<>();
```

```
Map<String,CartItem> cart = this.getCart(userId);

Set<String> keys = cart.keySet();

for(String key :keys){

    list.add(cart.get(key));

}

return Result.ok(list);

}
```

## 8.1.5 在 frontend\_cart 服务中实现用户登录状态下修改购物车商品数量

### 8.1.5.1 修改 controller

```
/**
 * 修改购物车中商品的数量
 */

@RequestMapping("/updateItemNum")

public Result updateItemNum(Long itemId,String userId,Integer
num,HttpServletRequest request,HttpServletResponse response){

    try{

        if(StringUtils.isBlank(userId)){

            //在用户未登录的状态下
```

```
        return

        this.cookieCartService.updateItemNum(itemId,num,request,response);

        }else{

            // 在用户已登录的状态

            return this.redisCartService.updateItemNum(itemId,num,userId);

        }

    }catch(Exception e){

        e.printStackTrace();

    }

    return Result.build(500,"error");

}
```

#### 8.1.5.2修改 service

```
Result updateItemNum(Long itemId, Integer num,String userId);
```

#### 8.1.5.3修改 serviceImpl

```
/**
 * 修改购物车中的商品数量
```



---

*\* @param itemId*

*\* @param num*

*\* @param userId*

*\* @return*

*\*/*

@Override

**public** Result updateItemNum(Long itemId, Integer num, String userId) {

Map<String,CartItem> cart = **this**.getCart(userId);

CartItem item = cart.get(itemId.toString());

**if**(item != **null**){

item.setNum(num);

}

*//将新的购物车缓存到 Redis 中*

**this**.addCartToRedis(userId, cart);

**return** Result.ok();

}

## 8.1.6 在 frontend\_cart 服务中实现用户登录状态下删除购物车中的商品

### 8.1.6.1 修改 controller

```
/**
 * 删除购物车中的商品
 */
@RequestMapping("/deleteItemFromCart")
public Result deleteItemFromCart(Long itemId,String userId,HttpServletRequest
request,HttpServletResponse response){

    try{

        if(StringUtils.isBlank(userId)){

            //在用户未登录的状态下

            return

this.cookieCartService.deleteItemFromCart(itemId,request,response);

        }else{

            // 在用户已登录的状态

            return this.redisCartService.deleteItemFromCart(itemId,userId);

        }

    }catch(Exception e){

        e.printStackTrace();

    }

}
```

```
return Result.build(500,"error");  
}
```

### 8.1.6.2修改 service

```
Result deleteItemFromCart(Long itemId,String userId);
```

### 8.1.6.3修改 serviceImpl

```
/**  
 * 删除购物车中的商品  
 * @param itemId  
 * @param userId  
 * @return  
 */  
  
@Override  
public Result deleteItemFromCart(Long itemId, String userId) {  
    Map<String,CartItem> cart = this.getCart(userId);  
    cart.remove(itemId.toString());  
    //将新的购物车缓存到 Redis 中  
    this.addCartToRedis(userId,card);
```

```
        return Result.ok();  
    }  
}
```

## 8.1.7 在 frontend\_sso 服务用户登录业务中实现同步购物车

### 8.1.7.1 修改 controller

```
/**  
 * 用户登录  
 */  
@RequestMapping("/userLogin")  
public Result userLogin(String username,String password,HttpServletRequest  
request){  
    try{  
        return this.ssoService.userLogin(username,password,request);  
    }catch(Exception e){  
        e.printStackTrace();  
    }  
    return Result.build(500,"error");  
}
```

---

### 8.1.7.2 修改 service

```
Result userLogin(String username,String password,HttpServletRequest request);
```

### 8.1.7.3 修改 serviceImpl

```
/**
 * 用户登录
 * @param username
 * @param password
 * @return
 */
@Override
public Result userLogin(String username, String password,HttpServletRequest
request) {
    //根据用户名密码查询数据库
    TbUser tbUser = this.login(username,password);
    if(tbUser == null){
        return Result.error("用户名或密码有误，请重新输入");
    }
    //将用户添加到 Redis 中
```

```
String userToken = UUID.randomUUID().toString();

Integer flag = this.insertUserToRedis(tbUser,userToken);

if(flag == 500){

    return Result.error("登录失败");

}

Map<String,String> map = new HashMap<>();

map.put("token",userToken);

map.put("userid",tbUser.getId().toString());

map.put("username",tbUser.getUsername());


//将临时购物车中的商品同步到永久购物车中

this.syncCart(tbUser.getId().toString(),request);

return Result.ok(map);

}

/**
 * 同步购物车
 */

private void syncCart(String userId,HttpServletRequest request){

    //获取临时购物车

    Map<String,CartItem> cookieCart = this.getCart(request);

    //获取永久购物车

    Map<String,CartItem> redisCart = this.getCart(userId);
```

```
//删除永久购物车中所包含临时购物车中的商品

Set<String> keys = cookieCart.keySet();

for(String key:keys){

    redisCart.remove(key);

}

//将同步后的购物车缓存到 redis 中

redisCart.putAll(cookieCart);

//将永久购物车重新缓存到 redis 中

this.addCartToRedis(userId,redisCart);

}

/**
 * 获取临时购物车
 */

private Map<String,CartItem> getCart(HttpServletRequest request){

    //临时购物车已存在

    String cartJson=

    CookieUtils.getCookieValue(request,this.cart_cookie_name,true);

    if(StringUtils.isBlank(cartJson)){

        //临时购物车不存在

        return new HashMap<String,CartItem>();

    }

    try{
```

```
//如果含有临时购物车, 那么需要做 json 转换

Map<String,CartItem> map =

JsonUtils.jsonToMap(cartJson,CartItem.class);

    return map;

}catch(Exception e){

    e.printStackTrace();

}

    return new HashMap<String,CartItem>();

}

/**

 * 获取永久购物车

 */

private Map<String,CartItem> getCart(String userId){

    try{

        Map<String,CartItem> cart =

this.commonRedisFeignClient.selectCartByUserId(userId);

        if(cart == null){

            cart = new HashMap<String,CartItem>();

        }

        return cart;

    }catch(Exception e){

        e.printStackTrace();

    }

}
```



```
}

    return new HashMap<String,CartItem>();
}

/**
 * 将永久购物车重新缓存到 redis 中
 */
private void addCartToRedis(String userId,Map<String,CartItem> cart){

    Map<String,Object> map = new HashMap<>();

    map.put("userId",userId);

    map.put("cart",cart);

    this.commonRedisFeignClient.insertCart(map);
}
```

#### 8.1.7.4修改配置文件添加购物车的 key

```
#临时购物车的 key

cart_cookie_name: CART_COOKIE_NAME
```

---

## 8.1.8 在 frontend\_sso 服务用户登录业务中实现删除临时购物车

### 8.1.8.1修改 controller

```
/**
 * 用户登录
 */
@RequestMapping("/userLogin")
public Result userLogin(String username,String password,HttpServletRequest
request,HttpServletResponse response){
    try{
        return this.ssoService.userLogin(username,password,request,response);
    }catch(Exception e){
        e.printStackTrace();
    }
    return Result.build(500,"error");
}
```

### 8.1.8.2修改 service

```
Result userLogin(String username,String password,HttpServletRequest
request,HttpServletResponse response);
```

### 8.1.8.3 修改 serviceImpl

```
/**
 * 用户登录
 * @param username
 * @param password
 * @return
 */
@Override
public Result userLogin(String username, String password, HttpServletRequest
request, HttpServletResponse response) {
    //根据用户名密码查询数据库

    TbUser tbUser = this.login(username,password);

    if(tbUser == null){
        return Result.error("用户名或密码有误，请重新输入");
    }

    //将用户添加到 Redis 中

    String userToken = UUID.randomUUID().toString();

    Integer flag = this.insertUserToRedis(tbUser,userToken);

    if(flag == 500){
        return Result.error("登录失败");
    }
}
```

```
}

Map<String,String> map = new HashMap<>();

map.put("token",userToken);

map.put("userid",tbUser.getId().toString());

map.put("username",tbUser.getUsername());


//将临时购物车中的商品同步到永久购物车中

this.syncCart(tbUser.getId().toString(),request);

//同步购物车成功后，需要将临时购物车中的商品删除掉

this.deleteCookieCart(request,response);

return Result.ok(map);
}
```

## 9 用户结算接口

### 9.1 在 frontend\_cart 服务中实现结算接口

#### 9.1.1 修改 controller

```
/**
 * 去结算
 */
```

```
@RequestMapping("/goSettlement")

public Result goSettlement(String[] ids,String userId){

    try{

        return this.redisCartService.goSettlement(ids,userId);

    }catch(Exception e){

        e.printStackTrace();

    }

    return Result.build(500,"error");

}
```

### 9.1.2 修改 service

```
Result goSettlement(String[] ids,String userId);
```

### 9.1.3 修改 serviceImpl

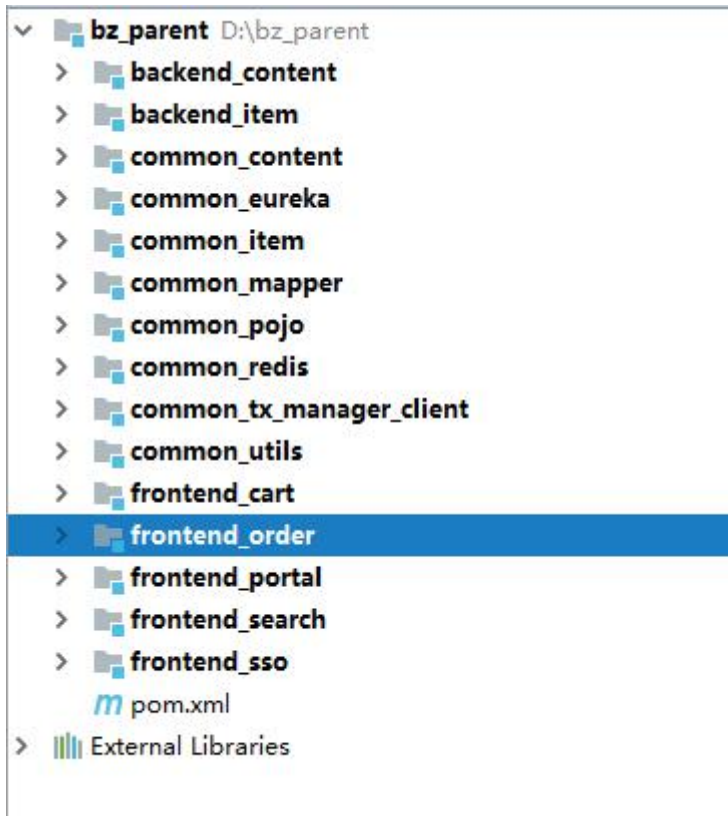
```
/**
 * 结算
 * @param ids
 * @param userId
 * @return
```

```
*/  
  
@Override  
  
public Result goSettlement(String[] ids, String userId) {  
  
    //获取购物车  
  
    Map<String,CartItem> cart = this.getCart(userId);  
  
    //从购物车中获取选中的商品  
  
    List list = this.getItemList(cart,ids);  
  
    return Result.ok(list);  
}  
  
/**  
  
 * 从购物车中获取选中的商品  
  
 */  
  
private List<CartItem> getItemList(Map<String,CartItem> cart,String[] ids){  
  
    List<CartItem> list = new ArrayList<>();  
  
    for(String id:ids){  
  
        list.add(cart.get(id));  
  
    }  
  
    return list;  
}
```

## 10 订单服务

### 10.1 创建订单服务

#### 10.1.1 创建项目



#### 10.1.2 修改 POM 文件，添加相关依赖

```
<?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
http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <parent>

    <artifactId>bz_parent</artifactId>
```

```
<groupId>com.bjsxt</groupId>
```

```
<version>1.0-SNAPSHOT</version>
```

```
</parent>
```

```
<modelVersion>4.0.0</modelVersion>
```

```
<artifactId>frontend_order</artifactId>
```

```
<dependencies>
```

```
<!--mapper-->
```

```
<dependency>
```

```
<groupId>com.bjsxt</groupId>
```

```
<artifactId>common_mapper</artifactId>
```

```
<version>1.0-SNAPSHOT</version>
```

```
</dependency>
```

```
<!--utils-->
```

```
<dependency>
```

```
<groupId>com.bjsxt</groupId>
```

```
<artifactId>common_utils</artifactId>
```

```
<version>1.0-SNAPSHOT</version>
```

```
</dependency>
```



---

```
<!--Spring Boot Web Starter-->
```

```
<dependency>
```

```
  <groupId>org.springframework.boot</groupId>
```

```
  <artifactId>spring-boot-starter-web</artifactId>
```

```
</dependency>
```

```
<!--Spring Cloud Eureka Client Starter-->
```

```
<dependency>
```

```
  <groupId>org.springframework.cloud</groupId>
```

```
  <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
```

```
</dependency>
```

```
<dependency>
```

```
  <groupId>com.bjsxt</groupId>
```

```
  <artifactId>common_tx_manager_client</artifactId>
```

```
  <version>1.0-SNAPSHOT</version>
```

```
</dependency>
```

```
<!--Spring Cloud OpenFeign Starter-->
```

```
<dependency>
```

```
  <groupId>org.springframework.cloud</groupId>
```

```
  <artifactId>spring-cloud-starter-openfeign</artifactId>
```

```
</dependency>
```

```
</dependencies>
```

```
</project>
```

---

### 10.1.3 修改配置文件，添加相关配置

**spring:**

**application:**

**name:** frontend-order

**datasource:**

**driverClassName:** com.mysql.jdbc.Driver

**url:** jdbc:mysql://localhost:3306/bz\_shop?characterEncoding=UTF-8

**username:** root

**password:** root

**type:** com.alibaba.druid.pool.DruidDataSource

**server:**

**port:** 9111

**eureka:**

**client:**

**serviceUrl:**

**defaultZone:** http://eureka-server:8761/eureka/

**tx-lcn:**

---

**client:**

**manager-address:** 192.168.70.157:8070

#### 10.1.4 创建启动类，添加相关注解

```
/**
 * 订单服务
 */

@SpringBootApplication
@EnableDiscoveryClient
@EnableDistributedTransaction
@MapperScan("com.bjsxt.mapper")
public class FrontendOrderApplication {

    public static void main(String[] args){

        SpringApplication.run(FrontendOrderApplication.class,args);

    }

}
```

---

## 10.2 在 common\_redis 服务中实现生成订单 ID 接口

### 10.2.1 创建 controller

```
/**
 * 订单业务
 */
@RestController
@RequestMapping("/redis/order")
public class OrderController {

    @Autowired
    private OrderService orderService;

    /**
     * 生成订单 ID
     */
    @RequestMapping("/selectOrderId")
    public Long selectOrderId(){
        return this.orderService.selectOrderId();
    }
}
```

---

### 10.2.2 创建 service

```
public interface OrderService {  
  
    Long selectOrderId();  
  
}
```

### 10.2.3 创建 serviceImpl

```
/**  
 * 订单处理业务层  
 */  
  
@Service  
  
public class OrderServiceImpl implements OrderService {  
  
    @Value("${order_item_id_key}")  
  
    private String order_item_id_key;  
  
    @Value("${init_item_id}")  
  
    private Long init_item_id;  
  
    @Autowired
```

```
private RedisTemplate<String,Object> redisTemplate;

@Override

public Long selectOrderId() {

    //查询 redis 中的订单 Id

    Integer value = (Integer)

this.redisTemplate.opsForValue().get(this.order_item_id_key);

    //如果订单 ID 的值不存在，将默认值 set 到 redis 中

    if(value == null || value <= 0){

this.redisTemplate.opsForValue().set(this.order_item_id_key,this.init_item_id);

    }

    //通过 redis 中的自增长将默认值做递增处理。

    Long orderId =

this.redisTemplate.opsForValue().increment(this.order_item_id_key);

    return orderId;

}

}
```

---

## 10.2.4 修改配置文件添加缓存订单 ID 的 key 以及订单 ID 初始值

*#缓存订单 ID 的 key*

**order\_item\_id\_key:** order:item:id:key

*#订单 Id 初始值*

**init\_item\_id:** 2000

## 10.3 在 frontend\_order 服务中实现创建订单接口

### 10.3.1 创建 controller

```
/**  
 * 订单服务 Controller  
 */  
  
@RestController  
@RequestMapping("/order")  
public class OrderController {  
  
    @Autowired  
    private OrderService orderService;  
  
    /**
```

```
* 创建订单

*/

@RequestMapping("/insertOrder")

public Result insertOrder(String orderItem, TbOrder tbOrder ,
TbOrderShipping tbOrderShipping){

    try{

        Result res = Result.formatObjectToList(orderItem, TbOrderItem.class);

        List<TbOrderItem> list = (List<TbOrderItem>) res.getData();

        return this.orderService.insertOrder(list,tbOrder,tbOrderShipping);

    }catch(Exception e){

        e.printStackTrace();

    }

    return Result.build(500,"error");

}

}
```

### 10.3.2 创建 service

```
public interface OrderService {

    Result insertOrder(List<TbOrderItem> tbOrderItem, TbOrder tbOrder ,
```



```
TbOrderShipping tbOrderShipping);  
  
}
```

### 10.3.3 创建 serviceImpl

```
/**  
 * 订单服务业务层  
 */  
  
@Service  
  
public class OrderServiceImpl implements OrderService {  
  
    @Autowired  
  
    private CommonRedisFeignClient commonRedisFeignClient;  
  
    @Autowired  
  
    private TbOrderMapper tbOrderMapper;  
  
    @Autowired  
  
    private TbOrderItemMapper tbOrderItemMapper;  
  
    @Autowired  
  
    private TbOrderShippingMapper tbOrderShippingMapper;
```

```
/**
 * 创建订单
 * @param tbOrderItem
 * @param tbOrder
 * @param tbOrderShipping
 * @return
 */

@Override
@LcnTransaction
public Result insertOrder(List< TbOrderItem> tbOrderItem, TbOrder tbOrder,
TbOrderShipping tbOrderShipping) {

    //获取订单 ID

    Long orderId = this.commonRedisFeignClient.selectOrderId();

    //补齐 TbOrder 数据

    tbOrder.setOrderId(orderId.toString());

    //状态：1、未付款，2、已付款，3、未发货，4、已发货，5、交易成功，6、交
    易关闭

    tbOrder.setStatus(1);

    Date date = new Date();

    tbOrder.setCreateTime(date);
}
```

```
tbOrder.setUpdateTime(date);

//0:未评价 1 : 已评价

tbOrder.setBuyerRate(0);

//将订单插入到数据库中

this.tbOrderMapper.insert(tbOrder);


//插入订单中所包含的商品

for(TbOrderItem item:tbOrderItem){

    item.setId(IDUtils.genItemId()+ "");

    item.setOrderId(orderId.toString());

    this.tbOrderItemMapper.insert(item);

}


//插入物流表数据

tbOrderShipping.setOrderId(orderId.toString());

tbOrderShipping.setUpdated(date);

tbOrderShipping.setCreated(date);

this.tbOrderShippingMapper.insert(tbOrderShipping);


return Result.ok(orderId);

}
```

```
}
```

### 10.3.4 创建 feignClient

```
@FeignClient(value = "common-redis")

public interface CommonRedisFeignClient {

    @PostMapping("/redis/order/selectOrderId")

    Long selectOrderId();

}
```

### 10.3.5 修改启动类，添加开启 feign

```
@SpringBootApplication

@EnableDiscoveryClient

@EnableDistributedTransaction

@EnableFeignClients

@MapperScan("com.bjsxt.mapper")

public class FrontendOrderApplication {

    public static void main(String[] args){

        SpringApplication.run(FrontendOrderApplication.class,args);

    }

}
```

```
}  
  
}
```

## 10.4 提交订单后将订单中所包含的商品从购物车中删除

### 10.4.1 修改 frontend\_cart 服务的 controller

```
/**  
 * 删除购物车中的商品  
 */  
  
@RequestMapping("/deleteItemFromCart")  
  
public Result deleteItemFromCart(@RequestParam Long itemId,  
@RequestParam String userId,HttpServletRequest request,HttpServletResponse  
response){  
  
    try{  
  
        if(StringUtils.isBlank(userId)){  
  
            //在用户未登录的状态下  
  
            return  
  
this.cookieCartService.deleteItemFromCart(itemId,request,response);  
  
        }else{  
  
            // 在用户已登录的状态  
  
            return this.redisCartService.deleteItemFromCart(itemId,userId);  
  
        }  
  
    }  
  
}
```

```
    }

    }catch(Exception e){

        e.printStackTrace();

    }

    return Result.build(500,"error");

}
```

#### 10.4.2 创建 feignClient

```
@FeignClient(value = "frontend-cart")

public interface FrontendCartFeignClient {

    @PostMapping("/cart/deleteItemFromCart")

    Result deleteItemFromCart(@RequestParam("itemId") Long itemId,

    @RequestParam("userId") String userId);

}
```

#### 10.4.3 修改 frontend\_order 服务的 serviceImpl

```
/**
```

---

\* 创建订单

\* @param tbOrderItem

\* @param tbOrder

\* @param tbOrderShipping

\* @return

\*/

@Override

@LcnTransaction

```
public Result insertOrder(List< TbOrderItem> tbOrderItem, TbOrder tbOrder,  
TbOrderShipping tbOrderShipping) {
```

```
    //获取订单 ID
```

```
    Long orderId = this.commonRedisFeignClient.selectOrderId();
```

```
    //补齐 TbOrder 数据
```

```
    tbOrder.setOrderId(orderId.toString());
```

```
    //状态：1、未付款，2、已付款，3、未发货，4、已发货，5、交易成功，6、交易关  
    闭
```

```
    tbOrder.setStatus(1);
```

```
    Date date = new Date();
```

```
    tbOrder.setCreateTime(date);
```

```
    tbOrder.setUpdateTime(date);
```

```
    //0:未评价 1：已评价
```

```
tbOrder.setBuyerRate(0);

//将订单插入到数据库中

this.tbOrderMapper.insert(tbOrder);


//插入订单中所包含的商品

for(TbOrderItem item:tbOrderItem){

    item.setId(IDUtils.genItemId()+"");

    item.setOrderId(orderId.toString());

    this.tbOrderItemMapper.insert(item);

    //将订单中的商品从购物车中删除

    this.frontendCartFeignClient.deleteItemFromCart(Long.parseLong(item.getItemId()),tbOrder.getUserId().toString());

}


//插入物流表数据

tbOrderShipping.setOrderId(orderId.toString());

tbOrderShipping.setUpdated(date);

tbOrderShipping.setCreated(date);

this.tbOrderShippingMapper.insert(tbOrderShipping);
```



```
        return Result.ok(orderId);
    }
}
```

## 10.5 结算时检查用户是否登录

### 10.5.1 在 common\_redis 服务中实现检查用户是否登录接口

#### 10.5.1.1 修改 controller

```
/**
 * 根据用户 token 校验用户在 redis 中是否失效
 */
@RequestMapping("/checkUserToken")
public TbUser checkUserToken(@RequestParam String token){
    return this.ssoService.checkUserToken(token);
}
```

#### 10.5.1.2 修改 service

```
TbUser checkUserToken(String token);
```

### 10.5.1.3 修改 serviceImpl

```
/**
 * 根据用户 token 判断用户在 redis 中是否失效
 * @param token
 * @return
 */
@Override
public TbUser checkUserToken(String token) {

    return (TbUser)
this.redisTemplate.opsForValue().get(this.user_session_redis_key+":"+token);
}
```

## 10.5.2 在 frontend\_cart 服务中添加校验用户是否登录的 Interceptor

### 10.5.2.1 创建 Interceptor

```
/**
 * 在结算之前判断用户是否登录的拦截器
 */
```

@Component

```
public class UserLoginInterceptor implements HandlerInterceptor {
```

@Autowired

```
private UserCheckService userCheckService;
```

@Override

```
public boolean preHandle(HttpServletRequest request, HttpServletResponse  
response, Object handler) throws Exception {
```

```
    //对用户的 token 做判断
```

```
    String token = request.getParameter("token");
```

```
    if(StringUtils.isBlank(token)){
```

```
        return false;
```

```
    }
```

```
    //如果用户 token 不为空，则校验用户在 redis 中是否失效
```

```
    TbUser tbUser = this.userCheckService.checkUserToken(token);
```

```
    if(tbUser == null){
```

```
        return false;
```

```
    }
```

```
    return true;
```

```
}
```

---

```
@Override
```

```
    public void postHandle(HttpServletRequest request, HttpServletResponse  
response, Object handler, ModelAndView modelAndView) throws Exception {  
  
    }  

```

```
@Override
```

```
    public void afterCompletion(HttpServletRequest request,  
HttpServletResponse response, Object handler, Exception ex) throws Exception {  
  
    }  
}
```

### 10.5.2.2 创建配置类，配置拦截器

```
/**
```

```
 * 拦截器配置类
```

```
 */
```

```
@Configuration
```

```
public class WebApplication implements WebMvcConfigurer {
```

@Autowired

**private** UserLoginInterceptor **userLoginInterceptor**;

/\*\*

\* 注册拦截器

\* @param registry

\*/

@Override

**public void** addInterceptors(InterceptorRegistry registry) {

    InterceptorRegistration registration =

registry.addInterceptor(**this.userLoginInterceptor**);

    //拦截那个 URI

    registration.addPathPatterns("/**cart/goSettlement/\*\***");

}

}

### 10.5.2.3 修改 feignClient

@PostMapping("/sso/redis/checkUserToken")

TbUser checkUserToken(@RequestParam("**token**") String token);

---

#### 10.5.2.4 创建校验用户登录业务层接口

```
public interface UserCheckService {  
  
    TbUser checkUserToken(String token);  
  
}
```

#### 10.5.2.5 创建校验用户登录业务层接口实现类

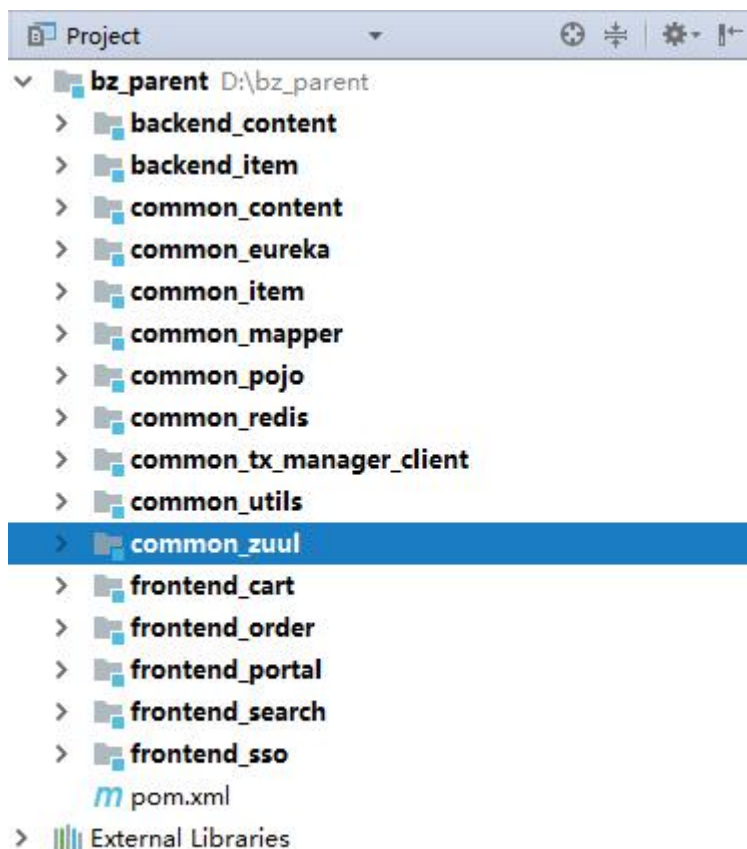
```
/**  
 * 根据用户 token 校验用户是否失效业务层  
 */  
  
@Service  
  
public class UserCheckServiceImpl implements UserCheckService {  
  
    @Autowired  
  
    private CommonRedisFeignClient commonRedisFeignClient;  
  
    @Override  
  
    public TbUser checkUserToken(String token) {  
  
        return this.commonRedisFeignClient.checkUserToken(token);  
    }  
}
```

```
}  
  
}
```

## 七、 网关服务

### 1 创建网关服务

#### 1.1 创建项目



#### 1.2 修改 POM 文件 , 添加相关依赖

```
<?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
http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <parent>

    <artifactId>bz_parent</artifactId>

    <groupId>com.bjsxt</groupId>

    <version>1.0-SNAPSHOT</version>

  </parent>

  <modelVersion>4.0.0</modelVersion>

  <artifactId>common_zuul</artifactId>

  <dependencies>

    <!--Spring Boot Web Starter-->

    <dependency>

      <groupId>org.springframework.boot</groupId>

      <artifactId>spring-boot-starter-web</artifactId>

    </dependency>

    <!--Spring Cloud Eureka Client Starter-->

    <dependency>

      <groupId>org.springframework.cloud</groupId>
```



```
<artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>

</dependency>

<!--Spring Cloud Zuul Starter-->

<dependency>

    <groupId>org.springframework.cloud</groupId>

    <artifactId>spring-cloud-starter-netflix-zuul</artifactId>

</dependency>

</dependencies>

<build>

    <plugins>

        <plugin>

            <groupId>org.springframework.boot</groupId>

            <artifactId>spring-boot-maven-plugin</artifactId>

        </plugin>

    </plugins>

</build>

</project>
```

---

### 1.3 创建配置文件，添加相关配置

```
spring:

  application:

    name: common-zuul

server:

  port: 7070

eureka:

  client:

    serviceUrl:

      defaultZone: http://eureka-server:8761/eureka/
```

### 1.4 创建启动类，添加相关注解

```
/**
 * CommonZuulServer
 */
@SpringBootApplication
@EnableDiscoveryClient
```

---

@EnableZuulProxy

```
public class CommonZuulApplication {  
  
    public static void main(String[] args){  
  
        SpringApplication.run(CommonZuulApplication.class,args);  
  
    }  
  
}
```

## 2 在网关中配置服务代理

### 2.1 配置后台服务代理

#### 2.1.1 修改前台页面代码

##### 2.1.1.1 修改 vue.config.js

```
module.exports = {  
  devServer: {  
    proxy: {  
      '/product_api': {  
        // target: 'http://121.42.14.194:9011',  
        target: 'http://192.168.1.101:7070',  
        pathRewrite: {  
          '^/product_api': ''  
        },  
        changeOrigin: true  
      },  
      '/content_api': {  
        target: 'http://192.168.1.101:7070',  
        // target: 'http://121.42.14.194:9021',  
        pathRewrite: {  
          '^/content_api': ''  
        },  
        changeOrigin: true  
      }  
    }  
  }  
}
```

```

    },
    "/api":{
      target: 'http://localhost:3000',
      pathRewrite: {
        '^/api': ''
      },
      changeOrigin: true
    }
  }
}

```

### 2.1.1.2 修改 base.js

```

const base = {
  baseUrl: "/api",
  basePrductUrl: "/product_api/backend_item",
  baseContentUrl: "/content_api/backend_content",
  login: "/api/login", // 登陆地址
  prodcutList: "/backend/item/selectTbItemAllByPage", // 商品列表地址
  selectItemCategoryByParentId: "/backend/itemCategory/selectItemCategoryByParentId", // 类目选择地址
  insertTbItem: "/backend/item/insertTbItem", // 商品添加地址
  groupParamData: '/backend/itemParam/selectItemParamByItemCatId', // 添加规格参数地址
  deleteItemById: "/backend/item/deleteItemById", // 商品删除地址
  preUpdateItem: "/backend/item/preUpdateItem", // 预更新商品地址
  updateProduct: '/backend/item/updateTbItem', // 修改商品信息地址
  selectItemParamAll: "/backend/itemParam/selectItemParamAll", // 规格参数查询地址
  insertItemParam: '/backend/itemParam/insertItemParam', // 规格参数添加地址
  deleteItemParamById: "/backend/itemParam/deleteItemParamById", // 规格参数删除地址
  selectContentCategoryByParentId: "/content/selectContentCategoryByParentId", // 分类内容管理 查询地址
  insertContentCategory: "/content/insertContentCategory", // 内容分类管理 添加接口
  deleteContentCategoryById: "/content/deleteContentCategoryById", // 内容分类管理 删除接口
  updateContentCategory: "/content/updateContentCategory", // 内容

```

```

分类管理 修改接口
    selectTbContentAllByCategoryId:"/content/selectTbContentAllByCategoryId", // 内容管理 查询接口
    insertTbContent:"/content/insertTbContent", // 内容管理 添加接口
    deleteContentByIds:"/content/deleteContentByIds" // 内容管理 删除接口
}

export default base;

```

## 2.1.2修改网关服务配置文件

```

spring:
  application:
    name: common-zuul

server:
  port: 7070

eureka:
  client:
    serviceUrl:
      defaultZone: http://eureka-server:8761/eureka/

zuul:
  sensitive-headers: true #全局配置，解决在网关服务中不传递请求头的问题(Cookie)
  routes:

    #后台商品服务路由规则
    backend-item:
      path: /backend_item/**

    #后台 CMS 服务的路由规则
    backend-content:
      path: /backend_content/**

```

## 3 配置前台服务代理

### 3.1 修改前台页面

#### 3.1.1 修改 vue.config.js

```
module.exports = {
  // publicPath: process.env.NODE_ENV === 'production'
  //   ? ''
  //   : '/',
  devServer: {
    port: 8080,
    proxy: {
      '/api': {
        //target: 'http://192.168.1.101:9030', //首页服
        务 frontend_portal
        target: 'http://192.168.1.101:7070', //首页服
        务 frontend_portal
        pathRewrite: {
          '^/api': ''
        },
        changeOrigin: true
      },
      '/search_api': {
        //target: 'http://192.168.1.101:9100', //搜索服
        务 frontend_serach
        target: 'http://192.168.1.101:7070', //搜索服
        务 frontend_serach
        pathRewrite: {
          '^/search_api': ''
        },
        changeOrigin: true
      },
      '/shopcar_api': {
        //target: 'http://192.168.1.101:9040', //购物车服
        务 frontend_cart
        target: 'http://192.168.1.101:7070', //购物车服
        务 frontend_cart
        pathRewrite: {
          '^/shopcar_api': ''
        },
        changeOrigin: true
      },
    },
  },
}
```

```

        "/payment_api": {
            // target: 'http://192.168.1.101:9111', //订单服
            务 frontend_order
            target: 'http://192.168.1.101:7070', //订单服
            务 frontend_order
            pathRewrite: {
                '^/payment_api': ''
            },
            changeOrigin: true
        },
        "/sxt": {
            target: 'http://192.168.1.116:3001',
            pathRewrite: {
                '^/sxt': ''
            },
            changeOrigin: true
        },
        "/register_api": {
            //target: 'http://192.168.1.101:9090', //用户注册与登
            录服务 frontend_sso
            target: 'http://192.168.1.101:7070', //用户注册与登录
            服务 frontend_sso
            pathRewrite: {
                '^/register_api': ''
            },
            changeOrigin: true
        }
    }
}
}
}

```

### 3.1.2修改 base.js

```

const base = {
    baseUrl: '/api',
    sxtBaseUrl: '/sxt',
    shopcarBaseUrl: "/shopcar_api", //首页服务
    searchBaseUrl: "/search_api", //搜索服务
    registerBaseUrl: "/register_api", //用户登录与注册服务
    payMentBaseUrl: "/payment_api", //订单服务
    menuItem: "/frontend_potat/frontend/itemCategory/selectItemCate
oryAll", // 左侧菜单

```

```

    search: "/frontend_search/search/list",
    // 搜索接口
    selectFrontendContentByAD: "/frontend_potat/frontend/content/selectFrontendContentByAD", // 首页 焦点轮播图接口
    register: '/frontend_sso/sso/userRegister',
    // 注册接口
    checkRegister: "/frontend_sso/sso/checkUserInfo",
    // 注册验证
    login: "/frontend_sso/sso/userLogin",
    // 登陆接口
    logOut: "/frontend_sso/sso/logOut",
    // 退出登陆
    selectItemInfo: "/frontend_potat/frontend/item/selectItemInfo",
    // 商品详情页 详情接口
    selectItemDescById: "/frontend_potat/frontend/item/selectItemDescById", // 商品详情页 商品介绍
    selectTbItemParamItemById: "/frontend_potat/frontend/item/selectTbItemParamItemById", // 商品详情页 规格参数
    addItem: "/frontend_cart/cart/addItem/",
    // 加入购物车
    showCart: "/frontend_cart/cart/showCart",
    // 购物车列表
    updateItemNum: "/frontend_cart/cart/updateItemNum",
    // 购物车数量变化
    deleteItemFromCart: "/frontend_cart/cart/deleteItemFromCart",
    // 购物车删除接口
    getItemList: "/frontend_cart/cart/goSettlement",
    // 去结算地址
    insertOrder: "/frontend_order/order/insertOrder"
    // 提交订单
  }

export default base;

```

### 3.2 修改网关服务配置文件

```

spring:
  application:
    name: common-zuul

server:
  port: 7070

```



```
eureka:
  client:
    serviceUrl:
      defaultZone: http://eureka-server:8761/eureka/
zuul:
  sensitive-headers: true #全局配置，解决在网关服务中不传递请求头的问题
                           (Cookie)
  routes:

    #后台商品服务路由规则
    backend-item:
      path: /backend_item/**

    #后台 CMS 服务的路由规则
    backend-content:
      path: /backend_content/**

    #前台首页服务的路由规则
    frontend-portal:
      path: /frontend_potal/**

    #前台搜索服务的路由规则
    frontend-search:
      path: /frontend_search/**

    #前台用户注册与登录服务
    frontend-sso:
      path: /frontend_sso/**

    #前台订单服务
    frontend-order:
      path: /frontend_order/**

    #前台购物车服务
    frontend-cart:
      path: /frontend_cart/**
```

## 4 在网关服务中配置超时时间

### 4.1 修改配置文件

```
#配置网关请求服务的超时时间
#第一层 hystrix 的超时时间
hystrix:
  command:
    default:
      execution:
        isolation:
          thread:
            timeoutInMilliseconds: 5000    #默认为线程池隔离，默认超时
时间 为 1000ms

#第二层 Ribbon 的超时时间设置
ribbon:
  ConnectTimeout: 3000 #设置请求连接的超时时间 默认为 5 秒
  ReadTimeout: 3000 #设置请求处理超时时间 默认为 5 秒
```

## 5 在网关中实现对服务降级处理

### 5.1 创建返回托底数据对象

#### 5.1.1 backend-item 服务

```
@Component
public class BackendItemFallback implements FallbackProvider {
    @Override
    public String getRoute() {
        return "backend-item";
    }

    @Override
    public ClientHttpResponse fallbackResponse(String route,
        Throwable cause) {
        if (cause != null && cause.getCause() != null) {
            String c = cause.getCause().getMessage();
            System.out.println(route+"\t"+c);
        }
    }
}
```

```
}
return new ClientHttpResponse() {
    @Override
    public HttpStatus getStatusCode() throws IOException {
        return HttpStatus.OK;
    }

    @Override
    public int getRawStatusCode() throws IOException {
        return this.getStatusCode().value();
    }

    @Override
    public String getStatusText() throws IOException {
        return this.getStatusCode().getReasonPhrase();
    }

    @Override
    public void close() {

    }

    @Override
    public InputStream getBody() throws IOException {
        String content = "服务超时，请重试";
        return new ByteArrayInputStream(content.getBytes());
    }

    @Override
    public HttpHeaders getHeaders() {
        HttpHeaders headers = new HttpHeaders();
        MediaType mediaType = new
MediaType("application", "json", Charset.forName("utf-8"));
        headers.setContentType(mediaType);
        return headers;
    }
};
}
```

## 5.1.2 backend-content 服务

```
@Component
public class BackendContentFallback implements FallbackProvider {
    @Override
    public String getRoute() {
        return "backend-content";
    }

    @Override
    public ClientHttpResponse fallbackResponse(String route,
        Throwable cause) {
        if (cause != null && cause.getCause() != null) {
            String c = cause.getCause().getMessage();
            System.out.println(route+"\t"+c);
        }
        return new ClientHttpResponse() {
            @Override
            public HttpStatus getStatusCode() throws IOException {
                return HttpStatus.OK;
            }

            @Override
            public int getRawStatusCode() throws IOException {
                return this.getStatusCode().value();
            }

            @Override
            public String getStatusText() throws IOException {
                return this.getStatusCode().getReasonPhrase();
            }

            @Override
            public void close() {

            }

            @Override
            public InputStream getBody() throws IOException {
                String content = "服务超时，请重试";
                return new ByteArrayInputStream(content.getBytes());
            }

            @Override
```

```

        public HttpHeaders getHeaders() {
            HttpHeaders headers = new HttpHeaders();
            MediaType mediaType = new
MediaType("application", "json", Charset.forName("utf-8"));
            headers.setContentType(mediaType);
            return headers;
        }
    };
}
}

```

### 5.1.3frontend-portal 服务

```

/**
 * 对 frontend_portal 服务降级处理，返回托底数据
 */
@Component
public class FrontendPortalFallback implements FallbackProvider {
    @Override
    public String getRoute() {
        return "frontend-portal";
    }

    @Override
    public ClientHttpResponse fallbackResponse(String route,
        Throwable cause) {
        if(cause != null && cause.getCause() != null){
            String c = cause.getCause().getMessage();
            System.out.println(route+"\t"+c);
        }
        return new ClientHttpResponse() {
            @Override
            public HttpStatus getStatusCode() throws IOException {
                return HttpStatus.OK;
            }

            @Override
            public int getRawStatusCode() throws IOException {
                return this.getStatusCode().value();
            }

            @Override

```

```

        public String getStatusText() throws IOException {
            return this.getStatusCode().getReasonPhrase();
        }

        @Override
        public void close() {

        }

        @Override
        public InputStream getBody() throws IOException {
            String content = "首页服务超时，请重试";
            return new ByteArrayInputStream(content.getBytes());
        }

        @Override
        public HttpHeaders getHeaders() {
            HttpHeaders headers = new HttpHeaders();
            MediaType mediaType = new
MediaType("application", "json", Charset.forName("utf-8"));
            headers.setContentType(mediaType);
            return headers;
        }
    };
}
}

```

### 5.1.4frontend-search 服务

```

@Component
public class FrontendSearchFallback implements FallbackProvider{
    @Override
    public String getRoute() {
        return "frontend-search";
    }

    @Override
    public ClientHttpResponse fallbackResponse(String route,
        Throwable cause) {
        if(cause != null && cause.getCause() != null){
            String c = cause.getCause().getMessage();
            System.out.println(route+"\t"+c);
        }
    }
}

```

```
}
return new ClientHttpResponse() {
    @Override
    public HttpStatus getStatusCode() throws IOException {
        return HttpStatus.OK;
    }

    @Override
    public int getRawStatusCode() throws IOException {
        return this.getStatusCode().value();
    }

    @Override
    public String getStatusText() throws IOException {
        return this.getStatusCode().getReasonPhrase();
    }

    @Override
    public void close() {

    }

    @Override
    public InputStream getBody() throws IOException {
        String content = "首页服务超时，请重试";
        return new ByteArrayInputStream(content.getBytes());
    }

    @Override
    public HttpHeaders getHeaders() {
        HttpHeaders headers = new HttpHeaders();
        MediaType mediaType = new
MediaType("application", "json", Charset.forName("utf-8"));
        headers.setContentType(mediaType);
        return headers;
    }
};
}
```

## 5.1.5frontend-sso 服务

```
@Component
public class FrontendSsoFallback implements FallbackProvider {
    @Override
    public String getRoute() {
        return "frontend-sso";
    }

    @Override
    public ClientHttpResponse fallbackResponse(String route,
        Throwable cause) {
        if (cause != null && cause.getCause() != null) {
            String c = cause.getCause().getMessage();
            System.out.println(route + "\t" + c);
        }
        return new ClientHttpResponse() {
            @Override
            public HttpStatus getStatusCode() throws IOException {
                return HttpStatus.OK;
            }

            @Override
            public int getRawStatusCode() throws IOException {
                return this.getStatusCode().value();
            }

            @Override
            public String getStatusText() throws IOException {
                return this.getStatusCode().getReasonPhrase();
            }

            @Override
            public void close() {

            }

            @Override
            public InputStream getBody() throws IOException {
                String content = "首页服务超时，请重试";
                return new ByteArrayInputStream(content.getBytes());
            }

            @Override
```



```
        public HttpHeaders getHeaders() {
            HttpHeaders headers = new HttpHeaders();
            MediaType mediaType = new MediaType("application",
"json", Charset.forName("utf-8"));
            headers.setContentType(mediaType);
            return headers;
        }
    };
}
}
```

### 5.1.6frontend-order 服务

```
@Component
public class FrontendOrderFallback implements FallbackProvider {
    @Override
    public String getRoute() {
        return "frontend-order";
    }

    @Override
    public ClientHttpResponse fallbackResponse(String route,
    Throwable cause) {
        if(cause != null && cause.getCause() != null){
            String c = cause.getCause().getMessage();
            System.out.println(route+"\t"+c);
        }
        return new ClientHttpResponse() {
            @Override
            public HttpStatus getStatusCode() throws IOException {
                return HttpStatus.OK;
            }

            @Override
            public int getRawStatusCode() throws IOException {
                return this.getStatusCode().value();
            }

            @Override
            public String getStatusText() throws IOException {
                return this.getStatusCode().getReasonPhrase();
            }
        }
    }
}
```

```

        @Override
        public void close() {

        }

        @Override
        public InputStream getBody() throws IOException {
            String content = "服务超时, 请重试";
            return new ByteArrayInputStream(content.getBytes());
        }

        @Override
        public HttpHeaders getHeaders() {
            HttpHeaders headers = new HttpHeaders();
            MediaType mediaType = new
MediaType("application", "json", Charset.forName("utf-8"));
            headers.setContentType(mediaType);
            return headers;
        }
    };
}
}

```

### 5.1.7 frontend-cart 服务

```

@Component
public class FrontendCartFallback implements FallbackProvider {
    @Override
    public String getRoute() {
        return "frontend-cart";
    }

    @Override
    public ClientHttpResponse fallbackResponse(String route,
        Throwable cause) {
        if (cause != null && cause.getCause() != null) {
            String c = cause.getCause().getMessage();
            System.out.println(route + "\t" + c);
        }
        return new ClientHttpResponse() {
            @Override

```

```
public HttpStatus getStatusCode() throws IOException {
    return HttpStatus.OK;
}

@Override
public int getRawStatusCode() throws IOException {
    return this.getStatusCode().value();
}

@Override
public String getStatusText() throws IOException {
    return this.getStatusCode().getReasonPhrase();
}

@Override
public void close() {

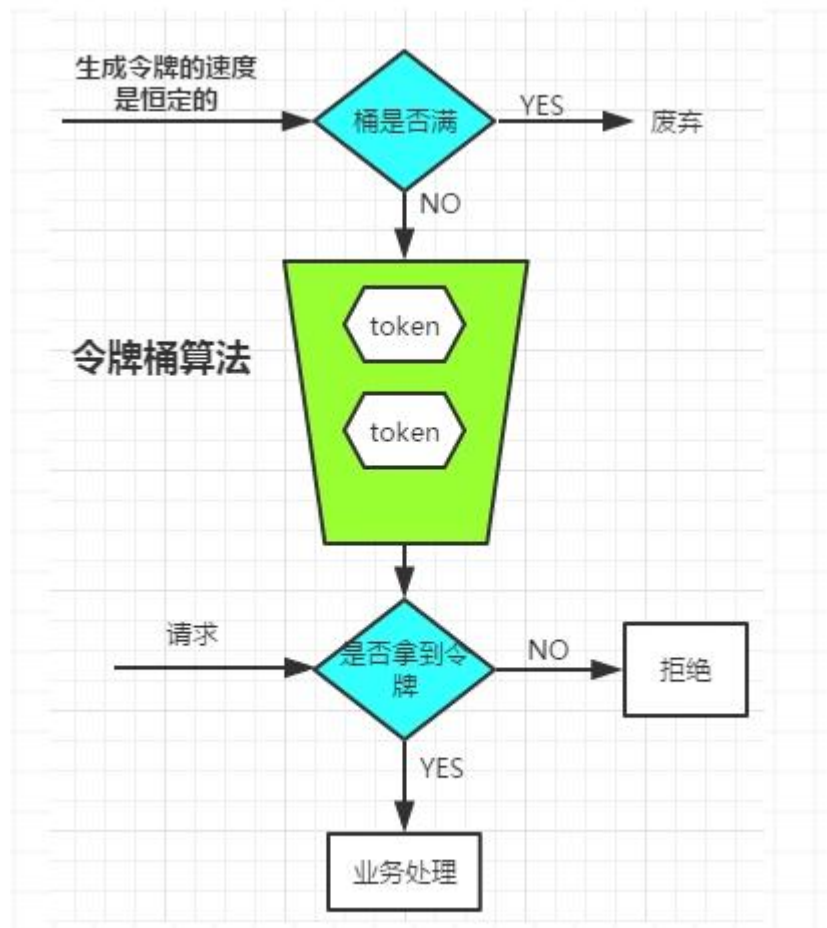
}

@Override
public InputStream getBody() throws IOException {
    String content = "服务超时，请重试";
    return new ByteArrayInputStream(content.getBytes());
}

@Override
public HttpHeaders getHeaders() {
    HttpHeaders headers = new HttpHeaders();
    MediaType mediaType = new
MediaType("application", "json", Charset.forName("utf-8"));
    headers.setContentType(mediaType);
    return headers;
}
};
}
}
```

## 6 使用令牌桶算法实现限流

### 6.1 令牌桶原理



### 6.2 创建限流 Filter

```
/**
 * 限流器
 */
@Component
public class RateLimitFilter extends ZuulFilter {

    //创建令牌桶
    //RateLimiter.create(1) 1:是每秒生成令牌的数量
    //数值越大代表处理请求量月多，数值越小代表处理请求量越少
    private static final RateLimiter RATE_LIMIT =
RateLimiter.create(1);
```

```
@Override
public String filterType() {
    return FilterConstants.PRE_TYPE;
}

/**
 * 限流器的优先级应为最高
 * @return
 */
@Override
public int filterOrder() {
    return FilterConstants.SERVLET_DETECTION_FILTER_ORDER - 1;
}

@Override
public boolean shouldFilter() {
    return true;
}

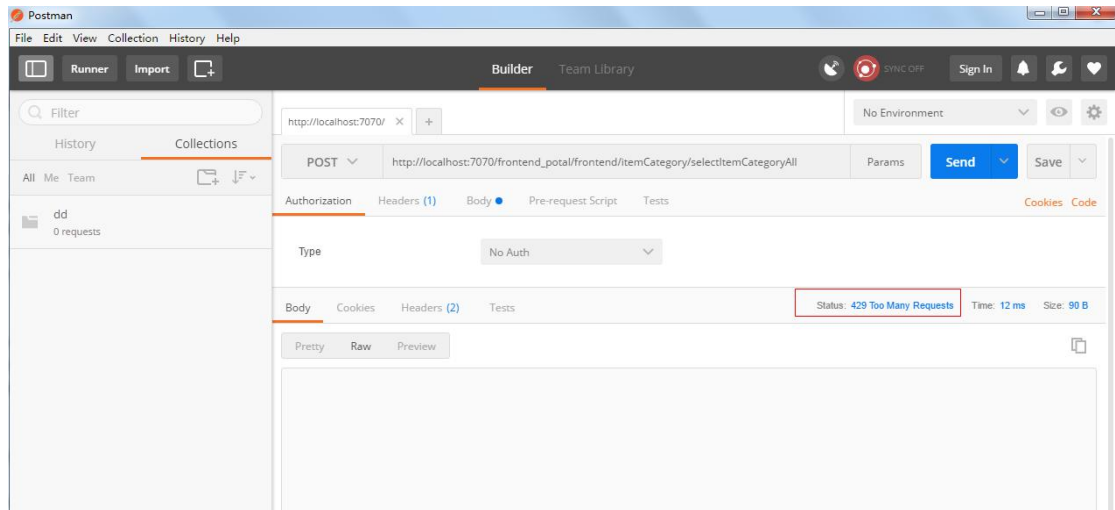
@Override
public Object run() throws ZuulException {

    //是否能从令牌桶中获取到令牌
    if(!RATE_LIMIT.tryAcquire()){
        RequestContext requestContext =
RequestContext.getCurrentContext();
        requestContext.setSendZuulResponse(false);
        requestContext.setResponseStatusCode(429);
        System.out.println("限流");
    }

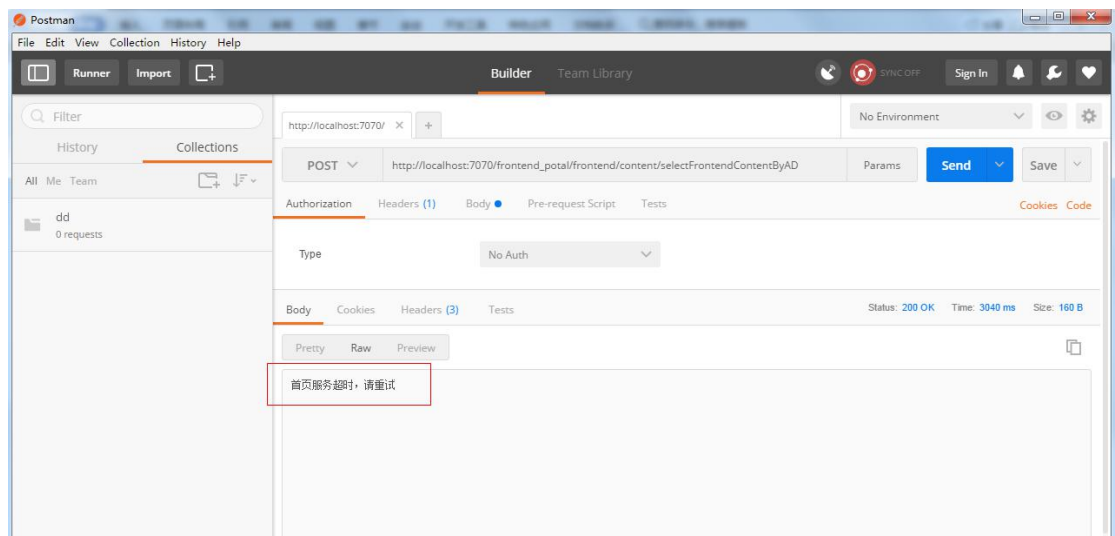
    return null;
}
}
```

## 7 使用 Postman 测试服务降级与网关限流

### 7.1 限流



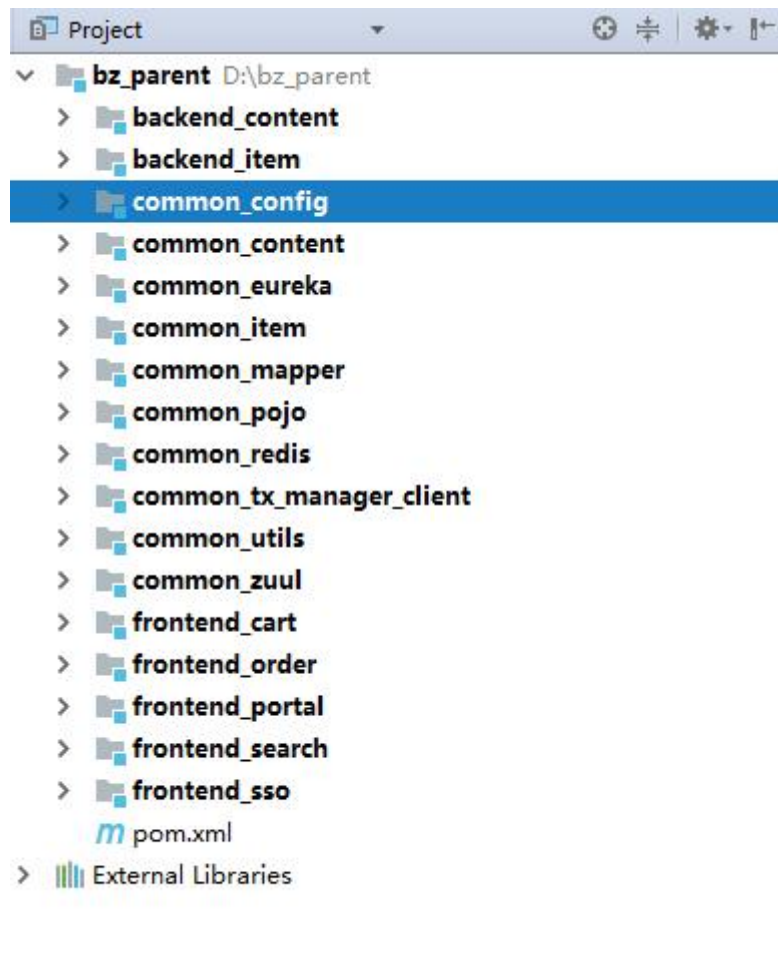
### 7.2 查看托底数据



## 八、 分布式配置中心

### 1 创建分布式配置中心服务端

#### 1.1 创建项目



#### 1.2 修改 POM 文件，添加相关依赖

```
<?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
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <parent>
    <artifactId>bz_parent</artifactId>
    <groupId>com.bjsxt</groupId>
    <version>1.0-SNAPSHOT</version>
  </parent>
```

```
<modelVersion>4.0.0</modelVersion>

<artifactId>common_config</artifactId>

<dependencies>

    <!--Spring Boot Web Starter-->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <!--Spring Cloud Eureka Client Starter-->
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
    </dependency>

    <!--Spring Cloud Config Server-->
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-config-server</artifactId>
    </dependency>

</dependencies>
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
</project>
```

### 1.3 修改配置文件，添加相关配置

```
spring:
  application:
    name: common-config
```



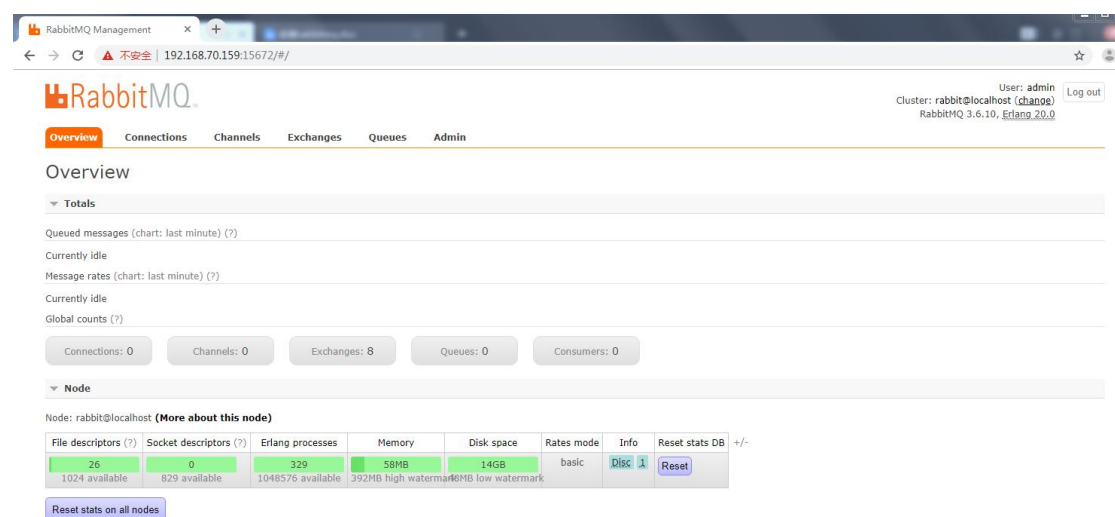
```
server:
  port: 9000

eureka:
  client:
    serviceUrl:
      defaultZone: http://eureka-server:8761/eureka/
```

## 1.4 创建启动类，添加相关注解

```
/**
 * 分布式配置中心服务端
 */
@SpringBootApplication
@EnableDiscoveryClient
@EnableConfigServer
public class CommonConfigApplication {
    public static void main(String[] args) {
        SpringApplication.run(CommonConfigApplication.class, args);
    }
}
```

## 2 安装 RabbitMQ



The screenshot shows the RabbitMQ Management interface in a web browser. The page title is "RabbitMQ Management" and the URL is "192.168.70.159:15672/#/". The user is logged in as "admin". The page has a navigation bar with tabs: Overview, Connections, Channels, Exchanges, Queues, and Admin. The "Overview" tab is selected. The "Overview" section shows "Totals" for the cluster: "Queued messages (chart: last minute) (?)", "Currently idle", "Message rates (chart: last minute) (?)", "Currently idle", and "Global counts (?)". Below these are buttons for "Connections: 0", "Channels: 0", "Exchanges: 8", "Queues: 0", and "Consumers: 0". The "Node" section shows details for the node "rabbit@localhost (More about this node)". It includes a table with metrics: File descriptors (26), Socket descriptors (0), Erlang processes (329), Memory (59MB), Disk space (14GB), Rates mode (basic), and Info (Disk, Reset). The table also shows available and watermark values for each metric. There are buttons for "Reset stats on all nodes" and "Reset stats DB".

## 3 在 gitee 中创建远程仓库

### 3.1 登录 gitee



### 3.2 新建仓库

#### 新建仓库

仓库名称 ☒ 仓库名称

shop\_config

归属 路径

kevin / shop\_config

仓库地址: [https://gitee.com/oldlu\\_wk/shop\\_config](https://gitee.com/oldlu_wk/shop_config)

仓库介绍 非必填

百战商城配置库

是否开源

☐ 私有 ☒ 公开

任何人都可以访问该仓库的代码和其他任何形式的资源

选择语言 添加 .gitignore 添加开源许可证

请选择语言 请选择 .gitignore 模板 请选择开源许可证

☒ 使用Readme文件初始化这个仓库

☐ 使用Issue模板文件初始化这个仓库

☐ 使用Pull Request模板文件初始化这个仓库

选择分支模型 (仓库初始化后将根据所选分支模型创建分支)

### 3.3 拷贝远程仓库地址

[https://gitee.com/oldlu\\_wk/shop\\_config.git](https://gitee.com/oldlu_wk/shop_config.git)

## 4 在分布式配置中心服务端中添加消息总线

### 4.1 修改 POM 文件，添加相关依赖

```
<?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
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <parent>
        <artifactId>bz_parent</artifactId>
        <groupId>com.bjsxt</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>common_config</artifactId>

    <dependencies>

        <!--Spring Boot Web Starter-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <!--Spring Cloud Eureka Client Starter-->
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
        </dependency>

        <!--Spring Cloud Config Server-->
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-config-server</artifactId>
```

```
</dependency>

<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-bus-amqp</artifactId>
</dependency>

</dependencies>
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
</project>
```

## 4.2 修改配置文件，添加相关配置

```
spring:
  application:
    name: common-config
  cloud:
    config:
      server:
        git:
          uri: https://gitee.com/oldlu_wk/shop_config.git
    bus:
      refresh:
        enabled: true #开启自动刷新
  rabbitmq:
    addresses: 192.168.70.159
    username: admin
    password: 123456
    port: 5672
    virtual-host: /

server:
  port: 9000
```

```
eureka:
  client:
    serviceUrl:
      defaultZone: http://eureka-server:8761/eureka/

management:
  endpoints:
    web:
      exposure:
        include: bus-refresh #在 Greewitch.SR2 版中需要开启
```

## 5 在分布式配置中心客户端中添加消息总线

### 5.1 修改每个客户端服务的 POM 文件添加坐标

```
<!--Spring Cloud Config Client Starter-->
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-config</artifactId>
</dependency>

<!--Spring Cloud AMQP: (RabbitMQ) Starter-->
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-bus-amqp</artifactId>
</dependency>
```

### 5.2 在每个客户端服务的配置文件中添加如下配置

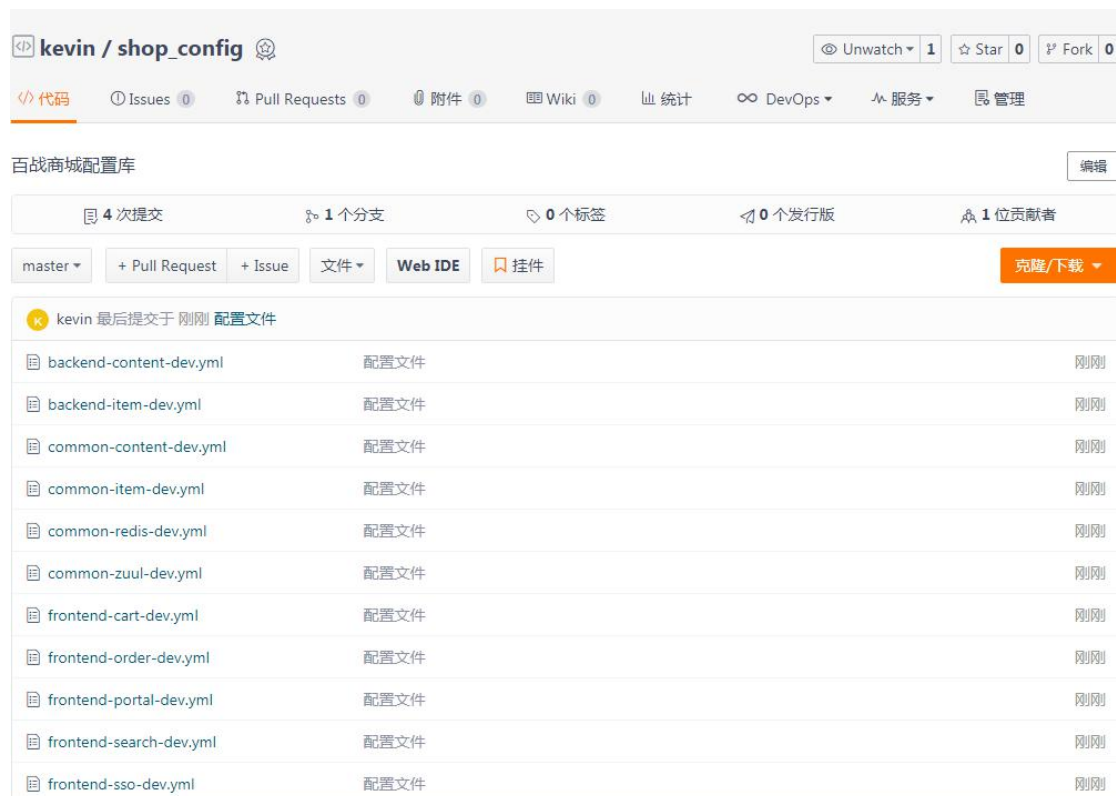
```
rabbitmq:
  addresses: 192.168.70.159
  username: admin
  password: 123456
  port: 5672
  virtual-host: /
```

## 6 将配置文件上传到 gitee 远程仓库中

### 6.1 目标分支

master

### 6.2 上传配置文件



## 7 在每个客户端服务中创建 bootstrap.yml 文件

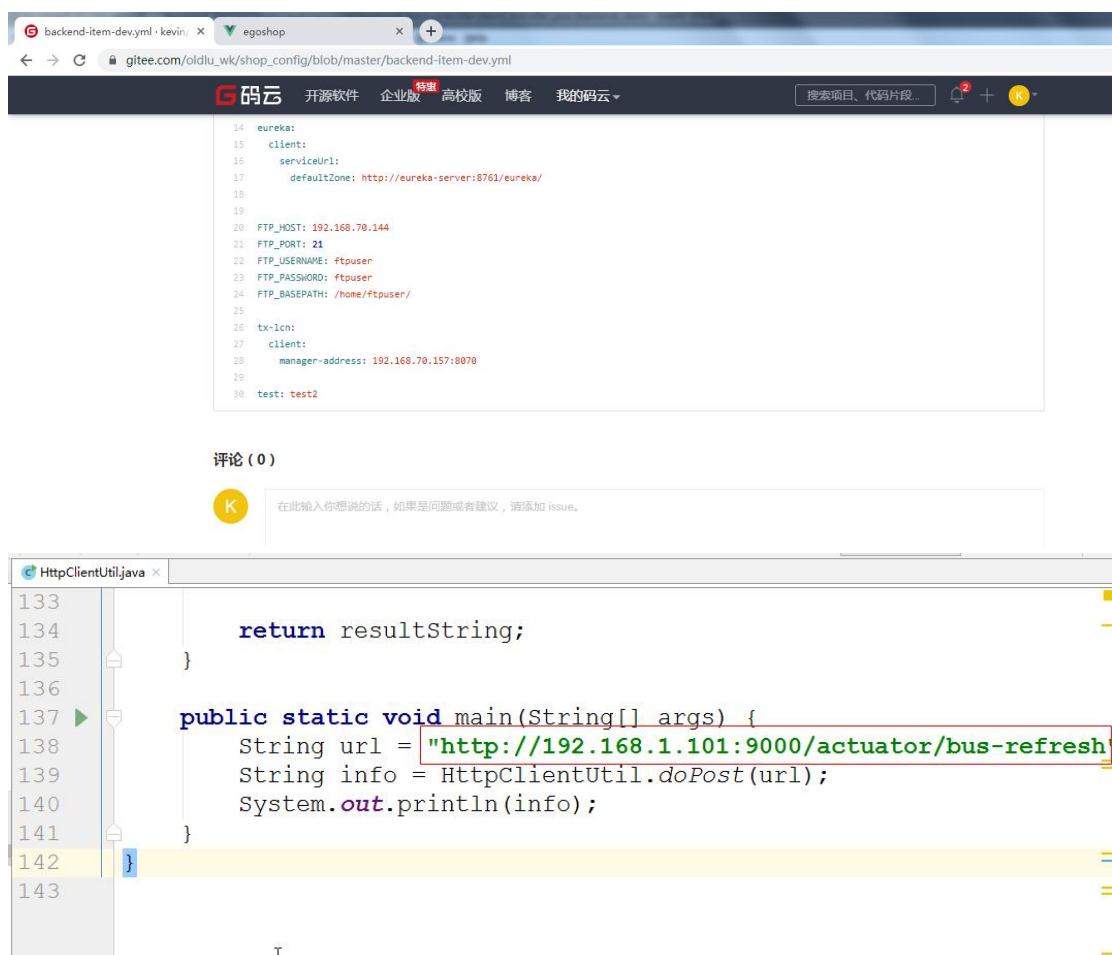
```
spring:
  cloud:
    config:
      discovery:
        enabled:
          server: true
          serviceId: common-config #指定配置中心服务端的服务名称
      name: backend-item #对应的{application}部分
      profile: dev #对应的{profile}部分
      uri: http://192.168.1.101:9000 #配置中心的具体地址，Greenwich 版中需要添加
      label: master
```

## 8 测试分布式配置中心



商品ID	商品标题	商品图片	商品卖点	商品价格	商品类目	商品库存	条形码	状态	创建日期	更新日期	操作
15666343...	全新Kindl...	http://19...	【8.26零...	99800	3	99		1	2019-08-...	2019-08-...	 
15674744...	荣耀10青...	http://19...	限时优惠9...	99900	560	99		1	2019-09-...	2019-09-...	 

## 9 测试自动刷新配置信息



```
14 eureka:
15   client:
16     serviceUrl:
17       defaultZone: http://eureka-server:8761/eureka/
18
19
20 FTP_HOST: 192.168.70.144
21 FTP_PORT: 21
22 FTP_USERNAME: ftpuser
23 FTP_PASSWORD: ftpuser
24 FTP_BASEPATH: /home/ftpuser/
25
26 tx-len:
27   client:
28     manager-address: 192.168.70.157:8070
29
30 test: test2
```

评论 (0)

在此输入你想说的话，如果是问题或者建议，请添加 issue。

```
133
134     return resultString;
135 }
136
137 public static void main(String[] args) {
138     String url = "http://192.168.1.101:9000/actuator/bus-refresh";
139     String info = HttpClientUtil.doPost(url);
140     System.out.println(info);
141 }
142 }
143
```

## 九、 通过 Hystrix 对下游服务做降级处理

### 1 为后台商品管理的下游服务做降级处理

#### 1.1 创建返回托底数据对象

```
/**
 * Common-item 服务返回托底数据
 */
@Component
public class CommonItemFeignClientFallbackFactory implements
FallbackFactory<CommonItemFeignClient>{
    @Override
    public CommonItemFeignClient create(Throwable throwable) {
        return new CommonItemFeignClient() {
            @Override
            public PageResult selectTbItemAllByPage(Integer page,
Integer rows) {
                return null;
            }

            @Override
            public Integer insertTbItem(TbItem tbItem) {
                return null;
            }

            @Override
            public Integer deleteItemById(TbItem tbItem) {
                return null;
            }

            @Override
            public Map<String, Object> preUpdateItem(Long itemId) {
                return null;
            }

            @Override
            public Integer updateTbitem(TbItem tbItem) {
                return null;
            }

            @Override
            public List<TbItemCat> selectItemCategoryByParentId(Long
```



```
id) {  
    return null;  
}  
  
@Override  
public TbItemParam selectItemParamByItemCatId(Long  
itemCatId) {  
    return null;  
}  
  
@Override  
public PageResult selectItemParamAll(Integer page, Integer  
rows) {  
    return null;  
}  
  
@Override  
public Integer insertItemParam(TbItemParam tbItemParam) {  
    return null;  
}  
  
@Override  
public Integer deleteItemParamById(Long id) {  
    return null;  
}  
  
@Override  
public Integer insertItemDesc(TbItemDesc tbItemDesc) {  
    return null;  
}  
  
@Override  
public Integer updateItemDesc(TbItemDesc tbItemDesc) {  
    return null;  
}  
  
@Override  
public Integer insertTbItemParamItem(TbItemParamItem  
tbItemParamItem) {  
    return null;  
}  
  
@Override  
public Integer upateItemParamItem(TbItemParamItem
```

```
tbItemParamItem) {  
    return null;  
}  
};  
}  
}
```

## 1.2 修改 feignClient

```
@FeignClient(value = "common-item", fallbackFactory =  
CommonItemFeignClientFallbackFactory.class)  
public interface CommonItemFeignClient {  
  
    //-----/Service/Item  
    @GetMapping("/service/item/selectTbItemAllByPage")  
    PageResult selectTbItemAllByPage(@RequestParam("page") Integer  
page, @RequestParam("rows") Integer rows);  
  
    @PostMapping("/service/item/insertTbItem")  
    Integer insertTbItem(@RequestBody TbItem tbItem);  
  
    @PostMapping("/service/item/deleteItemById")  
    Integer deleteItemById(@RequestBody TbItem tbItem);  
  
    @PostMapping("/service/item/preUpdateItem")  
    Map<String, Object> preUpdateItem(@RequestParam("itemId") Long  
itemId);  
  
    @PostMapping("/service/item/upateTbitem")  
    Integer updateTbitem(@RequestBody TbItem tbItem);  
  
    //-----/Service/itemCategory  
  
    @PostMapping("/service/itemCategory/selectItemCategoryByParentId"  
)  
    List<TbItemCat>  
selectItemCategoryByParentId(@RequestParam("id") Long id);  
  
    //-----/Service/itemParam-----  
    @PostMapping("/service/itemParam/selectItemParamByItemCatId")  
    TbItemParam
```

```

selectItemParamByItemCatId(@RequestParam("itemCatId") Long
itemCatId);

@PostMapping("/service/itemParam/selectItemParamAll")
PageResult selectItemParamAll(@RequestParam("page") Integer
page, @RequestParam("rows") Integer rows);

@RequestMapping("/service/itemParam/insertItemParam")
Integer insertItemParam(@RequestBody TbItemParam tbItemParam);

@RequestMapping("/service/itemParam/deleteItemParamById")
Integer deleteItemParamById(@RequestParam("id") Long id);

//-----/Service/itemDesc
@PostMapping("/service/itemDesc/insertItemDesc")
Integer insertItemDesc(@RequestBody TbItemDesc tbItemDesc);

@PostMapping("/service/itemDesc/updateItemDesc")
Integer updateItemDesc(@RequestBody TbItemDesc tbItemDesc);

//-----/Service/itemParamItem
@PostMapping("/service/itemParamItem/insertTbItemParamItem")
Integer insertTbItemParamItem(@RequestBody TbItemParamItem
tbItemParamItem);

@PostMapping("/service/itemParamItem/upateItemParamItem")
Integer upateItemParamItem(@RequestBody TbItemParamItem
tbItemParamItem);

}

```

### 1.3 修改 serviceImpl

```

/**
 * 商品管理
 */
@Service
public class ItemServiceImpl implements ItemService {

    @Autowired
    private CommonItemFeignClient commonItemFeignClient;
}

```

```

@Override
public Result selectTbItemAllByPage(Integer page, Integer rows)
{
    PageResult pageResult =
this.commonItemFeignClient.selectTbItemAllByPage(page, rows);
    if(pageResult != null && pageResult.getResult() != null &&
pageResult.getResult().size() > 0){
        return Result.ok(pageResult);
    }
    return Result.error("查无结果");
}

/**
 * 添加 TbItem, 添加 TbitemDesc, 添加 TbItemParamItem
 * @param tbItem
 * @param desc
 * @param itemParams
 * @return
 */
@Override
@LcnTransaction
public Result insertTbItem(TbItem tbItem, String desc, String
itemParams){
    //补齐 Tbitem 数据
    Long itemId = IDUtils.genItemId();
    Date date = new Date();
    tbItem.setId(itemId);
    tbItem.setStatus((byte)1);
    tbItem.setUpdated(date);
    tbItem.setCreated(date);
    Integer tbItemNum =
this.commonItemFeignClient.insertTbItem(tbItem);

    //补齐商品描述对象
    TbItemDesc tbItemDesc = new TbItemDesc();
    tbItemDesc.setItemId(itemId);
    tbItemDesc.setItemDesc(desc);
    tbItemDesc.setCreated(date);
    tbItemDesc.setUpdated(date);
    Integer tbitemDescNum =
this.commonItemFeignClient.insertItemDesc(tbItemDesc);

    //补齐商品规格参数
    TbItemParamItem tbItemParamItem = new TbItemParamItem();

```

```

        tbItemParamItem.setItemId(itemId);
        tbItemParamItem.setParamData(itemParams);
        tbItemParamItem.setUpdated(date);
        tbItemParamItem.setCreated(date);
        Integer itemParamItmeNum =
this.commonItemFeignClient.insertTbItemParamItem(tbItemParamItem)
;

        if(tbItemNum == null || tbitemDescNum== null
||itemParamItmeNum ==null ){
            throw new RuntimeException();
        }
        return Result.ok();
    }

    /**
     * 删除商品
     * @param itemId
     * @return
     */
    @Override
    @LcnTransaction
    public Result deleteItemById(Long itemId){
        TbItem item = new TbItem();
        item.setId(itemId);
        item.setStatus((byte)3);
        Integer itemNum =
this.commonItemFeignClient.deleteItemById(item);
        if(itemNum == null){
            throw new RuntimeException();
        }
        return Result.ok();
    }

    /**
     * 预更新商品查询
     * @param itemId
     * @return
     */
    @Override
    public Result preUpdateItem(Long itemId) {
        Map<String,Object> map =
this.commonItemFeignClient.preUpdateItem(itemId);
        if(map != null) {

```

```

        return Result.ok(map);
    }
    return Result.error("查无结果");
}

/**
 * 更新商品: 更新 TbItem 表, 更新 TbItemDesc 表, 更新 TbItemParamItem
表
 * @param tbItem
 * @param desc
 * @param itemParams
 * @return
 */
@Override
@LcnTransaction
public Result updateTbItem(TbItem tbItem, String desc, String
itemParams) {
    //更新商品
    Integer itemNum =
this.commonItemFeignClient.updateTbitem(tbItem);

    //更新商品描述
    TbItemDesc tbItemDesc = new TbItemDesc();
    tbItemDesc.setItemId(tbItem.getId());
    tbItemDesc.setItemDesc(desc);
    Integer itemDescNum =
this.commonItemFeignClient.updateItemDesc(tbItemDesc);

    //更新商品规格参数
    TbItemParamItem tbItemParamItem = new TbItemParamItem();
    tbItemParamItem.setParamData(itemParams);
    tbItemParamItem.setItemId(tbItem.getId());
    Integer itemParamItemNum =
this.commonItemFeignClient.upateItemParamItem(tbItemParamItem);

    if(itemNum ==null || itemDescNum == null || itemParamItemNum
==null){
        throw new RuntimeException();
    }
    return Result.ok();
}
}

```

## 十、 分布式日志管理

### 1 通过 Logback 向 ELK 中添加日志

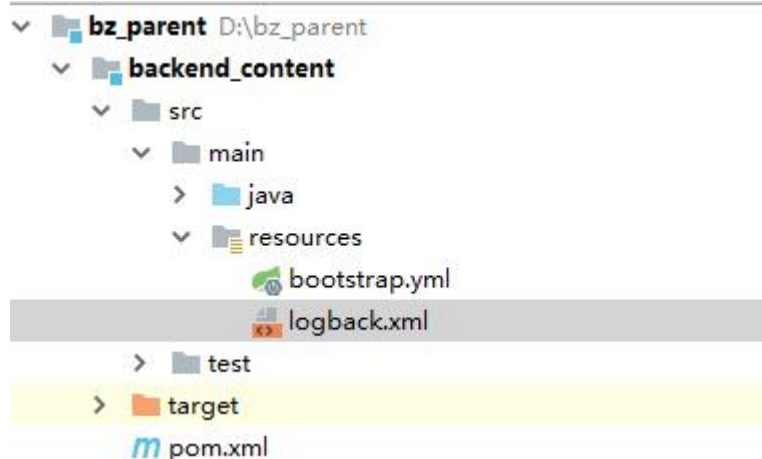
#### 1.1 访问 Kibana

`http://192.168.70.140:5601/app/kibana`

#### 1.2 修改 POM 文件，添加相关依赖

```
<!--Logback-->
<dependency>
  <groupId>net.logstash.logback</groupId>
  <artifactId>logstash-logback-encoder</artifactId>
</dependency>
```

#### 1.3 添加 Logback 配置文件



#### 1.4 修改 Logback 配置文件

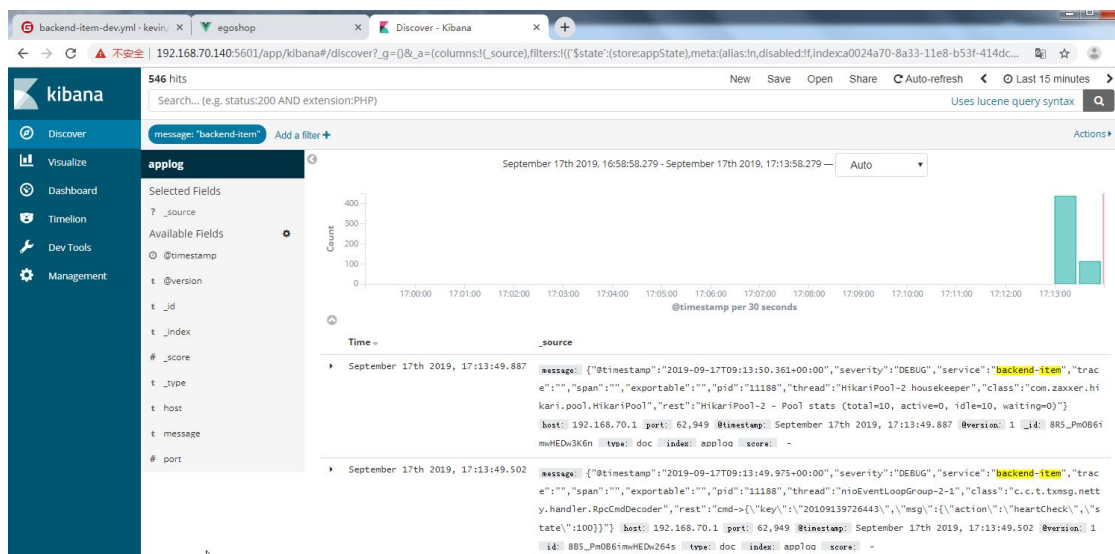
```
<!-- 为logstash 输出的JSON 格式的Appender -->
<appender name="logstash"
  class="net.logstash.logback.appender.LogstashTcpSocketAppender">
  <destination>192.168.70.140:9250</destination>
  <!-- 日志输出编码 -->
  <encoder
```

```

class="net.logstash.logback.encoder.LoggingEventCompositeJsonEncoder">
  <providers>
    <timestamp>
      <timeZone>UTC</timeZone>
    </timestamp>
    <pattern>
      <pattern>
        {
          "severity": "%level",
          "service": "${springAppName:-}",
          "trace": "%X{X-B3-TraceId:-}",
          "span": "%X{X-B3-SpanId:-}",
          "exportable": "%X{X-Span-Export:-}",
          "pid": "${PID:-}",
          "thread": "%thread",
          "class": "%logger{40}",
          "rest": "%message"
        }
      </pattern>
    </pattern>
  </providers>
</encoder>
</appender>

```

## 1.5 通过 Kibana 查询日志





---

## 十一、项目总结

### 1 项目部署

#### 1.1 在项目 POM 文件中添加打包插件

```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
```

## 1.2 打包项目

