

# easymall项目说明

2020年2月11日 9:26

## 1.easymall项目整合

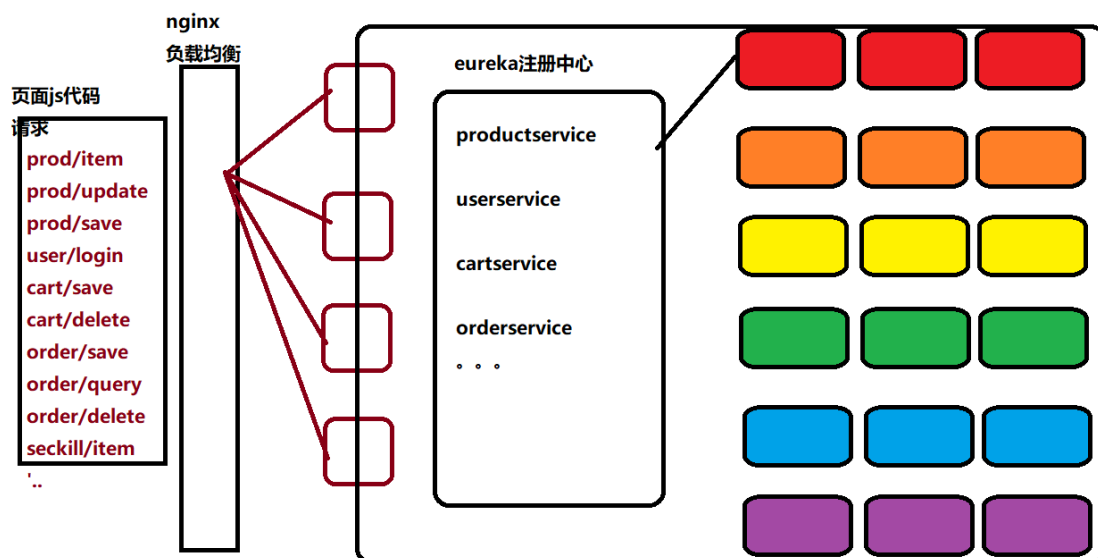
### 1.1介绍

easymall就是web阶段电商项目，具备的功能非常复杂，多，（web阶段只实现用户注册和登陆，没有使用架构技术）。

互联网框架中的easymall功能（微服务）

- 商品系统：处理商品分页查询，单个查询，商品新增，商品修改
- 图片系统：负责整个系统的所有图片的上传逻辑。（浏览交给nginx）
- 用户系统：注册，登录的逻辑交给用户系统，用户身份验证也是用户系统
- 购物车系统：用户的购物车数据
- 订单系统：用户提交删除订单
- 搜索系统：单独运行一个微服务工程，执行easymall中搜索商品功能
- 秒杀系统：单独处理瞬间高并发的秒杀逻辑。

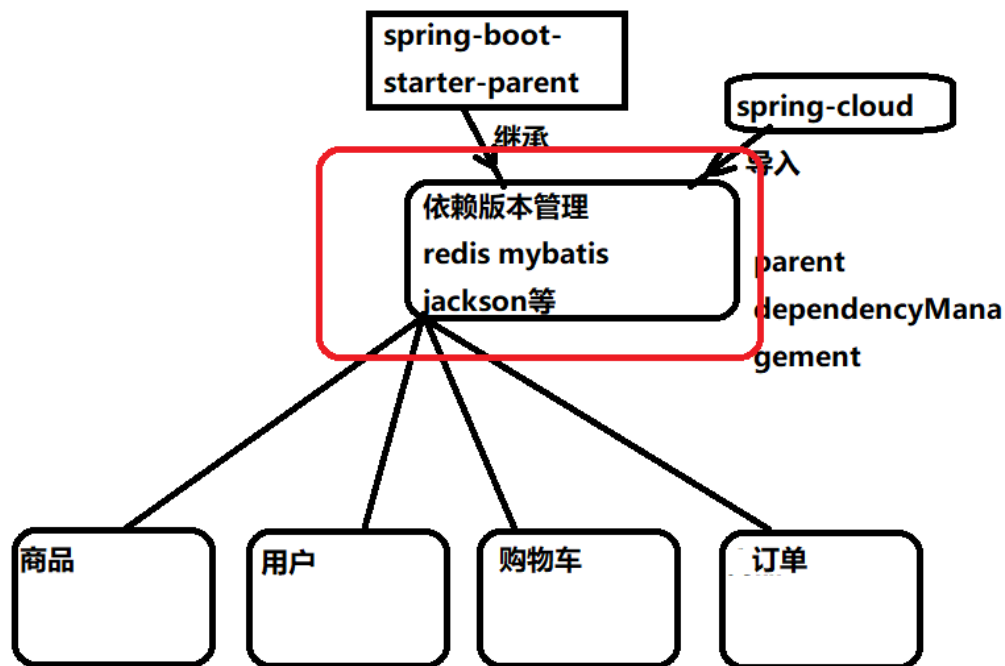
### 1.2springcloud微服务框架结构



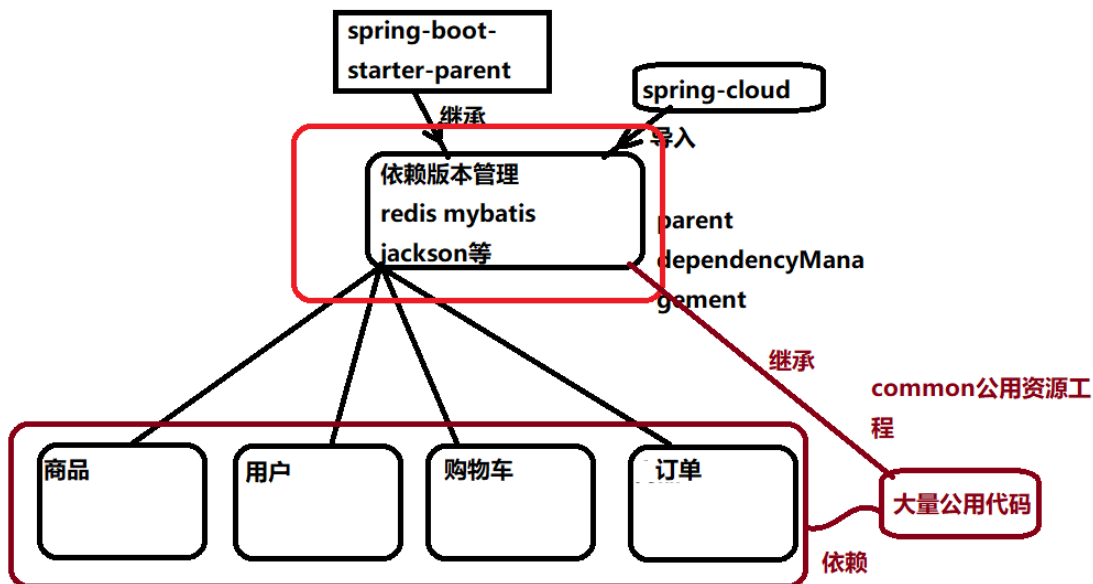
## 2.搭建一下基本的maven项目结构

### 2.1maven管理项目的结构

准备一个父级工程easymall-parent 管理所有easymall中子工程的资源版本（子工程中，看不到版本号的，依赖，依赖管理看不到，插件看不到）



对于每一个微服务中的公用代码可以提取到一个common工程通过依赖实现



## 2.3搭建easymall的管理结构

使用课前资料--06springcloud--07easymall项目--common工程代码粘贴

### 2.1easymall-parent工程

- 创建parent工程
- pom文件：
  - 继承spring-boot-starter-parent

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>1.5.9.RELEASE</version>
</parent>
```

- 导入springcloud
- 管理有版本号的资源 (dependencyManagement)
  - ◆ redis
  - ◆ mybatis
  - ◆ Elasticsearch

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-dependencies</artifactId>
      <version>Edgware.RELEASE</version>
      <scope>import</scope>
      <type>pom</type>
    </dependency>
    <!--子工程具有版本需求的资源在parent定义-->
    <!--redis依赖版本1.4.7.RELEASE-->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-redis</artifactId>
      <version>1.4.7.RELEASE</version>
    </dependency>
    <!--mybatis-->
    <dependency>
      <groupId>org.mybatis.spring.boot</groupId>
      <artifactId>mybatis-spring-boot-starter</artifactId>
      <version>1.3.0</version>
    </dependency>
    <!--elasticsearch相关依赖-->
    <dependency>
      <groupId>org.elasticsearch</groupId>
      <artifactId>elasticsearch</artifactId>
      <version>5.5.2</version>
    </dependency>
    <dependency>
      <groupId>org.elasticsearch.client</groupId>
      <artifactId>transport</artifactId>
      <version>5.5.2</version>
    </dependency>
  </dependencies>
</dependencyManagement>
```

```
</dependencies>
</dependencyManagement>
```

## 2.2 公用工具资源工程

- 创建工程（idea的和maven的聚合管理）
- pom文件：

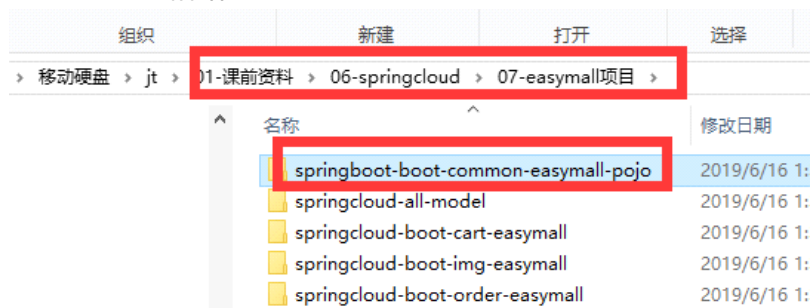
- 继承easymall-parent

```
<parent>
  <artifactId>easymall-parent</artifactId>
  <groupId>cn.tedu</groupId>
  <version>1.0-SNAPSHOT</version>
</parent>
```

- 依赖：starter-web（但是common工程不需要启动）

```
<dependencies>
  <!--starter-web-->
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.11</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

- src下的代码粘贴



粘贴的文件夹，一定要放对地方

代码的内容：

pojo：表示一个工程具备业务含义的所有javabean

util:公用的一些工具类的代码。

cookieUtils：操作cookie，不用在写Cookie[]

MapperUtil:序列化操作对象和json字符串的工具类

MD5Util:加密工具类

UUIDUtil: 生成唯一字符串的工具类

vo: view object,项目根据ajax接收的json字符串结构, 对应编写的视图对象, 例如, ajax要接收json

```
{ "age" :18,"level":19,"driver":"王翠花"}==Object1
```

对象包含三个属性

Integer age

Integer level

String driver

仅仅是按照ajax要求进行对话交互的逻辑。

# jackson依赖问题

2020年2月11日

14:02

## 1.jackson序列化和反序列化计算的包

jackson-core  
jackson-databind  
jackson-annotation

### 1.1返回数据不是视图页面

**No converter found** for class:Integer/User/Product

### 1.2拷贝的common代码

ListUtils  
MapperUtils..

父级工程添加jackson依赖，覆盖本来应该传递过来的jackson资源dependencies中做，强制继承给所有子工程。

```
<dependency>  
  <groupId>com.fasterxml.jackson.core</groupId>  
  <artifactId>jackson-annotations</artifactId>  
  <version>2.8.0</version>  
</dependency>  
<dependency>  
  <groupId>com.fasterxml.jackson.core</groupId>  
  <artifactId>jackson-core</artifactId>  
  <version>2.8.8</version>  
</dependency>  
<dependency>  
  <groupId>com.fasterxml.jackson.core</groupId>  
  <artifactId>jackson-databind</artifactId>  
  <version>2.8.8</version>  
</dependency>
```

## 1.搭建项目

### 1.1步骤

- quickstart module
  - pom文件
    - 继承easymall-parent:可以开发springboot 和springcloud代码
    - 依赖：所有依赖都不需要版本号
      - eureka client
      - 持久层
        - ◆ jdbc,mysql,mybatis
      - redis
      - 依赖公用资源工程
- ```
<dependency>  
  <groupId>cn.tedu</groupId>  
  <artifactId>easymall-common-resources</artifactId>  
  <version>1.0-SNAPSHOT</version>  
</dependency>
```
- application.properties
    - 端口：10001/10002/10003/10004/10005/10006
    - 微服务配置
      - 服务名称：productservice
      - ip优先
      - 注册中心访问接口地址
    - 持久层
      - datasource
        - ◆ root
        - ◆ root
        - ◆ jdbc:mysql:///easydb
        - ◆ com.jdbc.mysql.Driver
      - mybatis
        - ◆ 别名包：封装数据类型，都保存在common工程com.jt.common.pojo
        - ◆ 驼峰命名开启
    - 启动类
      - SpringBootApplication
      - EnableEurekaClient
      - MapperScan
    - 代码空架子
      - productController
      - ProductService
      - ProductMapper
      - ProductMapper.xml

### 1.2导入数据库，查看商品相关表格和结构

课前资料-->06springcloud-->08数据库-->easydb.sql导入到本地库中

商品表格结构：t\_product

每一行数据：都表示一个商品内容

Field Name	Datatype	Len	Default
* product_id	char	36	''
product_name	varchar	100	
product_price	double		0
product_category	varchar	100	''
product_imgurl	varchar	500	''
product_num	int	11	0
product_description	varchar	255	''

product\_id: String 表格主键，保证字符串生成的全局唯一性，UUID  
product\_name:商品名称  
product\_price:商品价钱，小数点的数据，一般在数据库都是用Long类型存储，提升计算速度。在js执行/100操作  
product\_category:商品分类  
product\_imgurl:数据库表格最好不要存储大字段数据blob/text，破坏降低搜索速度，商品的图片链接地址。  
product\_num:库存  
product\_description: 详情描述

## 2.功能开发

### 2.1商品分页查询

- 看接口文件内容

后台接收	/product/manage/pageManage?page=1&rows=1
请求方式	Get
请求参数	Get提交参数 Integer page,Integer rows
返回数据	根据查询结果封装2个数据到EasyUIResult对象中： Integer total:查询的总条数； List<Product> rows:查询分页的数据结果；
备注	数据库查询分页的sql语句关键字limit select * from table limit 0,5 表示从第0条开始，向后查询5条数据，需要解决页数page和limit条件之间的关系

有接收地址：RequestMapping

有参数：controller方法参数

有返回数据：controller方法返回值

- ProductController

```
@Autowired
private ProductService ps;
//商品分页查询
@RequestMapping("pageManage")
public EasyUIResult queryProductByPage(Integer page,Integer rows){
    //page是页数，rows条数，
    //封装数据交给业务层
    return ps.queryProductByPage(page,rows);
}
```

- ProductService

```
@Autowired
private ProductMapper pm;
//商品分页查询
public EasyUIResult queryProductByPage(Integer page,Integer rows){
    //准备一个空对象
    EasyUIResult result=new EasyUIResult();
    /*
    int total:封装分页查询时对应表格数据的全部商品个数
    List rows:利用持久层从数据库查询分页数据的包装对象
    */
    //total 总数 select count(*) from t_product
    int total=pm.selectProductTotal();//65
    result.setTotal(total);
    //利用page rows做分页查询语句，查询List<Product>
    //计算起始位置
    int start=(page-1)*rows;
    List<Product> pList=pm.selectProductByPage(start,rows);
    result.setRows(pList);
    return result;
}
```

- ProductMapper.xml

```
<!--查询总数-->
<select id="selectProductTotal" resultType="int">
    select count(*) from t_product;
</select>
```



```

<!--分页查询 limit-->
<select id="selectProductByPage" resultType="Product">
    select * from t_product limit #{start},#{rows}
</select>

```

- 检查功能是否能够正常执行

可以使用在线解析器

<http://www.bejson.com/jsonviewernew/>

- 页面效果整合



- zuul网关配置路由规则

接受请求到网关的url判断访问的是商品规则

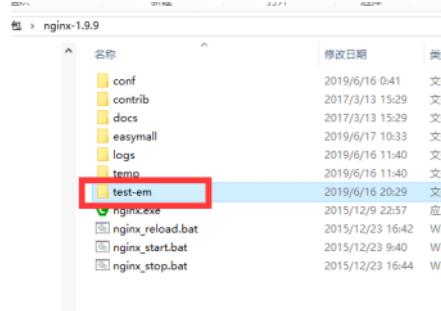
`zuul.routes.product.path=/zuul-product/**`

`zuul.routes.product.service-id=productservice`

- nginx配置

- 静态文件 html js css img 文件

追加文件--03nginx--01安装包解压安装包 test-em的文件夹拷贝到当前nginx根目录中



- nginx.conf实现js请求转发到zuul网关

```

server {
    listen 80;
    server_name www.easymall.com;
    location / {
        root test-em;
        index index.html;
    }

    location /products {
        proxy_pass http://127.0.0.1:8103/zuul-product/product/manage;
        add_header 'Access-Control-Allow-Credentials' 'true';
        add_header 'Access-Control-Allow-Origin' '*';
    }

    location /user {
        proxy_pass http://127.0.0.1:8103/zuul-user/user/manage;
        add_header 'Access-Control-Allow-Credentials' 'true';
        add_header 'Access-Control-Allow-Origin' '*';
    }

    location /cart {
        proxy_pass http://127.0.0.1:8103/zuul-cart/cart/manage;
        add_header 'Access-Control-Allow-Credentials' 'true';
        add_header 'Access-Control-Allow-Origin' '*';
    }

    location /order {
        proxy_pass http://127.0.0.1:8103/zuul-order/order/manage;
        add_header 'Access-Control-Allow-Credentials' 'true';
        add_header 'Access-Control-Allow-Origin' '*';
    }

    location /searchs {
        proxy_pass http://127.0.0.1:8103/zuul-search/search/manage;
        add_header 'Access-Control-Allow-Credentials' 'true';
        add_header 'Access-Control-Allow-Origin' '*';
    }
}

```

```

    }
    location /seckill {
        proxy_pass http://127.0.0.1:8103/zuul-seckill/seckill/manage;
        add_header 'Access-Control-Allow-Credentials' 'true';
        add_header 'Access-Control-Allow-Origin' '*';
    }
    location /uploadImg {
        proxy_pass http://127.0.0.1:8103/zuul-pic/pic/uploadImg;
        add_header 'Access-Control-Allow-Credentials' 'true';
        add_header 'Access-Control-Allow-Origin' '*';
    }
}

```

- hosts文件  
127.0.0.1 [www.easymall.com](http://www.easymall.com)
- 图片效果展示
  - 图片链接访问url一定从数据库获取了  
[http://image.jt.com/upload/2/6/4/a/a/5/2/3/ee6c796a-6333-4cd5-a06e-271d876aac8c\\_589577.jpg](http://image.jt.com/upload/2/6/4/a/a/5/2/3/ee6c796a-6333-4cd5-a06e-271d876aac8c_589577.jpg)
  - 需要使用nginx接收这个image.jt.com域名，访问图片静态资源。
    - 将图片资源从课前资料--02SSM--06图片资源--upload文件夹，拷贝到nginx静态资源文件夹中（D://static）
    - image.jt.com的nginx的虚拟server 配置一个
 

```

server {
    listen 80;
    server_name image.jt.com;
    location / {
        root D://static;
    }
}

```
- hosts文件中准备映射  
127.0.0.1 image.jt.com
- 流转逻辑  
[http://image.jt.com/upload/2/6/4/a/a/5/2/3/ee6c796a-6333-4cd5-a06e-271d876aac8c\\_589577.jpg](http://image.jt.com/upload/2/6/4/a/a/5/2/3/ee6c796a-6333-4cd5-a06e-271d876aac8c_589577.jpg)  
|ip映射到达nginx  
|nginx image.jt.com  
|location / 剩余/upload/2/6/4/a/a/5/2/3/ee6c796a-6333-4cd5-a06e-271d876aac8c\_589577.jpg  
|root d://static/拼接剩余url  
D://static/upload/2/6/4/a/a/5/2/3/ee6c796a-6333-4cd5-a06e-271d876aac8c\_589577.jpg

## 2.2商品的单个查询

只要在第一个功能中实现了zuul网关，nginx配置，后续3个功能不用重新完成页面效果整合

- 接口文件

后台接收	/product/manage/item/{productId}
请求方式	Get
请求参数	路径中携带的参数 String productid
返回数据	返回查询的Product对象,在ajax中解析使用
备注	使用路径参数productId 作为条件查询商品 select * from t_product where product_id=#{productId}

- ProductController

```

//商品单个查询
@RequestMapping("item/{productId}")
//url到达springmvc后，都可以使用{}包含一个路径值
//(值)表示一个变量名称
public Product queryOneProduct(@PathVariable String productId){
    //利用商品id查询商品对象数据
    return ps.queryOneProduct(productId);
}

```

- ProductService

```

public Product queryOneProduct(String productId){
    //调用持久层mapper 发送sql语句封装product对象
}

```

```
//TODO 商品单个查询缓存逻辑，提升查询效率
return pm.selectProductById(productId);
}
```

## 2.3商品新增（表单提交 insert）

### o 接口文件

后台接收	/product/manage/save
请求方式	Post
请求参数	Product product对象接参 缺少id
返回数据	返回SysResult对象的json,其结构: Integer status; 200表示成功,其他表示失败 String msg;成功返回“ok”,失败返回其他信息 Object data;根据需求携带其他数据
备注	返回的SysResult对象是一个标准的和ajax对话的后台vo对象,它可以将当前的操作结果通过status传递给ajax,可以将操作中出现任何后台的消息,例如错误信息,以msg传递给前台,也可以将很多其他数据封装到data传递给前台使用

### o ProductController

```
//商品数据新增
@RequestMapping("save")
public SysResult saveProduct(Product product){
    //在业务层补齐数据新增到数据库
    //SysResult :
    /*
    status:Integer 表示状态数字，200成功，201失败，202表示其他，203...
    msg:String 表示与前端交互的文字，明确的传递信息。200 msg:太棒了，这次操作完成的不错
    data:Object 当ajax请求时查询，可以将查询结果封装到data中，伴随着status，msg将交互结构完整的返回
    这个SysResult对象的结构，标准的与ajax对话的结构，基本上任何应用场景，都可以使用这个对象。
    */
    try{
        //调用业务层新增商品
        ps.saveProduct(product);
        //返回一个表示操作正常的SysResult status=200
        return SysResult.ok();//status=200 msg=ok data=null;
    }catch(Exception e){
        //进入异常表示新增商品失败
        e.printStackTrace();//打印给自己看
        return SysResult.build(201,"到底怎么回事",null);
    }
}
```

### o ProductService

```
//新增商品
public void saveProduct(Product product){
    //补齐id值
    String productId= UUIDUtil.getUUID();
    //一台服务器生成的uuid每次一定不一样的
    //服务器集群生成uuid有可能一样（几率极低）
    product.setProductId(productId);
    //TODO redis缓存
    pm.insertProduct(product);
}
```

## 2.4商品修改 (表单提交 update)

### 接口文件

后台接收	/product/manage/update
请求方式	Post
请求参数	Product product (属性值完整的)
返回数据	返回SysResult对象的json,其结构: Integer status; 200表示成功,其他表示失败 String msg;成功返回“ok”,失败返回其他信息 Object data;根据需求携带其他数据

### ProductController

//修改商品

```
@RequestMapping("update")
public SysResult updateProduct(Product product){
    try{
        ps.updateProduct(product);
        return SysResult.ok();
    }catch(Exception e){
        e.printStackTrace();
        return SysResult.build(201,"错了",null);
    }
}
```

### ProductService

```
public void updateProduct(Product product){
    //TODO redis缓存, 高并发下保证缓存与数据库数据一致
    pm.updateProductByld(product);
}
```

## 商品单个查询添加缓存逻辑

- a. 先将user系统中编写相关redis整合代码拷贝
  - i. `clusterConfig` cn.tedu.product.config
  - ii. 在商品系统依赖spring-boot-starter-redis
  - iii. application.properties中准备属性值
- b. 在代码中注入使用JedisCluster

@Autowired

```
private JedisCluster cluster;

public Product queryOneProduct(String productId){
    //调用持久层mapper 发送sql语句封装product对象
    //product数据在redis存储结构 string类型 json key值保证商品数据唯一性 productId
    String productKey="product_" + productId;
    //使用集群判断商品数据是否存在
    try{
        ObjectMapper mp= MapperUtil.MP;
        if(cluster.exists(productKey)){
            //if如果进入, 说明能够从redis集群获取商品数据
            //可以通过key值返回json字符串
            String pJson = cluster.get(productKey);
            //将json转化为对象
            Product product= mp.readValue(pJson,Product.class);//pJson源数据, Product.class
            //把json转化回来的类型反射对象
            return product;
        }else{
            //说明缓存未命中, 到数据库查询数据
            Product product = pm.selectProductByld(productId);
            //不着急返回数据, 将product对象转化为json放到redis中
```

```

        String pJson = mp.writeValueAsString(product);
        //将缓存存储在redis
        cluster.setex(productKey,60*60*24,pJson);
        //查询结果返回给前端
        return product;
    }
} catch (Exception e) {
    e.printStackTrace();
    return null;
}
}
}

```

#### c. 新增，修改商品缓存处理

//新增商品

```

public void saveProduct(Product product){
    //补齐id值
    String productId= UUIDUtil.getUUID();
    //一台服务器生成的uuid每次一定不一样的
    //服务器集群生成uuid有可能一样（几率极低）
    product.setProductId(productId);
    //新增数据可以看成绝大多数都是热点数据。可以在新增时直接添加缓存逻辑
    try{
        ObjectMapper mp= MapperUtil.MP;
        String productKey="product_"+productId;
        //转化json
        String pJson = mp.writeValueAsString(product);
        cluster.setex(productKey,60*60*24,pJson);
    } catch (Exception e) {
        e.printStackTrace();
    }
    pm.insertProduct(product);
}

public void updateProduct(Product product){
    //保证更新商品与缓存数据一致
    //更新之前将商品缓存数据删除
    String productKey="product_"+product.getProductId();
    cluster.del(productKey);
    pm.updateProductById(product);
}
}

```

面试题：数据库2000万条数据，但是redis只能存储最大缓存容量200万条。如何保证200万条绝大部分是热点数据？热点数据--热门数据。例如：商品热门数据，小米10pro。。冷门数据相对访问频率不大，不太会需要存储在缓存。例如：诺基亚1110（考察的就是淘汰策略使用）

可以通过redis的配置实现 绝大部分数据存储的都是热点数据。

LRU：最近最久未使用删除策略

redis中可以通过设置缓存内存容量，达到内存上限后，可以根据设置删除淘汰数据策略实现数据淘汰，默认情况下，没有淘汰策略，一旦达到内容上限，将会抛出错误。

```
# output buffers (but this is not needed in the po
ction').
#
# maxmemory <bytes>
# MAXMEMORY POLICY: how Redis will select what to
xmemory
# is reached. You can select among five behaviors:
#
@
```

537,:

没有设置maxmemory 默认最大内存占用物理内存上限。一旦到达上限，系统不可用了。  
配合淘汰策略，达到内存上限进行数据删除。

volatile-lru -> remove the key with an expire set using an LRU algorithm  
allkeys-lru -> remove any key according to the LRU algorithm  
volatile-random -> remove a random key with an expire set  
allkeys-random -> remove a random key, any key  
volatile-ttl -> remove the key with the nearest expire time (minor TTL)  
noeviction -> don't expire at all, just return an error on write operations

volatile-lru: 对于已经设置了有超时时间的数据，利用LRU计算算法实现淘汰

volatile-random: 对于设置了超时时间的数据，采用随机淘汰

volatile-ttl: 对于设置了超时时间的数据，采用到达了内存上限之后，计算谁的剩余时间少先淘汰谁。

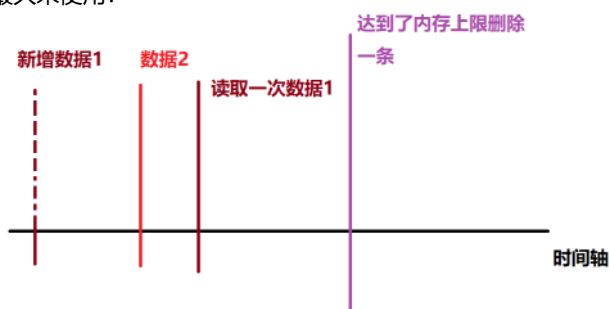
allkeys-lru: 所有数据包括永久数据，采用lru算法淘汰

allkeys-random: 所有数据随机淘汰

noeviction: 没有淘汰策略 到达了上限，返回error

LRU: 可以保证绝大部分数据在淘汰发生时，是热点数据（频繁访问）；

最近最久未使用:



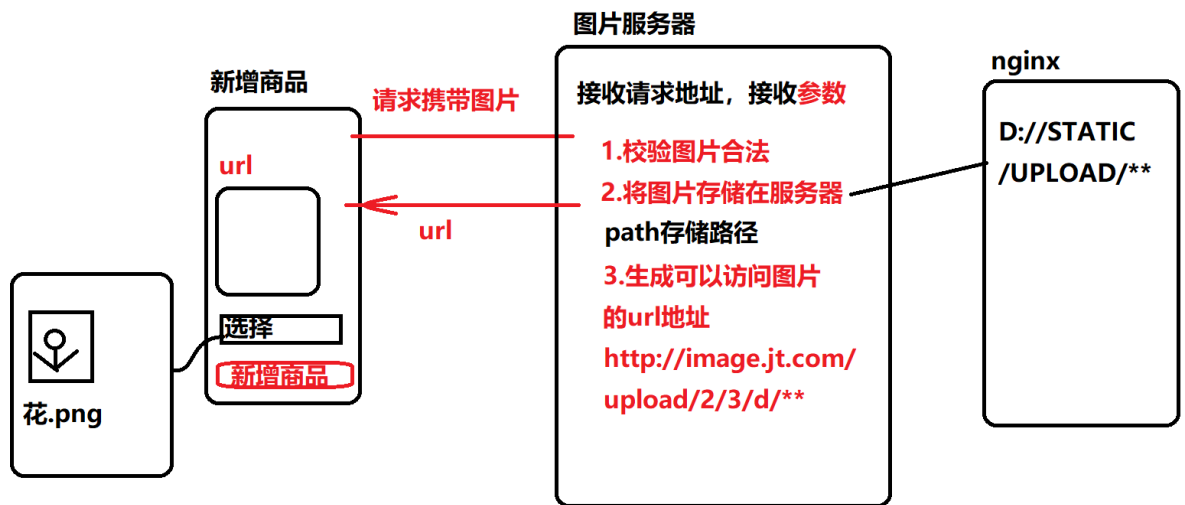
频繁访问的热点数据，绝大多数都会在达到内存上限时，不满足最近最久未使用。

可以根据缓存存储的数据特点--超时，没超时，指定volatile-lru还是allkeys-lru

# 图片上传

2020年2月11日 15:06

## 1.easymall图片上传逻辑



图片easymall处理的逻辑，仅仅是实现上传，回显。  
没有解决海量图片的问题。

## 2.图片工程功能开发

### 2.1搭建项目

- quickstart
- pom:
  - 继承自定义easymall-parent
  - 依赖:
    - eureka client
    - common resources
- application.properties
  - 端口10002
  - 服务名称: imgservice
  - 注册中心地址: 8761
- 启动类
  - springBootApplication
  - EnableEurekaClient
- 代码架子 (springmvc支持的代码api)
  - picController
  - PicService

### 2.2开发具体功能

- 接口文件

后台接收地址	/pic/upload
请求方式	Post

请求参数	MultipartFile pic
返回数据	PicUploadResult对象的json,结构是: String url 生成的访问图片路径 Integer error 上传成功和失败的标志 0表示无错误 1表示有错误
备注	MultipartFile类是springmvc专门提供实现文件上传使用的类, 图片算文件, 视频算文件, 表格excel 文档word都可以通过这个类来接收数据

○ PicController

```
@Autowired
private PicService ps;
//接收请求参数, 调用业务层实现图片上传
@RequestMapping("pic/upload")
public PicUploadResult picUpload(MultipartFile pic){
    return ps.upload(pic);
}
```

○ PicService

```
public PicUploadResult upload(MultipartFile pic){
    /*
    图片上传实现的流程
    1.对上传的文件做校验是否合法
    1.1判断后缀是否属于图片 jpg,png,gif...
    2.生成存储路径 d:/static/upload/1/d/d/3/d/3/d/3/
    3.图片重命名, 防止上传同一个图片名称, 存储到文件夹
    d:/static/upload/1/d/d/3/d/3/d/3/重命名.png
    4.生成对应可以访问到该文件的url地址, 通过nginx静态访问
    http://image.it.com/upload/1/d/d/3/d/3/d/3/重命名.png
    5.封装数据, 返回对象 url包装返回给ajax使用
    整个流程过程任何一个位置出现异常, 都会导致图片上传失败
    try catch包含所有代码
    */
    //PicUploadResult 表示图片上传是否成功, 成功携带url
    PicUploadResult result=new PicUploadResult();//error=0 url=null
    //编写上传图片具体逻辑
    try{
        //从pic参数中拿到文件原名称, 解析后缀.png,.jpg..判断合法
        //拿到原名称
        String oName = pic.getOriginalFilename();//qige.png
        //通过后缀判断, 截取后缀
        String extName = oName.substring(oName.lastIndexOf(".");
        //判断合法, 正则表达式
        if(!extName.matches("(.(png|gif|jpg)$")){
            //说明 后缀截取值不满足要求, 不是图片, 不合法
            result.setError(1);
            return result;
        }
        //根据图片, 生成一个多级路径/upload/1/d/3/d/3/d/3/d/,调用一个工具类
        String dir = UploadUtil.getUploadPath(oName, "upload");
        //根据生成的dir, 创建磁盘d:/static/upload/1/d/3/d/3/d/3/d//
```



```

String path="d:/static/"+dir+"/";
File _dir=new File(path);//代表文件夹内存对象
//如果没有这个文件夹需要创建多级目录
if(!_dir.exists()){
    _dir.mkdirs();
}
//重命名图片，存储图片到多级目录下
String nName= UUIDUtil.getUUID()+extName;//3kd3-3md23dk434dfs23-3.jpg
pic.transferTo(new File(path+nName));
//根据上传的图片，生成可以url访问图片
String url="http://image.jt.com/"+dir+"/"+nName;
//http://image.jt.com/upload/5/e/d/5/4/5/e/b/5f0d34dc-157f-49ba-ad39-1b28927ba6ae_
1005714.jpg
result.setUrl(url);
return result;
}catch(Exception e){
    e.printStackTrace();
    //出现异常了上传图片表示失败
    result.setError(1);
    return result;
}
}

```

- 页面整合

zuul网关路由配置一对

#pic图片

zuul.routes.pic.path=/zuul-pic/\*\*

zuul.routes.pic.service-id=imgservice

nginx转发到网关location配置

```

location /uploadImg{
    proxy_pass http://127.0.0.1:8103/zuul-pic/pic/upload;
}

```

## 1.观察用户相关表格

### 1.1t\_user

每一行数据：表示一个系统的用户信息

Field Name	Datatype
* user_id	char
user_name	varchar
user_password	varchar
user_nickname	varchar
user_email	varchar
user_type	int

user\_id:String 主键值，全局唯一

user\_name:用户名称，在数据库做了唯一校验，用户名测试时不能注册相同用户名

user\_password:数据库中密码不能使用明文。

user\_nickname:昵称

user\_email:邮箱

user\_type:没有使用的字段，本来愿意表示一个与用户界别相关的字段

## 2.用户系统功能

### 2.1注册

新的用户使用系统进行表单填写，注册提交生成系统新用户数据。

- 注册表单中进行用户名重复校验 (select)
- 注册表单提交(insert)

### 2.2登录 (redis学习之前无法实现)

登录使用用户名密码，服务校验，成功后才能使用不同功能。

通过对session共享文件解释，引入redis

学习了redis相关功能就可以将登陆实现（三个代码版本的编写）

## 3.用户系统搭建功能编写

### 3.1搭建

- quickstart
- pom:
  - 继承easymall-parent
  - 依赖:
    - common-resources
    - 被调用的服务 eureka client
    - 持久层相关
    - redis依赖
- application.properties (product)
  - 端口: 10003
  - 服务名称: userservice
  - 注册中心
  - 持久层相关配置
- 启动类
  - springboot核心注解
  - 开启eurekaclient
  - 扫描mapper接口包
- 代码空架子

### 3.2注册功能

- 用户名重复校验

在注册表单中，通过填写的用户名，ajax先发送请求到服务中验证用户名是否在数据库存在

- 接口文件

后台接收	/user/manage/checkUserName
请求方式	Post
请求参数	String userName
返回数据	返回SysResult对象的json,其结构: Integer status; 200可用(不存在),其他表示不可用(已经存在) String msg;成功返回“ok”,失败返回其他信息 Object data;根据需求携带其他数据
备注	SELECT COUNT(*) FROM t_user WHERE user_name="eseee"

- UserController

```
//注入一个userService
```

```
@Autowired
```

```
private UserService us;
```

```
@RequestMapping("checkUserName")
```

```
public SysResult checkUserName(String userName){
    return us.checkUserName(userName);
}
```

```
UserService
@Autowired
private UserMapper um;
public SysResult checkUserName(String userName){
    //从数据库读取数据判断是否可用
    int exist=um.selectCountByUserName(userName);
    //根据返回结果返回SysResult的数据
    if(exist==1){//已经存在 不可用 status=201
        return SysResult.build(201,"",null);
    }else{
        //表示等于0 可用 status=200
        return SysResult.ok();
    }
}
```

#### ○ 页面整合

- zuul网关路由匹配
  - #user用户系统
  - zuul.routes.user.path=/zuul-user/\*\*
  - zuul.routes.user.service-id=userservice
- nginx配置不需要修改

### 3.3 登录功能

#### 3.3.1 代码替换成jedisCluster

不替换使用的还是6379 6380 6381的一致性hash的分布式计算。

注入JedisCluster

删除Pool的获取资源

删除Pool的finally还回资源

#### 3.3.2 登录状态的续租

问题：登录时，保存了用户状态超时时间2小时，一旦用户访问超过2小时，会在2小时到达时登录状态突然小时，要求重新登录才能做后续操作。

解决思路：续租。判断什么时候需要续租。续租多长时间。

利用redis提供的超时ttl查看和expire实现续租

```
public String queryUserData(String ticket){
    try{
        //实现续租逻辑，判断剩余时间。如果有个访问超过1.30小时 剩余30分钟小于30分钟，给超时
        //判断剩余时间
        Long leftTime = jedis.pttl(ticket);
        if(leftTime<1000*60*30){
            //剩余时间不足30分钟，重新刷一次超时新的2小时
            //延长 leftTime+想延长时间
            jedis.pexpire(ticket,1000*60*60*2);
        }
        return jedis.get(ticket);
    }catch(Exception e){
        e.printStackTrace();
        return null;
    }
}
```

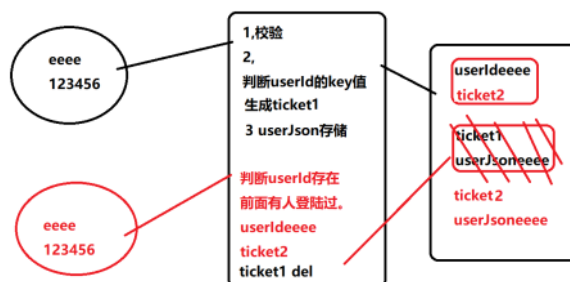
#### 3.3.3 用户登录顶替

可以在多个客户端拿着一个用户用户名密码，实现登录操作。可以对其做限制，一个用户最多登录一次。也可以在登录时，利用redis的结构实现在redis中准备2个key-value的数据

第一个：以userId为key值，当前登录的ticket为value，用户id唯一，一个用户只能在redis中存储一个这样的key-value，无论同一个用户在不同客户端登录多少次，只能从redis获取一个---有效ticket只有一个。最后登录那个人

第二个：ticket-userjson，在保证了一个用户只有一个有效ticket时，可以对正常用户状态数据做维护。

顶替的逻辑



```

@Autowired
private JedisCluster jedis;
public String doLogin(User user){
    String ticket="";
    user.setUserPassword(MD5Util.md5(user.getUserPassword()));
    User exist=um.selectUserByNameAndPw(user);//select * from t_user where name= and pw=
    if(exist==null){
        //说明没有查询到user对象登陆时不合法的
        return ticket;
    }else{
        ticket="EM_TICKET"+System.currentTimeMillis()+user.getUserName();
        //生成顶替逻辑中的userId的唯一值
        String loginKey="login_"+exist.getUserId();
        //判断是否需要顶替
        if(jedis.exists(loginKey)){
            //进入到if说明有人已经登录，要把它登录时使用的ticket删掉
            String lastTicket=jedis.get(loginKey);
            jedis.del(lastTicket);
            //在这个loginKey覆盖上次登录有效ticket
            jedis.set(loginKey,ticket);
        }
        //如果不存在，就把自己登录时用2个key-value设置在redis
        jedis.set(loginKey,ticket);
        exist.setUserPassword(null);
        ObjectMapper om= MapperUtil.MP;
        try{
            String uJson = om.writeValueAsString(exist);

            jedis.setex(ticket,60*60*2,uJson);//不能set永久数据，假设超时时间2小时
        }catch(Exception e){
            e.printStackTrace();
            return "";
        }
    }
    return ticket;
}
}

```

### 3.3.4 登录顶替逻辑扩展

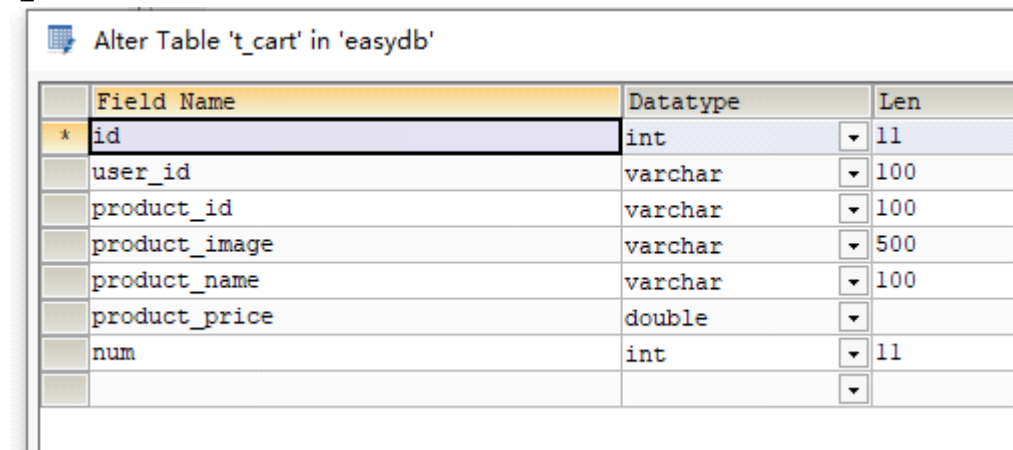
只允许一个用户在登录。扩展变成多个同时登录。一个账号可以有3个人在同时使用。  
userId不使用String 使用List，添加多个元素

# 购物车系统

2020年5月2日 17:19

## 1.数据库表格结构

t\_cart:每一行数据都表示某个用户的某个购物车商品。



	Field Name	Datatype	Len
*	id	int	11
	user_id	varchar	100
	product_id	varchar	100
	product_image	varchar	500
	product_name	varchar	100
	product_price	double	
	num	int	11

id:做表格结构的，没有业务意义；

user\_id:表示一个购物车商品所属的用户

product\_id:表示一个购物车商品。

上述2个字段user\_id,product\_id同时生效，就可以唯一的定位购物车表格一行数据---复合主键；

product\_image:商品图片链接地址

product\_name:商品名称

product\_price:商品价钱

完全可以去掉，使用product\_id作为外键关联从商品表格关联查询获取。冗余字段。减少关联查询，提升表格操作效率。

num:购物车中商品数量。

## 2.搭建新的购物车系统

### 2.1步骤

a. quickstart

b. pom:

i. 继承 easymall-parent

ii. 依赖:

1. common-resources

2. 持久层: jdbc mysql mybatis

3. eureka-client

4. ribbon: 负载均衡客户端从购物车系统中访问商品系统获取冗余字段的信息。

c. application.properties(user,product随便粘贴过来一个)

- i. 端口号: 10004
- ii. 服务名称: cartservice
- d. 启动类
- e. 代码架子

## 2.2根据接口文件开发具体功能

### a. 查询我的购物车

后台接收	/cart/manage/query?userId=**
请求方式	Get
请求参数	String userId
返回数据	返回List<Cart>数据
备注	select * from t_cart where user_id=#{userId}

- i. CartController  
CartService
- ii. 页面效果整合
  - 1. 配置zuul网关的路由规则

#购物车使用

zuul.routes.cart.path=/zuul-cart/\*\*

zuul.routes.cart.service-id=cartservice

order系统

search系统

seckill系统

#### 2. nginx (略)

### b. 新增购物车

新增购物车在页面中看到的就是添加购物车操作, 对于系统数据, 不一定是insert。

新增数据时, 应该先判断数据库是否已经存在了该用户的该商品

(userId,productId)

如果不存在, 将封装好的cart对象新增到数据库(insert)

如果存在, 将存在的num+新增过来的num 修改数据库 (update)

#### i. 接口文件

后台接收	/cart/manage/save
请求方式	post
请求参数	Cart cart, cart中仅仅存在三个字段值userId, productId,num, product_name,product_image,product_price都不在参数
返回数	返回SysResult对象的json, 其结构:

据	Integer status; 200表示成功,其他表示失败 String msg;成功返回 “ok” ,失败返回其他信息 Object data;
备注1	为了获取这三个字段, 可以利用productId作为查询需要的内容。select product_name product_pric product_image from t_product where product_id=#{参数}--购物车微服务需要访问商品
备注2	ribbon发起服务调用服务的请求, 访问商品系统单个商品查询。

## ii. 新增购物车数据

1. 需要从productservice中调用微服务功能获取查询的product数据
2. 引入ribbon负载均衡客户端通过服务名称调用商品功能
3. cart系统中, 生成支持ribbon访问的restTemplate对象

## c. 更新购物车num数量

后台接收	/cart/manage/update
请求方式	get
请求参数	Cart cart,具有三个属性userId,productid,num
返回数据	返回SysResult对象的json,其结构: Integer status; 200表示成功,其他表示失败 String msg;成功返回 “ok” ,失败返回其他信息 Object data;

## d. 删除购物车

delete from t\_cart where user\_id= and product\_id=

后台接收	/cart/manage/delete
请求方式	get
请求参数	Cart cart,具有两个属性userId,productid
返回数据	返回SysResult对象的json,其结构: Integer status; 200表示成功,其他表示失败 String msg;成功返回 “ok” ,失败返回其他信息 Object data;

# 订单系统的开发

2020年5月2日 17:25

## 1. 功能开发

### 1.1 订单查询

连接mycat没有数据，使用本地数据库，注意使用连接url地址赋予一些参数。  
否则会出现各种访问数据库的问题（sql变化）

#### a. 接口文件

后台接	/order/manage/query/{userId}
请求方	get
请求参	路径参数String userId
返回数	List<Order>数据, 根据长度判断查询是否成功, 如果长度为0可能是未登录也可

#### b. order对象的封装

##### i. pojo不同名字的意义

1. entity: 实体类，一个类对应一个表格，类的对象表格一个表格封装的一行数据。（product,user,car 都可以叫entity）
2. domain: 域对象，根据应用场景，封装多个表格关联查询的数据，（order 域对象）
3. dto: 专门封装的对象对应页面表单结构的数据。（product,user 都是dto）前端传过来的数据结构
4. vo: 后台系统经过业务逻辑代码之后，封装的返回给前端ajax使用对话对象（\*\*Result对象）

#### c. 业务层封装order域对象

##### i. 从两个表格获取相关数据，第一种使用业务层代码

```
//先查父表数据 select * from t_order where user_id
List<Order> oList=om.selectOrder(userId);
//挨个封装
for(Order order:oList){
    //每循环一个 对应使用orderId查询子表
    //select * from t_order_item where order_id=
    List<OrderItem>
oiList=om.selectOrderItem(order.getOrderId());
    order.setOrderItems(oiList);
}
return oList;
```

优点：如果2个相关数据的表格没有按照ER分片表，或者使用mycat无



法管理2个相关表格的相关数据，可以在业务层使用业务逻辑代码实现访问数据库数据

缺点：影响这个方法执行的性能。

ii. 从持久层框架一次性拿到所有数据封装对应对象order

1. mybatis可以利用返回resultMap实现1对1,1对多数据封装

d. 页面效果整合

i. zuul路由配置

#订单使用

zuul.routes.order.path=/zuul-order/\*\*

zuul.routes.order.service-id=orderservice

## 1.2订单新增

一次接受一个order对象，包含父表的一条数据，和子表的若干条。

一次性执行insert语句多条。连接datasource数据源的url地址要添加一个参数 allowMultiQueries=true,否则，在执行第二个，或更多insert时，会出现mybatis持久层。仅仅是连接本地库时需要添加的参数，如果连接的是mycat不需要添加，mycat在使用提供的writeHost readHost信息连接后端数据创建datasource时，已经添加了这个参数

?useUnicode=true&characterEncoding=utf8

&autoReconnect=true&allowMultiQueries=true

a. 接口文件

后台接收	/order/manage/save
请求方式	Post
请求参数	Order order,其中缺少order的id值
返回数据	返回SysResult对象的json,其结构: Integer status; 200表示成功,其他表示失败 String msg;成功返回 "ok" ,失败返回其他信息 Object data;根据需求携带其他数据

b. 将数据写入数据库，2个表格，两种方法

i. 使用业务层

//order对象中的orderItems 元素对象orderItem也需要

//orderId属性值。

om.insertOrder(order);//写主表

for(OrderItem oi:order.getOrderItems()){

//oi对应t\_order\_item中的一个行数据

//每个oi对象都缺少orderId

oi.setOrderId(order.getOrderId());

//新增oi到t\_order\_item

om.insertOrderItem(oi);

}

i. 映射文件编写多条插入的insert语句

## 1.3删除订单

### a. 接口文件

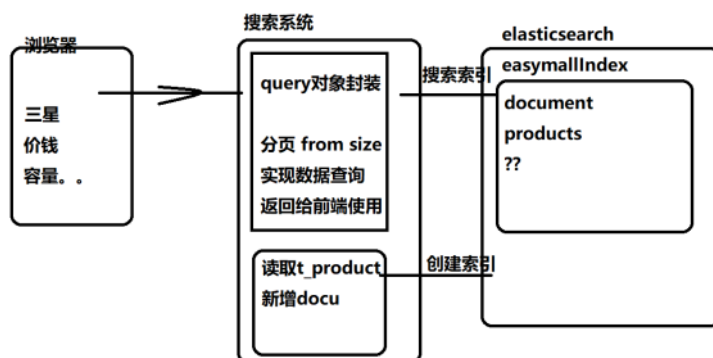
后台接收	/order/manage/delete/{orderId}
请求方式	Get
请求参数	路径参数String orderId
返回数据	返回SysResult对象的json,其结构: Integer status; 200表示成功,其他表示失败 String msg;成功返回 “ok” ,失败返回其他信息 Object data;根据需求携带其他数据

# 搜索系统

2020年5月2日 17:28

## 搜索系统结构

- 数据整理封装，放到es中保存，index
- 客户端系统访问es封装请求的参数为query对象查询数据



通过业务逻辑考虑数据一致性，数据库和索引文件。数据库的数据源可能发生各种数据变动，考虑保证es中index的一致性。

## ELK家族

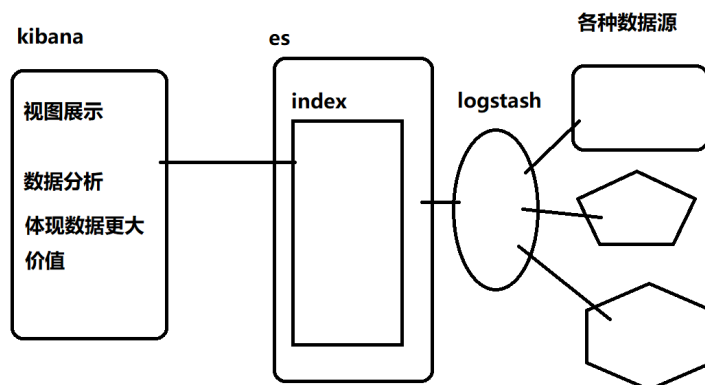
基于elasticsearch的全文检索功能行程一整套技术体系。包含了数据导入，数据更新，包含了全文检索数据管理，包含了数据视图化展示和分析。

E: elasticsearch--有大量的数据 index

L: logstash--日志数据收集器，将各种数据源的数据导入es中

k: kibana--视图展示，数据分析工具

这样一个结构，使得存储在es中的数据可以实时更新，并且除了提供搜索以外，还可以实现更高的价值



## 1.搭建一个搜索系统

### 1.1搭建系统步骤

- 创建一个搜索工程

- i. quickstart
- ii. pom:
  - 1. 继承
  - 2. 依赖:
    - a. common-resource
    - b. eureka-client
    - c. elasticsearch2个
    - d. 持久层3个jdbc,mysql,mybatis
- iii. application.properties
  - 1. 10006
  - 2. searchservice
  - 3. 按照配置整合准备es信息
- iv. 启动类
  - 1. springbootapplication
  - 2. enableeurekaclient
  - 3. mapperscan
- b. springboot和transportclient整合
  - i. 准备一个配置类
  - ii. 使用配置类读取属性es.nodes
  - iii. 客户端连接对象的初始化代码

```

1 package cn.tedu.search.config;
2
3
4 import
5 import org.elasticsearch.common.settings.Settings;
6 import
7 import org.elasticsearch.common.transport.InetSocketTransport
8 import
9 import
10 org.springframework.boot.context.properties.Configuration
11 import org.springframework.context.annotation.Bean;
12 import
13
14 import java.net.InetAddress;
15 import java.util.List;
16
17
18 /**
19  * 是一个配置类，读取属性，初始化封装一个
20  * transportclient对象
21  */
22
23 @Configuration
24 @ConfigurationProperties("es")
25 public class ESConfig {
26     //application.properties准备属性
27     private List<String> nodes;
28     @Bean
29     public TransportClient initTransportClient() {
30         //初始化一个Transportclient对象
31         try{
32             TransportClient client=new
33                 (Settings.EMPTY);
34

```

```

35         for(String node:nodes){
36             //node解析ip地址和端口
37             String host=node.split(":")[0];
38             int port=Integer.parseInt(node.split(":")
39             //InetSocketAddress封装使用ip port
40             InetSocketAddress address=new
41             InetSocketAddress(
42
43
44                 client.addTransportAddress(address);
45             }
46             return client;
47         }catch(Exception e){
48             e.printStackTrace();
49             return null;
50         }
51     }
    public List<String> getNodes() {
        return nodes;
    }

    public void setNodes(List<String> nodes) {
        this.nodes = nodes;
    }
}

```

准备application.properties属性

es.nodes=10.9.140.184:9300,10.9.100.26:9300,10.9.48.69:9300

## 1.2手动创建一个索引导入数据

- i. 索引文件：easymallindex 导入的数据 从数据库查询List<Product>挨个存放到索引中。

```

1 package cn.tedu.search.controller;
2
3
4 import cn.tedu.search.mapper.CreateIndexMapper;
5 import com.jt.common.pojo.Product;
6 import com.jt.common.utils.MapperUtil;
7 import com.jt.common.utils.UUIDUtil;
8 import
9 org.elasticsearch.action.admin.indices.exists.indices
10 .IndicesExistsRequestBuilder;
11 import
12 org.elasticsearch.action.admin.indices.exists.indices
13 .IndicesExistsResponse;
14 import
15 import org.elasticsearch.client.AdminClient;
16 import org.elasticsearch.client.IndicesAdminClient;
17 import
18 import
19 import
20 import
21
22 import java.util.List;
23
24
25 @RestController
26 public class CreateIndexController {
27     @Autowired
28     private CreateIndexMapper createIndexMapper;
29     /*
30     注入一个mapper对象，获取list的商品数据
31

```

32	按照新增文档的测试代码将商品数据转化成json
33	存入到es集群中。通过外部的一次调用实现这个过程
34	*/
35	@Autowired
36	private TransportClient client;
37	@RequestMapping("create")
38	public String createIndex(String indexName,
39	String typeName){
40	
41	//读取数据库 商品数据
42	List<Product> pList=
43	//封装成json字符串client发起请求写入es索引
44	try{
45	
46	//判断索引是否存在, 不存在则创建
47	AdminClient admin = client.admin();
48	IndicesAdminClient indices =
49	IndicesExistsRequestBuilder request =
50	indices.prepareExists(indexName);
51	IndicesExistsResponse response =
52	if(!response.isExists()){
53	//索引不存在, 创建索引
54	
55	}
56	for (Product p:pList){
57	
58	//转化对象为json
59	String pJson =
60	//调用client的api生成request 将json包装
61	IndexRequestBuilder request1 =
	client.prepareIndex(indexName, typeName,
	//pJson放到request的source数据中
	request1.setSource(pJson);
	request1.get();
	}
	return "success";
	}catch(Exception e){
	e.printStackTrace();
	return "fail";
	}
	}
	}

### 1.3按照接口文件实现搜索功能编写

#### i. 接口文件

后台接收	/search/manage/query?query=三星&page=1&rows=5
请求方式	get
请求参数	String query,Integer page,Integer rows
返回数据	List<Product>查询返回结果
备注	分页条件page rows是写死的并没有实现前端分页插件

#### ii. SearchController

1	package cn.tedu.search.controller;
2	
3	
4	import cn.tedu.search.service.SearchService;
5	import com.jt.common.pojo.Product;
6	import
7	import
8	import

```

9
10 import java.util.List;
11
12
13 @RestController
14 public class SearchController {
15     @Autowired
16     private SearchService searchService;
17     //调用一个方法
18     @RequestMapping("search/manage/query")
19     public List<Product> search(String query,Integer
20 page,Integer rows){
11         return searchService.search(query,page,rows);
12     }
13 }

```

### iii. SearchService

```

1 package cn.tedu.search.service;
2
3
4 import com.jt.common.pojo.Product;
5 import com.jt.common.utils.MapperUtil;
6 import
7 import
8 import
9 import
10 import org.elasticsearch.index.query.QueryBuilders;
11 import org.elasticsearch.search.SearchHit;
12 import org.elasticsearch.search.SearchHits;
13 import
14 import org.springframework.stereotype.Service;
15
16 import java.util.ArrayList;
17 import java.util.List;
18
19
20 @Service
21 public class SearchService {
22     @Autowired
23     private TransportClient client;
24     public List<Product> search(String text, Integer
25 page, Integer rows) {
26         //搜索功能 封装查询条件query matchQuery
27         MatchQueryBuilder query =
28             "productName", text);
29         //开始调用查询, 解析数据
30         SearchRequestBuilder request =
31 client.prepareSearch("easyindex");
32
33 request.setQuery(query).setFrom((page-1)*rows).setSize
34 SearchResponse response = request.get();
35 //解析数据 获取第二层的hits对象
36 SearchHits topHits = response.getHits();//第一
37 SearchHit[] hits = topHits.getHits();//第二层
38 try{
39     //准备一个list商品空对象
40     List<Product> pList=new ArrayList<>();
41     for(SearchHit hit:hits){
42         //每循环一次,都可以从hit中拿到source字符串
43         String pJson =
44         //从字符串转化到对象过程,

```

```
49                                     //传递参数json字符串, 和转化对象类反射
50                                     Product p =
51                                     MapperUtil.MP.readValue(pJson, Product.class);
52                                     //add到返回的list中
                                     pList.add(p);
                                     }
                                     return pList;
                                     }catch(Exception e){
                                     e.printStackTrace();
                                     return null;
                                     }
                                     }
                                     }
```

## 1.4整合前端页面

zuul网关的路由规则配置

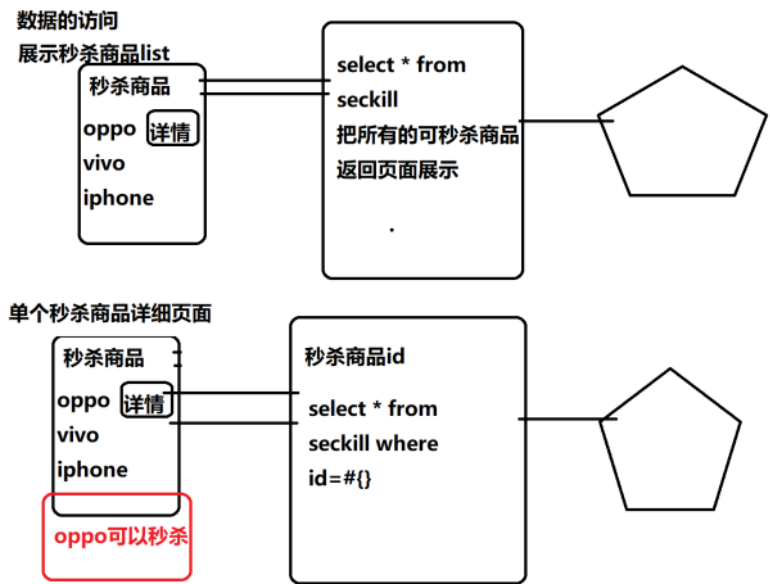


# 秒杀系统

2020年5月2日 17:27

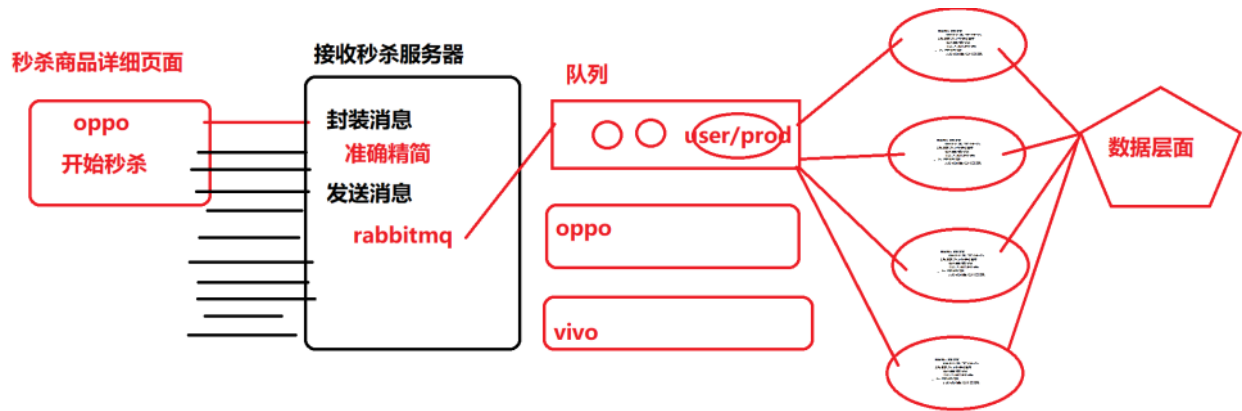
## 1. 秒杀系统逻辑

### 1.1 页面数据查询



搭建项目系统，实现页面数据的获取

### 1.2 秒杀核心逻辑



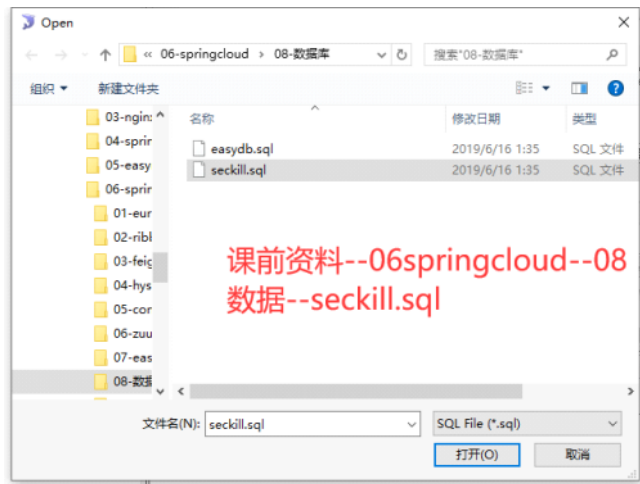
前端接收请求的系统，尽量提升单位时间并发，减少线程处理任务的时间。作为生产端

发送消息。消息携带信息精简准确，谁秒杀了什么商品。userId/seckillId  
消费端可以启动在多个进程中，并发争抢处理消息，实现数据层的判断减库存入库。

## 2. 实现秒杀系统的开发

### 2.1 读取数据库数据

a. 导入数据库表格 seckill.sql



i. seckill:每一行数据都表示一个可秒杀的商品

ALTER TABLE seckill IN seckill

	Field Name	Datatype	Len
*	seckill_id	bigint	20
	name	varchar	120
	number	int	11
	initial_price	bigint	20
	seckill_price	bigint	20
	sell_point	varchar	500
	create_time	timestamp	
	start_time	timestamp	
	end_time	timestamp	

当前系统时间小于start-time, 秒杀未开始

大于end-time 秒杀结束

在中间, 正常进行秒杀

ii. success:成功秒杀的数据入库信息, 谁秒杀了什么

	Field Name	Datatype	Len
*	success_id	bigint	20
	seckill_id	bigint	20
	user_phone	bigint	20
	state	tinyint	4
	create_time	timestamp	

b. 创建一个系统 (seckill系统)

i. quickstart

ii. pom:

1. 继承

2. 依赖:

a. common-resources

b. eureka-client

c. 持久层

d. amqp

iii. application.properties

1. 端口

2. 服务名称

### 3. datasource的数据库名称seckill

#### c. 启动类

- i. SpringBootApplication
- ii. MapperScan
- iii. EnableEurekaClient

#### d. 读取秒杀商品list

##### i. 接口文件

后台接收	seckill/manage/list
请求方式	get
请求参数	空的
返回数据	List<Seckill>的查询数据

#### e. 查询某个秒杀商品数据

##### i. 接口文件

后台接收	/seckill/manage/detail?seckillId=1
请求方式	Get
请求参数	Long seckillId
返回数据	Seckill根据id的查询对象

#### f. 页面效果整合

页面前端有个错误数据。修改

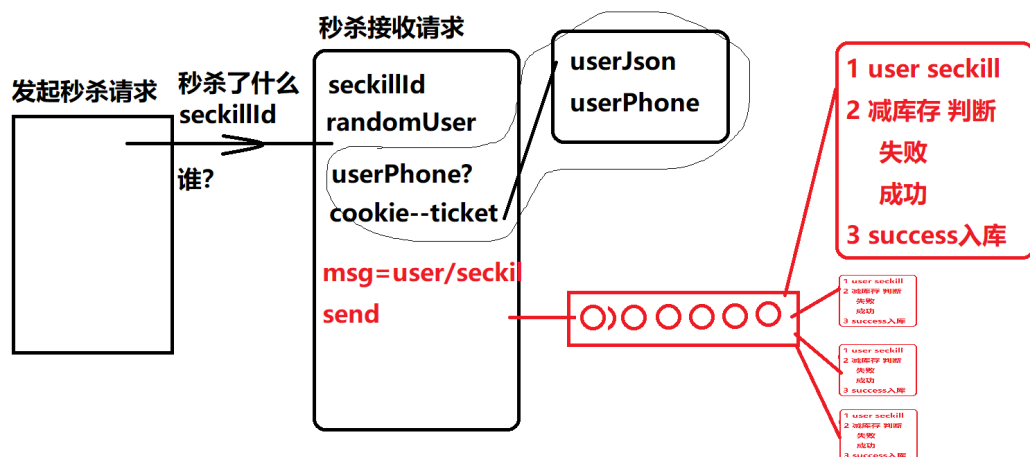
seckill-list.html 修改js代码，57行将2个拼接在<td>中的值

第二个dateStart改成dateEnd

zuul网关添加一对路由规则

## 2.2秒杀发起

#### a. 流程代码逻辑



#### b. springboot整合rabbitmq

##### i. 核心逻辑:

配置类实现连接对象，连接工厂的管理（Bean对象创建）

封装注入使用，获取channel 实现生产端发送消息，消费端监听消息

##### ii. 自定义配置:

###### 1. @Configuration

2. @ConfigurationProperties属性
3. @Bean创建对象 connctionFactory
4. 生产端，消费端注入实现发送，监听消费
5. 消费端异步监听无法实现（NIO）

### iii. 使用springboot自动配置

rabbitmqAutoConfiguration

读取属性，创建连接，RabbitTemplate实现注入生产端

使用申明式注解实现消费端逻辑

### c. 整合详细步骤

#### i. 按照配置类读取的属性结构准备属性

#rabbitmq

spring.rabbitmq.host=10.9.104.184

spring.rabbitmq.virtual-host=/

spring.rabbitmq.username=guest

spring.rabbitmq.password=guest

#### ii. 生产端消息发送处理

注入RabbitTemplate调用api发送消息

#### 1. convertAndSend：关注的是消息body作为源数据怎么发送，发送到哪

- a. exchange:已存在的交换机名称
- b. routingkey：消息路由key
- c. msg：Object 可以任意内存对象数据，一般String

#### 2. send：关注的是消息本身携带的各种属性封装

- a. exchange：交换机名称
- b. routingKey：路由key
- c. Message：封装了springboot格式的消息对象
  - i. 包含body=byte[]
  - ii. 包含MessageProperties消息属性

#### iii. 上述生产逻辑，是在已存在组件情况下可以实现发送成功

1. 想要在执行代码的一瞬间，生成需要声明的交换机，队列
2. 通过配置类，加载到容器内存对象Queue表示队列
3. 通过配置类，加载到容器内存对象Exchange表示交换机
4. 在配置类中实现这些声明的组件的对象封装使用@Bean交给容器管理

```

1 package cn.tedu.seckill.config;
2
3
4 import org.springframework.amqp.core.Binding;
5 import
6 import
7 import org.springframework.amqp.core.Queue;
8 import
9 import
10
11 /*
12 实现一个配置类编写
13 通过声明大量的对象，使得连接rabbitmq的底层
14 connction可以创建很多组件内容
15
16 和调用底层代码queueDeclare exchangeDeclare效果一样
17

```

```

18  */
19  @Configuration
20  public class QueueCompConfig {
21      //声明队列对象
22      @Bean
23      public Queue queue01() {
24          //springboot在底层通过连接
25          //调用queueDeclare name false false false
26          return new
27      }
28      @Bean
29      public Queue queue02() {
30          //springboot在底层通过连接
31          //调用queueDeclare name false false false
32          return new
33      }
34      //配置声明交换机对象
35      @Bean
36      public DirectExchange exD1() {
37          return new
38      }
39      DirectExchange("seckillD01");//默认不自动删除，默认
40      //配置声明交换机对象
41      @Bean
42      public DirectExchange exD2() {
43          return new
44      }
45      DirectExchange("seckillD02");//默认不自动删除，默认
46      //绑定关系，通过返回代码对象实现 Binding
47      @Bean
48      public Binding bind01() {
49          return BindingBuilder.bind(queue01()).
50              to(exD1()).with("seckill");
51          //seckill01使用seckill的路由绑定到了
52      }
53      @Bean
54      public Binding bind02() {
55          return BindingBuilder.bind(queue02()).
56              to(exD2()).with("haha");
57          //seckill02使用haha的路由绑定到了seckillD02
58      }
59  }

```

#### iv. 消费的监听逻辑

springboot整合之后，可以使用注解的形式，在任意一个component对象中实现消费逻辑的代码。可以结合其他对象注入将消费逻辑实现的更丰富。

##### @RabbitListener 监听注解

在方法上，添加这个注解，启动容器时能够扫描到component，也扫描到了RabbitListener

可以通过属性指定该方法对应消费者监听的队列命令。

```

1  package cn.tedu.seckill.consumer;
2
3
4  import
5  org.springframework.amqp.rabbit.annotation.Rabbi
import org.springframework.stereotype.Component;

```

```

6      import org.springframework.stereotype.Component;
7
8      @Component
9      public class SeckillConsumer2 {
10
11          //任意编辑一个方法，实现消费逻辑
12          //方法的参数就是发送到rabbitmq中的对象
13          //可以String 接收body 也可以是Message接收
14          //包含消息属性
15          @RabbitListener(queues = "seckill01")
16          public void consume(String msg){
17              /*try{}catch(){}*/
18              //执行消费逻辑
19
20              System.out.println("消费者2接收到消
21
22          }
23      }

```

#### d. 发起秒杀接口文件

后台接收	/seckill/manage/{seckillId}
请求方式	Get
请求参数	Long seckillId 路径传参
返回数据	SysResult的返回对象 Integer status 200表示秒杀成功 String msg:ok表示成功 Object data:其他数据

#### 消费端代码

```

1      package cn.tedu.seckill.consumer;
2
3
4      import cn.tedu.seckill.mapper.SecMapper;
5      import com.jt.common.pojo.Success;
6      import
7      import
8      import org.springframework.stereotype.Component;
9
10
11      import java.util.Date;
12
13      @Component
14      public class SeckillConsumer {
15          @Autowired
16          private SecMapper secMapper;
17          //任意编辑一个方法，实现消费逻辑
18          //方法的参数就是发送到rabbitmq中的对象
19          //可以String 接收body 也可以是Message接收
20          //包含消息属性
21          @RabbitListener(queues = "seckill01")
22          public void consume(String msg){
23              //接收到消息msg="1330119123/1"包含了user信息 商品id
24              //解析出来
25              Long userPhone=Long.parseLong(msg.split("/") [0]);
26              Long seckillId=Long.parseLong(msg.split("/") [1]);
27              //执行减库存逻辑 seckill表格 number=number-1
28              /*
29              UPDATE seckill SET number=number-1 WHERE
30              AND number>0 AND NOW()>start_time AND NOW()
31              */
32              int result=secMapper.decrSeckillNum(seckillId);

```

```

36         int result=secMapper.deleteSeckill(seckillId);
37         //判断减库存是否成功, result=1成功 result=0失败
38         if(result==1){
39             //成功 当前用户具备购买商品的资格
40             //将成功的信息封装数据入库记录success表格
41             Success suc=new Success();
42             suc.setCreateTime(new Date());
43             suc.setSeckillId(seckillId);
44             suc.setUserPhone(userPhone);
45             suc.setState(1);
46             secMapper.insertSuccess(suc);
47         }else{
48             //失败, result==0 减库存失败, 是因为卖完了
49             System.out.println("用户"+userPhone+"秒
50                 "库存见底, 秒杀失败");
51         }
52     }
53 }

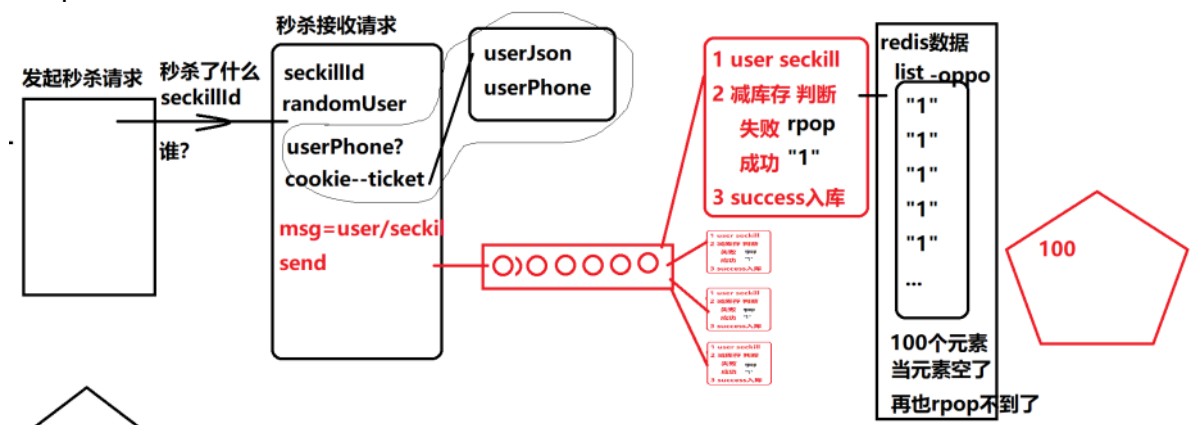
```

### 2.3秒杀减库存权限问题

消费逻辑中。会造成对数据库高并发访问。虽然只有100个人能够成功减库存  
入库记录, 但是剩下的消息, 依然要到数据库执行减库存的逻辑 (因为库存见底的  
判断  
是在数据库sql执行时实现)

应该在数据库执行update减库存之前, 就判断哪些用户的消息具备减库存的权  
限。

可以使用redis做个拦截。只有从redis中获取减库存权限的消息才能继续执行数据  
库  
的sql语句



```

1 package cn.tedu.seckill.consumer;
2
3
4 import cn.tedu.seckill.mapper.SecMapper;
5 import com.jt.common.pojo.Success;
6 import
7 import
8 import org.springframework.stereotype.Component;
9 import redis.clients.jedis.JedisCluster;
10
11 import java.util.Date;
12
13
14 @Component
15 public class SeckillConsumer {

```

```

16     @Autowired
17     private SecMapper secMapper;
18     //任意编辑一个方法，实现消费逻辑
19     //方法的参数就是发送到rabbitmq中的对象
20     //可以String 接收body 也可以是Message接收
21     //包含消息属性
22     @Autowired
23     private JedisCluster jedisCluster;
24     @RabbitListener(queues = "seckill01")
25     public void consume(String msg){
26         //接收到消息msg="1330119123/1"包含了user信息 商品id
27         //解析出来
28         Long userPhone=Long.parseLong(msg.split("/") [0]);
29         Long seckillId=Long.parseLong(msg.split("/") [1]);
30         //先连接redis 从redis中获取一个 seckill_1 rpop元素，成
31         //说明元素没有被rpopped完，具备秒杀减库存的权利
32         //手动创建这个list 在redis集群，没创建会报错
33         String listKey="seckill_"+seckillId;
34         String rpop = jedisCluster.rpop(listKey);
35         if(rpop==null){
36             //如果rpopped结果是null说明元素已经被拿完了，后续减库存
37             //都不做了
38             System.out.println("用户"+userPhone+"秒
39                 "redis库存见底，秒杀失败");
40             //解决了大量请求到达数据判断number>0时出现
41             //线程安全问题导致的超卖
42             return;
43         }
44         //执行减库存逻辑 seckill表格 number=number-1
45         /*
46         UPDATE seckill SET number=number-1 WHERE
47         AND number>0 AND NOW()>start_time AND NOW()
48         */
49         int result=secMapper.decrSeckillNum(seckillId);
50         //判断减库存是否成功，result=1成功 result=0失败
51         if(result==1){
52             //成功 当前用户具备购买商品的资格
53             //将成功的信息封装数据入库记录success表格
54             Success suc=new Success();
55             suc.setCreateTime(new Date());
56             suc.setSeckillId(seckillId);
57             suc.setUserPhone(userPhone);
58             suc.setState(1);
59             secMapper.insertSuccess(suc);
60         }else{
61             //失败，result==0 减库存失败，是因为卖完了
62             System.out.println("用户"+userPhone+"秒
63                 "库存见底，秒杀失败");
64         }
65     }
66 }

```

## 2.4展示成功者信息

### a. 接口文件

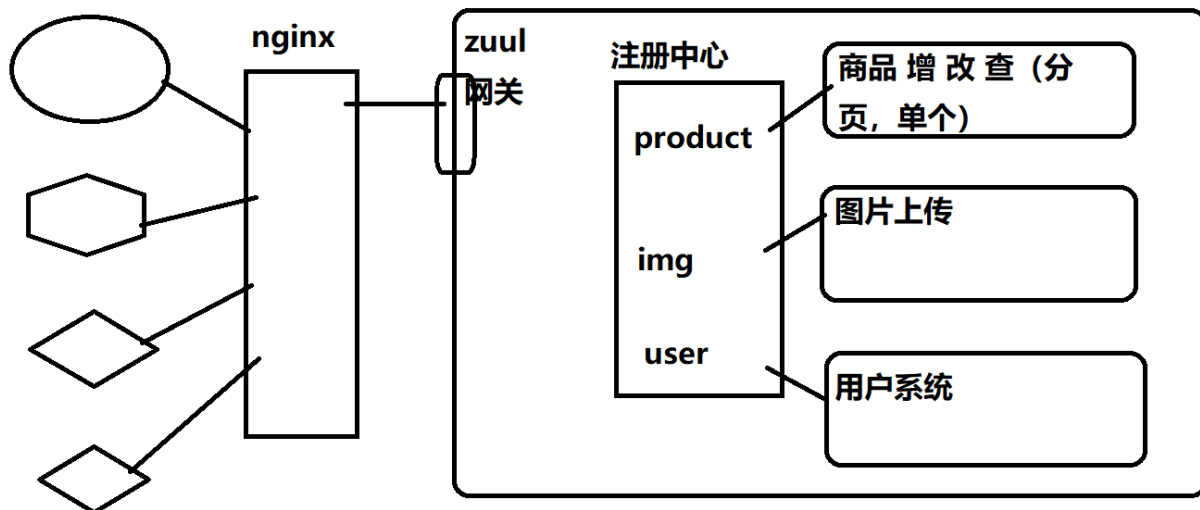
后台接收	/seckill/manage/{seckillId}/userPhone
请求方式	Get



请求参数	Long seckillId 路径传参
返回数据	List<Success>对象,将成功者信息封装返回页面

b. 前端js代码（略）

## 1.springcloud整合easymall的架构



## 2.功能

### 2.1商品系统

分页查询: limit 将参数page 和start关联  $start = (page - 1) * rows$   
以下三个功能, 在引入redis之前 业务逻辑基本就一行调用持久层  
单个商品查询  
新增商品  
修改商品

### 2.2图片上传

easymall, 简单实现上传和回显功能  
nginx访问静态文件流程详细了解  
<http://www.image.com/1.png> 访问d:/static/1.png逻辑一模一样  
唯一区别, 保存文件图片时, 使用了多级路径, 避免了一个文件夹下存储大量图片效率低下  
<http://www.image.com/1/2/3/1.png> d:/static/1/2/3/1.png  
/upload/2/3/d/3/d/3/f/3/23ljasljddfdas.png

### 2.3用户系统

注册: 用户名重复校验  
用户登录为什么不能实现: session共享。

## 3.自习任务

easymall最基本要求, 所有功能一定当天实现, 理解结构的基础上可以使用课上源码, 粘贴。

做好使用云主机的准备

有vpn能够连接启动的云主机。