# 项目的系统功能

## 项目中需要有的功能 ——互联网的架构是非常复杂的

注册

登录

首页

搜索

商品详情页

购物车

订单（正向交易、逆向交易）

支付

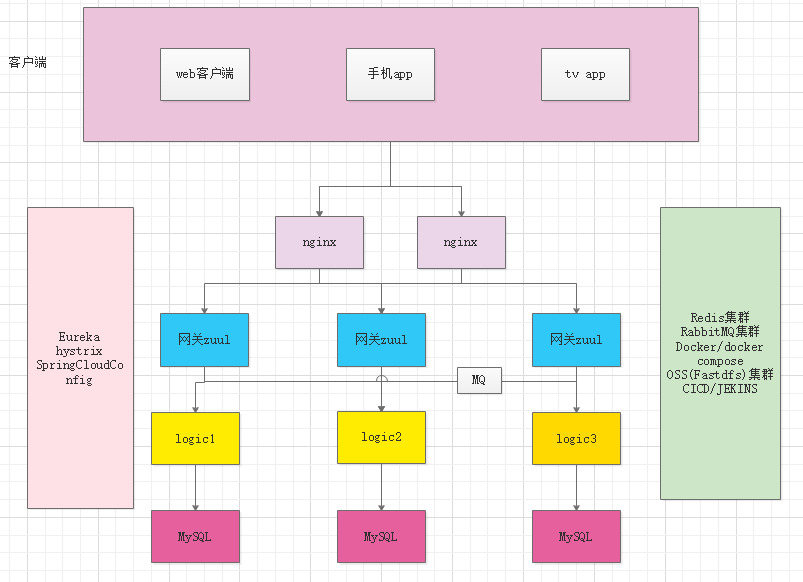
营销线：

秒杀（体会到高可用、高性能、高并发）

积分（京豆、淘金币）

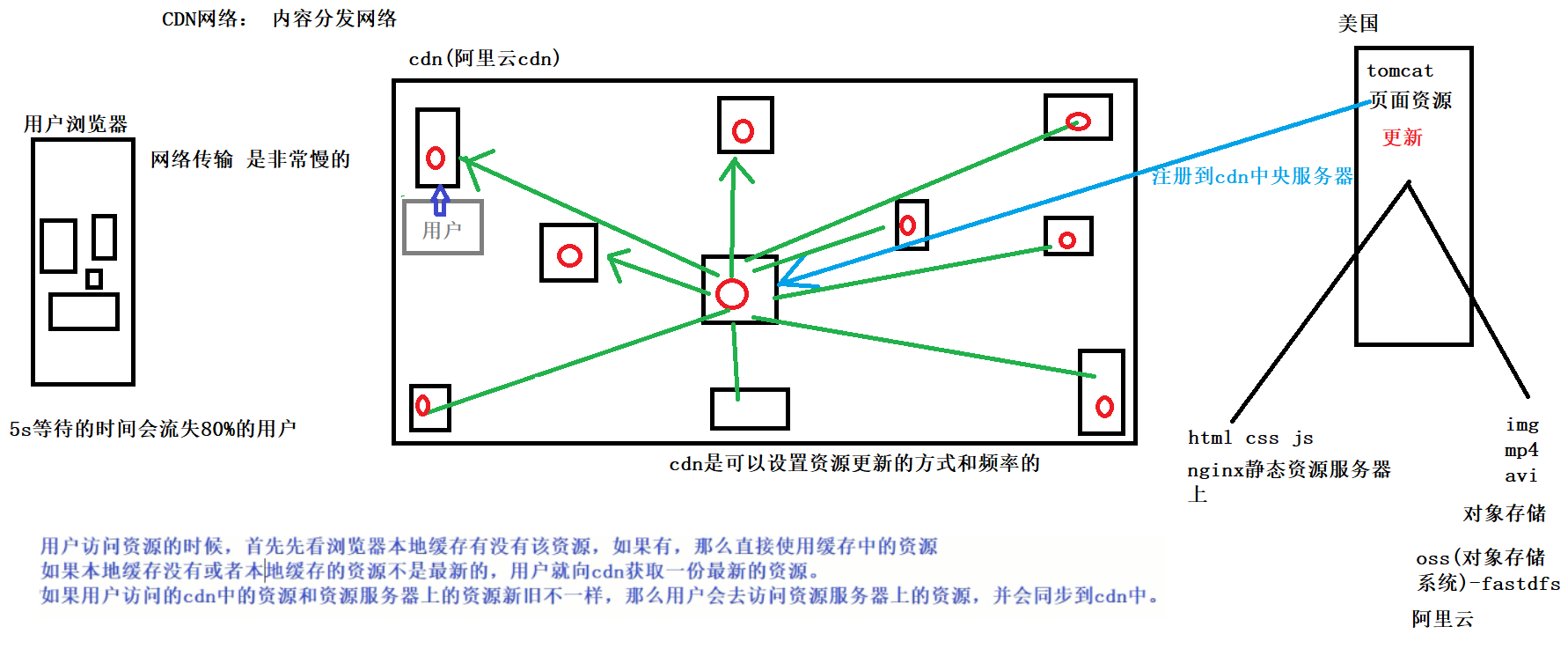
优惠券（拉新的一种手段）

## 架构图



## 前端页面的部署

### 1）使用内容分发网络



### 使用nginx对html css js做访问优化 ip和端口

（1）使用docker-compose搭建一台nginx虚拟主机

|  |
| --- |
| version: '3.1'  services:  nginx:  restart: always  image: nginx  container\_name: nginx  ports:  - 80:80  volumes:  - ./conf/nginx.conf:/etc/nginx/nginx.conf  - ./wwwroot:/usr/share/nginx/wwwroot |

1. 将静态资源存入到 wwwroot文件夹内
2. 在conf文件夹内创建nginx.conf配置文件

|  |
| --- |
| user nginx;  worker\_processes 1;  events {  worker\_connections 1024;  }  http {  include mime.types;  default\_type application/octet-stream;  sendfile on;  keepalive\_timeout 65;  server {  listen 80;  server\_name localhost;  # 解决nginx的跨域问题  add\_header Access-Control-Allow-Origin \*;  add\_header Access-Control-Allow-Headers X-Requested-With;  add\_header Access-Control-Allow-Methods GET,POST,OPTIONS;  location / {  root /usr/share/nginx/wwwroot/;  index index.html index.htm;  }  }  } |

### 使用OSS(FASTDFS)对象存储系统来存储图片、音频、视频

# 项目的具体实现——注册模块

## 注册模块的搭建

## Thymeleaf的介绍

目前用它来解决我们前端展示数据的问题，使用thymeleaf可以通过模板+数据的方式==得到页面，这个页面在有数据时展示数据，在没数据时，只展示模板。

## Thymeleaf的使用

1. 要在页面上的html标签中使用thymeleaf的标签库

|  |
| --- |
| <**html xmlns:th="http://www.thymeleaf.org"**> |

1. 使用Thymeleaf表达式获得后端的数据

一定要注意： thymeleaf是必须写在标签内的

1. username:zhangsan ${username}

|  |
| --- |
| <**a href="# "**>联系<**span th:text="${username}"**>我们</**span**></**a**> **这样可以** <**a href="# "**>联系${username}</**a**> **这样不行** |

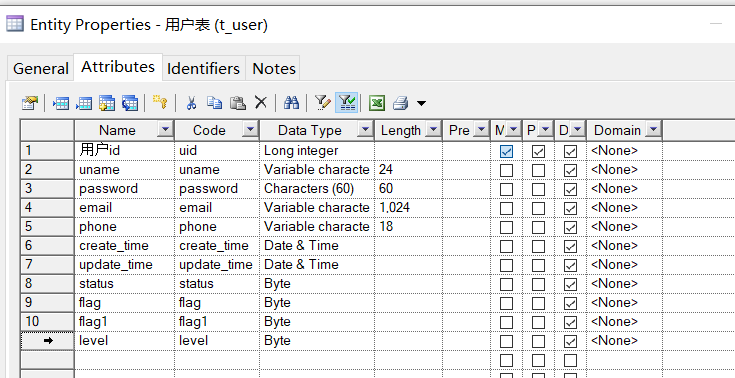
1. 获得对象中的属性
2. 如何执行if判断
3. 如何执行循环遍历

|  |
| --- |
| @RequestMapping(**"/demo"**) **public** String demo(Model model){  Student stu1 = **new** Student(**"zhangsan"**,20);  Student stu2 = **new** Student(**"lisi"**,22);   List<Student> stus = **new** ArrayList<>();  stus.add(stu1);  stus.add(stu2);   model.addAttribute(**"stus"**,stus);   **return "demo"**; } |

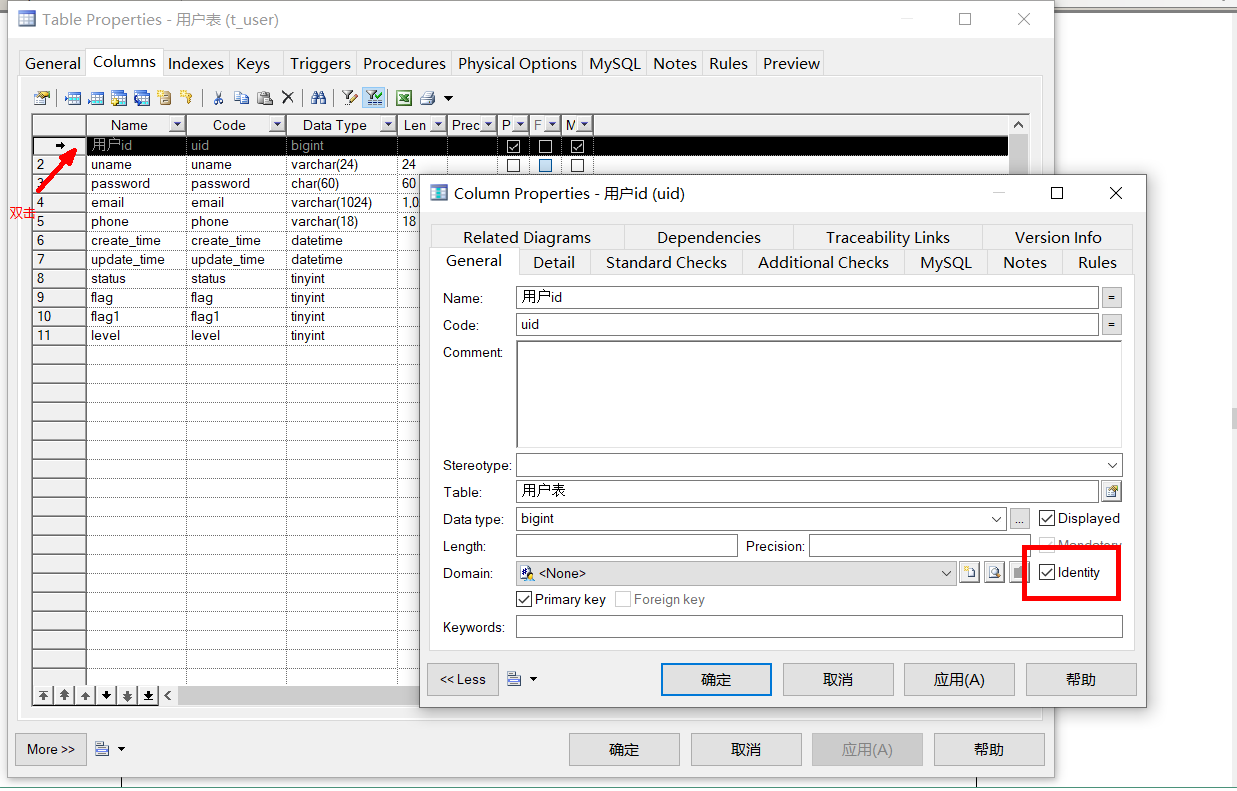
|  |
| --- |
| <**table border="1px" width="100%"**>   <**tr**>  <**td**>姓名</**td**>  <**td**>年龄</**td**>  </**tr**>  <**tr th:each="stu : ${stus}" th:if="${stu.age>20}"**> if表示只有age大于20才会显示  <**td th:text="${stu.name}"**>姓名</**td**>  <**td th:text="${stu.age}"**>年龄</**td**>  </**tr**>  </**table**> |

# 数据库的设计

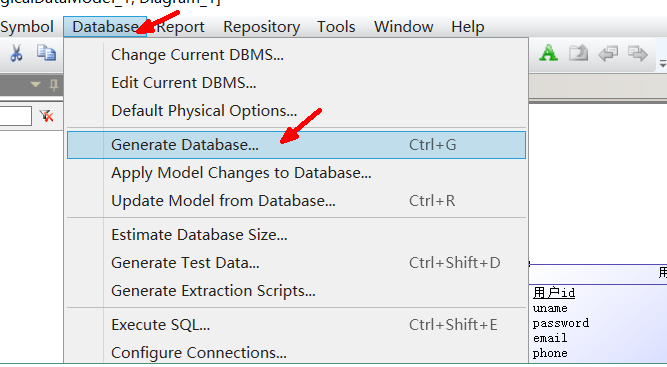
## 1.设计用户表的逻辑模型



## 转换成物理模型后设置主键自增

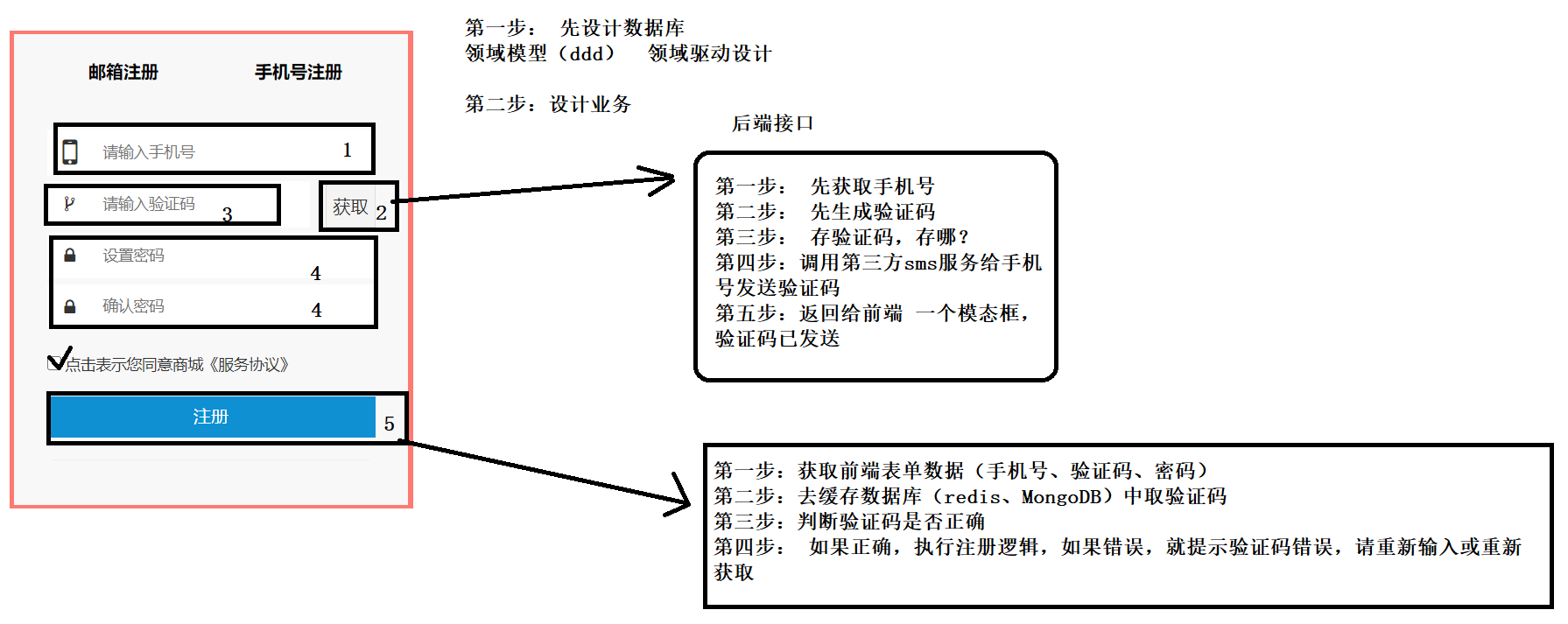


生成好sql语句导入到数据库中



# 手机号注册

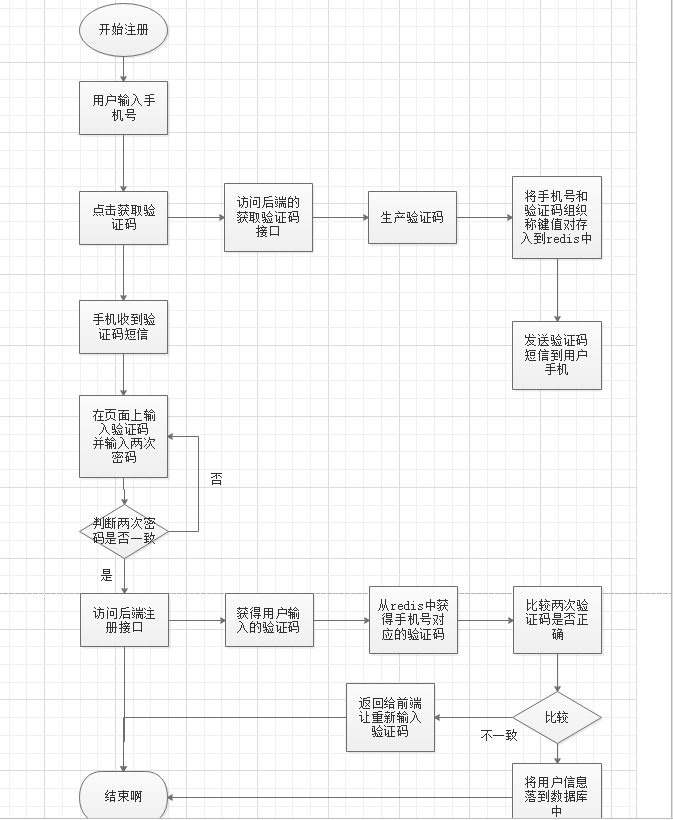
## 流程梳理



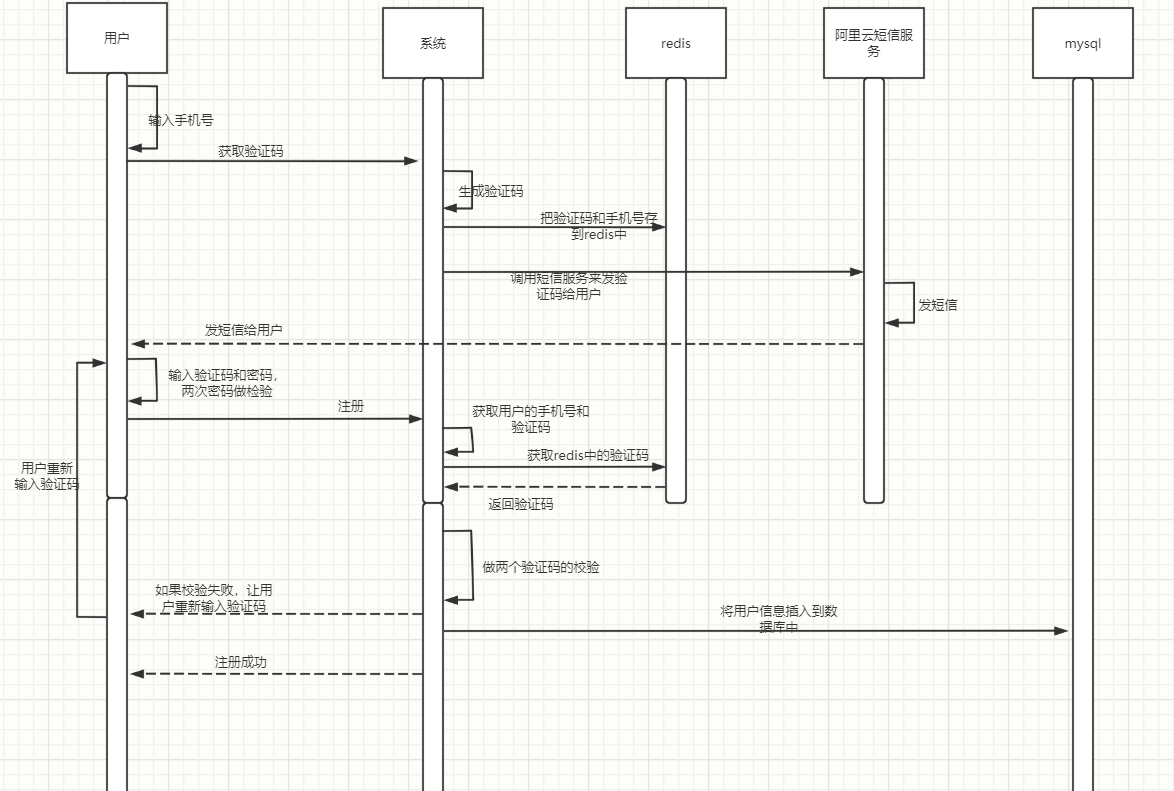
## 代码实现

|  |
| --- |
| **package** com.qf.my.shop.regist.service.impl;  **import** com.qf.common.constant.RedisConstant; **import** com.qf.common.dto.ResultBean; **import** com.qf.common.util.RestHttpUtil; **import** com.qf.my.shop.regist.service.IRegistService; **import** com.qf.my.shop.regist.util.StringUtil; **import** org.springframework.beans.factory.annotation.Autowired; **import** org.springframework.http.HttpHeaders; **import** org.springframework.stereotype.Service; **import** org.springframework.util.LinkedMultiValueMap; **import** org.springframework.util.MultiValueMap; **import** org.springframework.web.client.RestTemplate;  @Service **public class** RegistServiceImpl **implements** IRegistService {   @Autowired  **private** RestTemplate **restTemplate**;   @Override  **public** ResultBean getSmsCode(String phone) {  String smsCode = StringUtil.*getSmsCode*();  *//=========把验证码存到redis中==================  //regist:手机号===验证码  //组织redis中的键* String pre = RedisConstant.***REGIST\_PRE***;  String redisKey = StringUtil.*getRedisKey*(pre, phone);   *//封装请求头* HttpHeaders headers = **new** HttpHeaders();  MultiValueMap<String, Object> map = **new** LinkedMultiValueMap<>();  map.add(**"key"**, redisKey);  map.add(**"value"**, smsCode);   String url = **"http://CACHE-DATA-ACCESS/cache/data/insert"**;   RestHttpUtil.*requestByPost*(headers, map, url, **restTemplate**);    *//=========使用短信服务来发送短信==================* HttpHeaders headers1 = **new** HttpHeaders();  MultiValueMap<String, Object> map1 = **new** LinkedMultiValueMap<>();  map1.add(**"phone"**, phone);  map1.add(**"code"**, smsCode);  String url1 = **"http://SMS-SERVER/sms/send"**;  ResultBean resultBean = RestHttpUtil.*requestByPost*(headers1, map1, url1, **restTemplate**);  **return** resultBean;  } } |

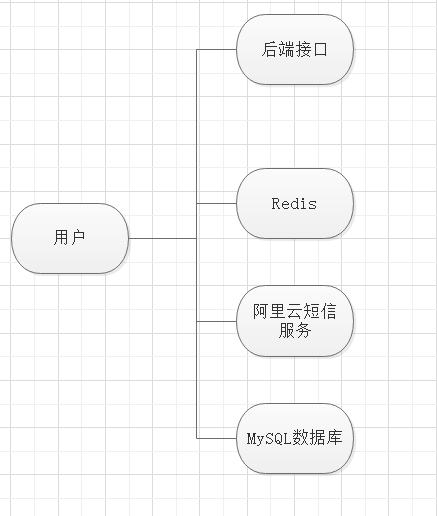
## 画流程图



## 画时序图



## 画用例图



# RestTemplate的使用

## getForObject

|  |
| --- |
| String username = restTemplate.getForObject("http://localhost:8080/getNameById?id={1}&name={2}",String.class,new Object[]{id,name});  Map<String,Object> params = new HashMap<>();  params.put("id",id);  params.put("name",name);  User user = restTemplate.getForObject("http://localhost:8080/getUserById?id={id}&name={name}",User.class,params);  UriComponents uriComponents = UriComponentsBuilder.fromUriString("http://localhost:8080/getUserByname?name={name}").build().expand(name).encode();  URI uri = uriComponents.toUri();  String response = restTemplate.getForObject(uri,String.class); |

## postForObject

|  |
| --- |
| HttpHeaders headers = new HttpHeaders();  MultiValueMap<String, Object> parammap = new LinkedMultiValueMap<>();  parammap.add("privilege",privilege);    HttpEntity<Map> entity = new HttpEntity<>(parammap,headers);    User user = restTemplate.postForObject(userService+"/queryUserByPrivilege",entity,User.class);  List<String> menulist = restTemplate.postForObject("http://localhost:8080/queryLoginPrivilegeByUser",user,List.class); |

# 阿里云的短信服务

1. 登录阿里云短信服务控制台
2. 登录用户中心进行充值



1. 添加签名

在国内消息中添加签名



测试用的话，签名就是ABC商城

1. 添加模板



# 注册时的数据落库

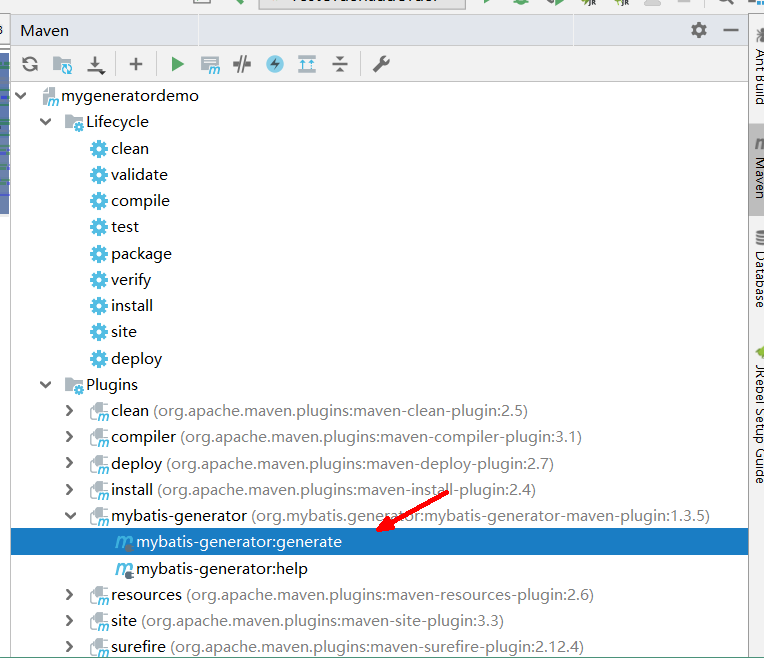
## 需要有相应的映射文件

这些组件Jpa、MybatisPlus、逆向工程都是可以使用的

逆向工程的配置

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* **<!DOCTYPE generatorConfiguration  PUBLIC "-//mybatis.org//DTD MyBatis Generator Configuration 1.0//EN"  "http://mybatis.org/dtd/mybatis-generator-config\_1\_0.dtd"*>*** <**generatorConfiguration**>  *<!--指定了驱动jar包的位置，这个是针对下载Jar包的方式，因为用了maven所以这个就用不上了-->  <!-- <classPathEntry location="D:/mvn\_repository\_new/mysql/mysql-connector-java/5.1.45/mysql-connector-java-5.1.45.jar"/>-->* <**context id="DB2Tables" targetRuntime="MyBatis3"**>  <**commentGenerator**>  <**property name="suppressDate" value="false"**/>  *<!-- 是否去除自动生成的注释 true：是 ： false:否 -->* <**property name="suppressAllComments" value="true"**/>  </**commentGenerator**>   *<!--数据库链接URL，用户名、密码 -->* <**jdbcConnection driverClass="com.mysql.jdbc.Driver"  connectionURL="jdbc:mysql://localhost:3306/my-shop-v3?serverTimezone=Asia/Shanghai" userId="root"  password="123456"**>  *<!--<property name="serverTimezone" value="UTC"/>-->* <**property name="nullCatalogMeansCurrent" value="true"**/>  </**jdbcConnection**>  *<!--指定生成entity实体类的具体位置-->* <**javaModelGenerator targetPackage="com.qf.common.entity" targetProject="./src/main/java"**>  <**property name="enableSubPackages" value="true"**/>  <**property name="trimStrings" value="true"**/>  </**javaModelGenerator**>  *<!--指定生成mybatis映射xml文件的包名和位置-->* <**sqlMapGenerator targetPackage="mapper" targetProject="./src/main/resources"**>  <**property name="enableSubPackages" value="true"**/>  </**sqlMapGenerator**>  *<!--指定生成mapper接口的具体位置-->* <**javaClientGenerator targetPackage="com.qf.common.mapper" targetProject="./src/main/java" type="XMLMAPPER"**>  <**property name="enableSubPackages" value="true"**/>  </**javaClientGenerator**>   <**table tableName="t\_user" enableCountByExample="false" enableUpdateByExample="false" enableDeleteByExample="false" enableSelectByExample="false" selectByExampleQueryId="false"**/>     </**context**> </**generatorConfiguration**> |

设置完以后，点击插件 来生成即可：



## 引入相关的依赖

|  |
| --- |
| *<!-- https://mvnrepository.com/artifact/org.mybatis.spring.boot/mybatis-spring-boot-starter -->* <**dependency**>  <**groupId**>org.mybatis.spring.boot</**groupId**>  <**artifactId**>mybatis-spring-boot-starter</**artifactId**>  <**version**>2.1.3</**version**> </**dependency**>   <**dependency**>  <**groupId**>mysql</**groupId**>  <**artifactId**>mysql-connector-java</**artifactId**> </**dependency**>  <**dependency**>  <**groupId**>com.alibaba</**groupId**>  <**artifactId**>druid-spring-boot-starter</**artifactId**>  <**version**>1.1.10</**version**> </**dependency**> |

## 3.配置文件

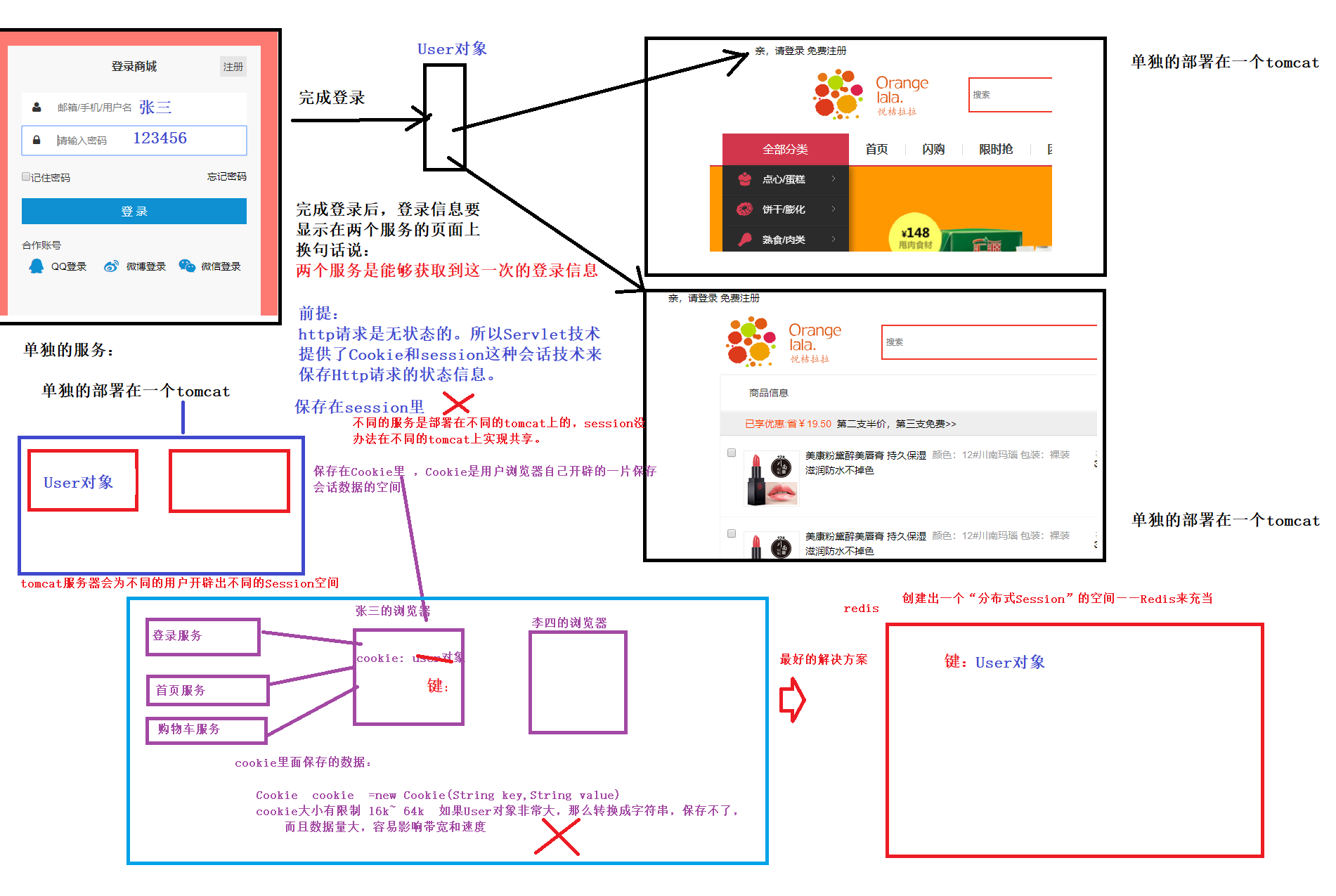
|  |
| --- |
| **server**:  **port**: 8201 **spring**:  **application**:  **name**: regist-server  *# 配置连接池Druid* **datasource**:  **driver-class-name**: com.mysql.cj.jdbc.Driver  **url**: jdbc:mysql://localhost:3306/my-shop-v3?serverTimezone=Asia/Shanghai  **username**: root  **password**: 123456  **type**: com.alibaba.druid.pool.DruidDataSource **eureka**:  **client**:  **service-url**:  **defaultZone**: http://localhost:9201/eureka/ *# 指明mapper映射文件的位置* **mybatis**:  **mapper-locations**: classpath:mapper/\*.xml |

## 4.业务代码

|  |
| --- |
| **package** com.qf.my.shop.regist.controller;  **import** com.qf.common.constant.RedisConstant; **import** com.qf.common.dto.ResultBean; **import** com.qf.common.mapper.TUserMapper; **import** com.qf.my.shop.regist.common.vo.TUserVO; **import** com.qf.my.shop.regist.service.IRegistService; **import** com.qf.my.shop.regist.util.StringUtil; **import** org.springframework.beans.factory.annotation.Autowired; **import** org.springframework.stereotype.Controller; **import** org.springframework.ui.Model; **import** org.springframework.web.bind.annotation.GetMapping; **import** org.springframework.web.bind.annotation.RequestMapping; **import** org.springframework.web.bind.annotation.ResponseBody; **import** org.springframework.web.client.RestTemplate;  **import** java.util.HashMap; **import** java.util.Map;  @Controller @RequestMapping(**"/regist"**) **public class** RegistController {   @Autowired  **private** IRegistService **registService**;   @Autowired  **private** RestTemplate **restTemplate**;   @Autowired  **private** TUserMapper **mapper**;   @RequestMapping(**"/show"**)  **public** String show(Model model) {  model.addAttribute(**"username"**, **"张三"**);  **return "register"**;  }   @GetMapping(**"/sms/code"**)  @ResponseBody  **public** ResultBean getSmsCode(String phone) {  **return registService**.getSmsCode(phone);  }   @RequestMapping(**"/action"**)  **public** String registAction(TUserVO user){   *//1.获得用户手机号对应的验证码——从redis——要去访问数据访问层* String phone = user.getPhone();  *//1-1 组织redis的键* String redisKey = StringUtil.*getRedisKey*(RedisConstant.***REGIST\_PRE***, phone);   *//2.redis中的验证码与user中的验证码进行比对* String url = **"http://CACHE-DATA-ACCESS/cache/data/get?key={key}"**;  Map<String,Object> map = **new** HashMap<>();  map.put(**"key"**,redisKey);  ResultBean resultBean = **restTemplate**.getForObject(url, ResultBean.**class**, map); *// System.out.println(resultBean);  //3.根据不同的结果返回不同的内容* String code\_cache = (String) resultBean.getObject();  **if**(user.getCode().equals(code\_cache)){  *//验证码相同 // 把数据存到数据库中  //第一种做法： 创建TUser对象，并进行封装  //第二种做法： 将TUserVO设计成TUser的子类* **mapper**.insert(user); *// return "redict:http://localhost:8202/login/show";* **return "login"**;  }  *//验证码不相同  //回到注册页面* **return "register"**;  }  } |

# 登录业务

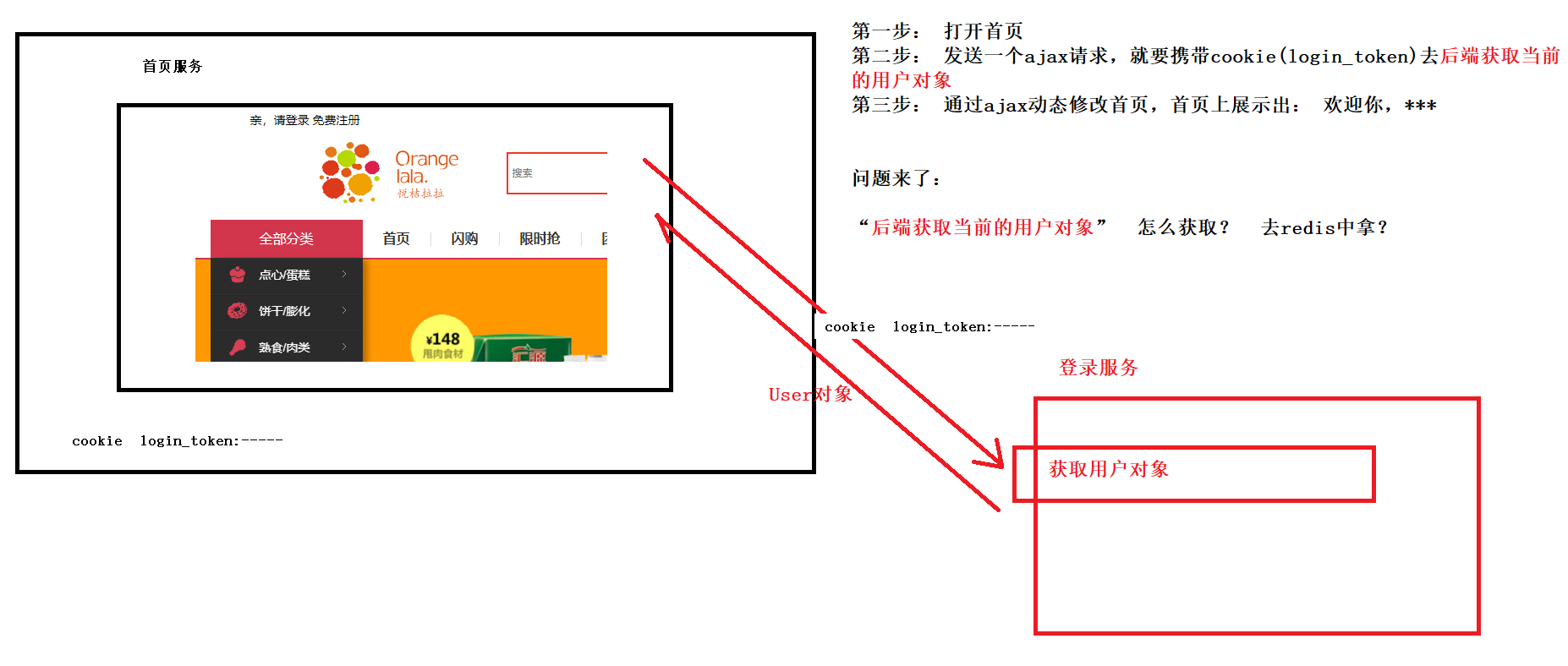
## 使用分布式session保存登录状态



## 业务代码

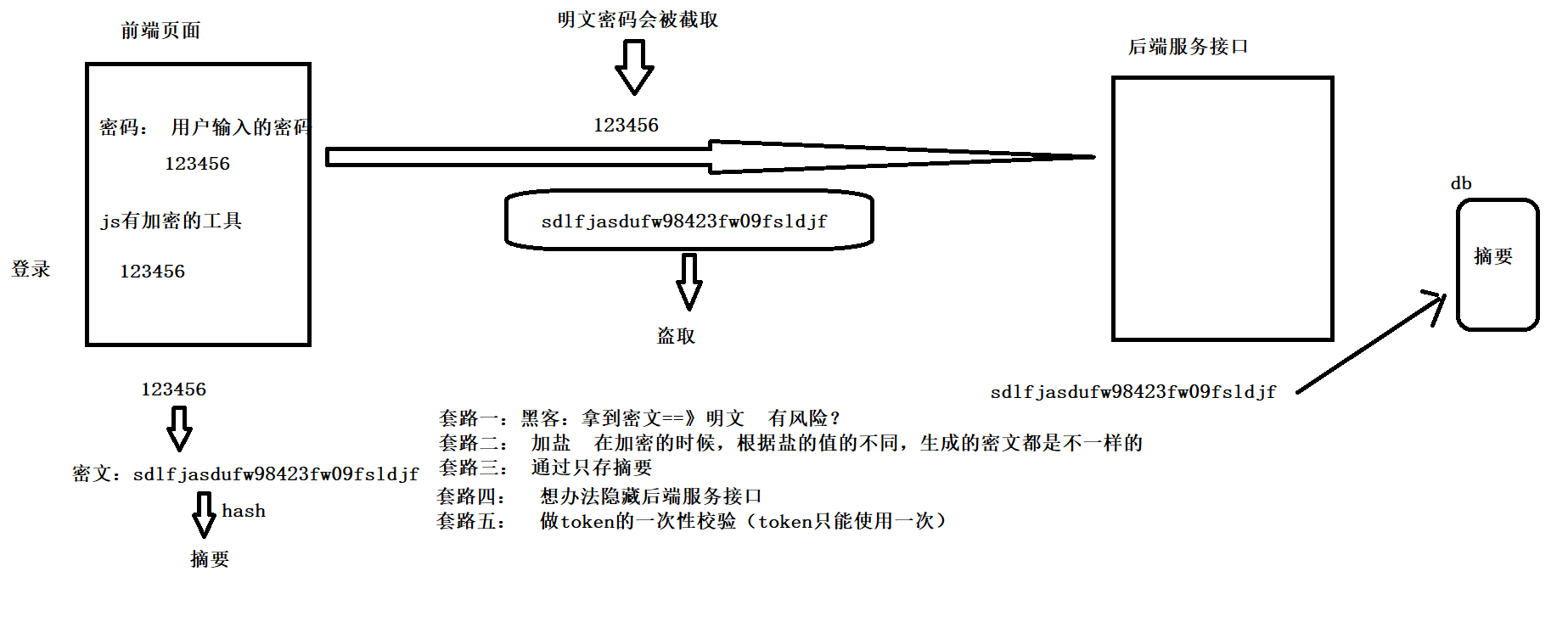
|  |
| --- |
| **package** com.qf.my.shop.login.controller;  **import** com.qf.common.constant.CookieConstant; **import** com.qf.common.constant.RedisConstant; **import** com.qf.common.dto.ResultBean; **import** com.qf.common.entity.TUser; **import** com.qf.common.mapper.TUserMapper; **import** com.qf.common.util.RestHttpUtil; **import** com.qf.common.util.StringUtil; **import** com.qf.common.vo.TUserVO; **import** org.springframework.beans.factory.annotation.Autowired; **import** org.springframework.http.HttpHeaders; **import** org.springframework.stereotype.Controller; **import** org.springframework.util.LinkedMultiValueMap; **import** org.springframework.util.MultiValueMap; **import** org.springframework.web.bind.annotation.RequestMapping; **import** org.springframework.web.bind.annotation.ResponseBody; **import** org.springframework.web.client.RestTemplate;  **import** javax.servlet.http.Cookie; **import** javax.servlet.http.HttpServletResponse;  @Controller @RequestMapping(**"/login"**) **public class** LoginController {   @Autowired  **private** TUserMapper **userMapper**;   @Autowired  **private** RestTemplate **restTemplate**;   @RequestMapping(**"/show"**)  **public** String show(){  **return "login"**;  }   @RequestMapping(**"/action"**)  @ResponseBody  **public** ResultBean loginAction(TUserVO user, HttpServletResponse response){  *//user phone password(明文) select \* from t\_user where phone=? 能让她闲着一定要让她闲着  //1.登录验证* TUser tUser = **userMapper**.selectByPhone(user.getPhone());  **if**(tUser!=**null**&&tUser.getPassword().equals(user.getPassword())){  *//登录成功  //2.生成redis的键：  // 2-1 拿token* String token = StringUtil.*crtToken*();  *//2-2 组织redis的键* String redisKey = StringUtil.*getRedisKey*(RedisConstant.***LOGIN\_TOKEN\_PRE***, token);   *//3.把键值对存入到redis中* tUser.setPassword(**""**);*//保证密码安全* HttpHeaders headers = **new** HttpHeaders();  MultiValueMap<String, Object> map = **new** LinkedMultiValueMap<>();  map.add(**"key"**,redisKey);  map.add(**"value"**,tUser);  String url = **"http://CACHE-DATA-ACCESS/cache/data/insert"**;  RestHttpUtil.*requestByPost*(headers, map, url, **restTemplate**);   *//4.存Cookie : ResponseHeader: Cookie : login\_token=lskjdfljsdfwerwer* Cookie cookie = **new** Cookie(CookieConstant.***LOGIN\_TOKEN***,token);  cookie.setMaxAge(7\*24\*24\*60);*//cookie的有效期* cookie.setPath(**"/"**); *// cookie.setDomain("qf.com");// 指定一级域名，对应的二级域名都能被携带www.qf.com admin.qf.com login.qf.com // cookie.setHttpOnly(true);//只有向后端发http请求，此cookie才会被携带* response.addCookie(cookie);  **return** ResultBean.*success*(**"登录成功"**,**null**);  }   *//登录失败* **return** ResultBean.*error*(**"用户名或密码错误"**);     }    } |

# 登录服务之获取用户对象



# 明文密码传输的解决方案

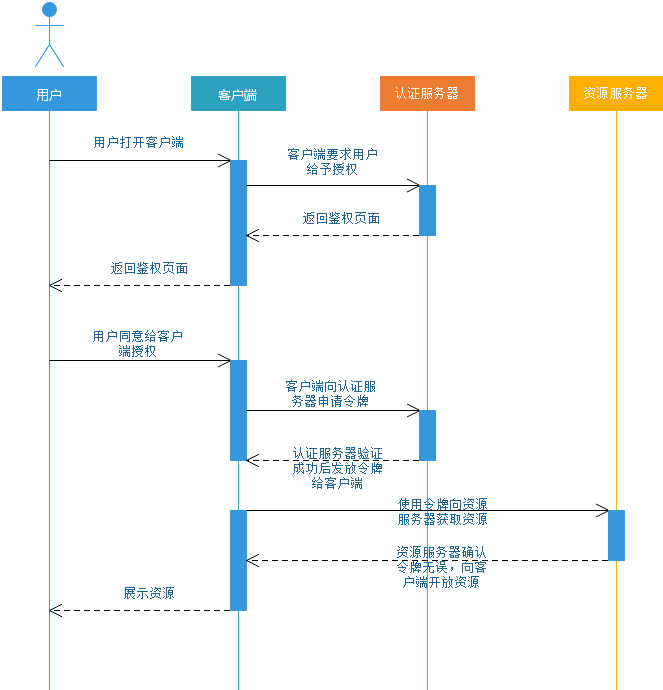
## 1.各种方案介绍



## 2.使用SpringSecurity做MD5加密

|  |
| --- |
| @Autowired **private** BCryptPasswordEncoder **encoder**;  @Test **public void** test2(){  //匹配  System.***out***.println(**encoder**.matches(**"123456"**, **"$2a$10$PJFVx7jUkrhWftsCnj5Knu./6XAwDLRRRA3t.mtUddBndPTnyB7N6"**));  System.***out***.println(**encoder**.matches(**"123456"**,**"$2a$10$6E1C.77Tg7bytqEKl9L/Hu4kP2P7ke1rFjtke1UBYHFx0d.yGYh3G"**)); }  @Test **public void** test1(){   *//加密* String pass = **"123456"**;  String encode = **encoder**.encode(pass);  System.***out***.println(encode);  *//$2a$10$PJFVx7jUkrhWftsCnj5Knu./6XAwDLRRRA3t.mtUddBndPTnyB7N6* System.***out***.println(**"$2a$10$6E1C.77Tg7bytqEKl9L/Hu4kP2P7ke1rFjtke1UBYHFx0d.yGYh3G"**.length());  } |

# 第三方登录的流程



# 首页的无限级分类

## 冷热数据分离

数据库中的一条记录，有很多字段，这些字段是有冷热之分的，比如商品的名称、价格、图片、分类信息是经常要被访问的，也称为热数据，但商品描述这个字段是相对来说比较少被访问的，也称为冷数据，因此在设计数据库表的时候，要对冷热数据进行分离。

## 无限极分类



## 抽取公共的base



## 前端页面显示无限极分类

|  |
| --- |
| *<!--侧边导航 -->* <**div id="nav" class="navfull"**>  <**div class="area clearfix"**>  <**div class="category-content" id="guide\_2"**>   <**div class="category"**>  <**ul class="category-list" id="js\_climit\_li"**>  <**li class="appliance js\_toggle relative first" th:each="type:${types}" th:if="${type.pid==0}"**>  <**div class="category-info"**>  <**h3 class="category-name b-category-name"**><**i**><**img src="images/cake.png"**></**i**><**a class="ml-22" th:title="${type.cname}"**><**span th:text="${type.cname}"**>测试一级分类</**span**></**a**></**h3**>  <**em**>**&gt;**</**em**></**div**>  <**div class="menu-item menu-in top"**>  <**div class="area-in"**>  <**div class="area-bg"**>  <**div class="menu-srot"**>  <**div class="sort-side"**>  <**dl class="dl-sort" th:each="secondType:${types}" th:if="${secondType.pid==type.id}"**>  <**dt**><**span th:title="${secondType.cname}"**><**span th:text="${secondType.cname}"**>测试二级分类</**span**></**span**></**dt**>  <**dd th:each="thirdType:${types}" th:if="${thirdType.pid==secondType.id}"**>  <**a th:title="${thirdType.cname}" href="#"**><**span th:text="${thirdType.cname}"**>测试三级分类</**span**></**a**></**dd**>   </**dl**>   </**div**>  <**div class="brand-side"**>  <**dl class="dl-sort"**><**dt**><**span**>实力商家</**span**></**dt**>  <**dd**><**a rel="nofollow" title="呵官方旗舰店" target="\_blank" href="#"** ><**span class="red"** >呵官方旗舰店</**span**></**a**></**dd**>  <**dd**><**a rel="nofollow" title="格瑞旗舰店" target="\_blank" href="#"** ><**span** >格瑞旗舰店</**span**></**a**></**dd**>  <**dd**><**a rel="nofollow" title="飞彦大厂直供" target="\_blank" href="#"** ><**span class="red"** >飞彦大厂直供</**span**></**a**></**dd**>  <**dd**><**a rel="nofollow" title="红e·艾菲妮" target="\_blank" href="#"** ><**span** >红e·艾菲妮</**span**></**a**></**dd**>  <**dd**><**a rel="nofollow" title="本真旗舰店" target="\_blank" href="#"** ><**span class="red"** >本真旗舰店</**span**></**a**></**dd**>  <**dd**><**a rel="nofollow" title="杭派女装批发网" target="\_blank" href="#"** ><**span class="red"** >杭派女装批发网</**span**></**a**></**dd**>  </**dl**>  </**div**>  </**div**>  </**div**>  </**div**>  </**div**>  <**b class="arrow"**></**b**>  </**li**>   </**ul**>  </**div**>  </**div**>   </**div**> </**div**> |

# 全文搜索——Solr

## solr介绍

Solr和es相同，也是建立在Lucene全文搜索引擎之上的一套搜索的框架。和es的最大区别就是：solr适合检索更新不是特别频繁的数据，es更加适合对实时数据的检索，比如：检索日志。

同样的，solr也是一个nosql数据库，存放键值对。

## solr的安装

使用docker-compose来安装即可。

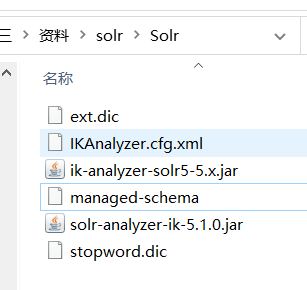
1. 创建docker-compose.yml文件

|  |
| --- |
| version: '3.1'  services:  solr:  build: ikanalyzer  restart: always  container\_name: solr  ports:  - 8983:8983  volumes:  - ./solrdata:/opt/solrdata |

1. 在docker-compose.yml同级目录下创建文件夹： ikanalyzer
2. 进入到ikanalyzer文件夹内，创建Dockerfile文件

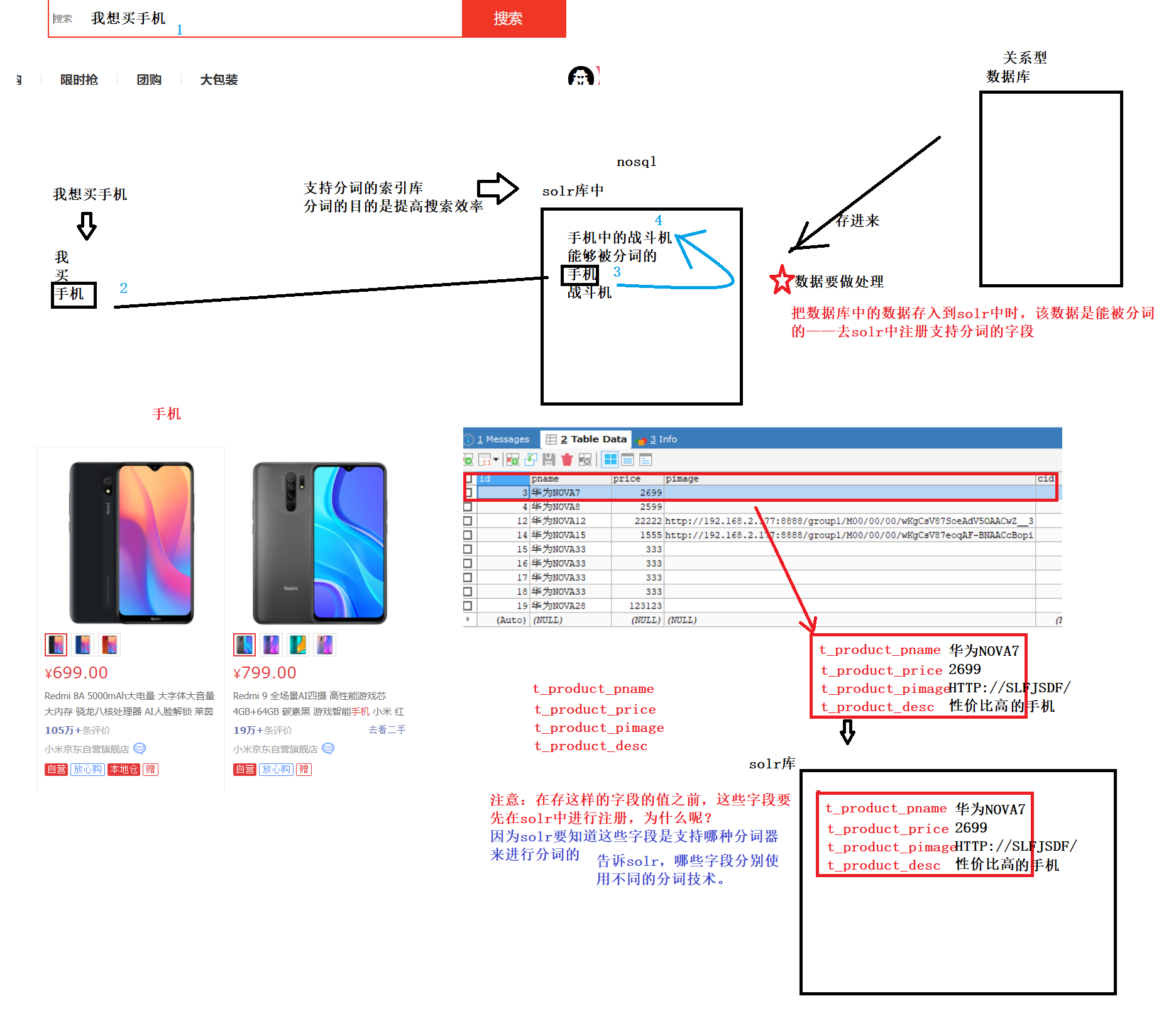
|  |
| --- |
| FROM solr:7.1.0  # 创建 Core  WORKDIR /opt/solr/server/solr  RUN mkdir ik\_core  WORKDIR /opt/solr/server/solr/ik\_core  RUN echo 'name=ik\_core' > core.properties  RUN mkdir data  RUN cp -r ../configsets/sample\_techproducts\_configs/conf/ .  # 安装中文分词  WORKDIR /opt/solr/server/solr-webapp/webapp/WEB-INF/lib  ADD ik-analyzer-solr5-5.x.jar .  ADD solr-analyzer-ik-5.1.0.jar .  WORKDIR /opt/solr/server/solr-webapp/webapp/WEB-INF  ADD ext.dic .  ADD stopword.dic .  ADD IKAnalyzer.cfg.xml .  # 增加分词配置  COPY managed-schema /opt/solr/server/solr/ik\_core/conf  WORKDIR /opt/solr |

1. 把准备好的ik的相关文件放入到当前目录下（ikanalyzer）



1. 通过docker-compose up -d来启动容器即可

## solr中的分词相关技术



## 向solr注册字段、复制域字段

|  |
| --- |
| <field name="t\_product\_pname" type="text\_ik" indexed="true" stored="true"/>  <field name="t\_product\_price" type="pfloat" indexed="true" stored="true"/>  <field name="t\_product\_pimage" type="text\_general" indexed="true" stored="true"/>  <field name="t\_product\_pdesc" type="text\_ik" indexed="true" stored="true"/>  <field name="tb\_item\_keywords" type="text\_ik" indexed="true" stored="false" multiValued="true"/>  <copyField source="t\_product\_pname" dest="tb\_item\_keywords"/>  <copyField source="t\_product\_pdesc" dest="tb\_item\_keywords"/> |

## 从数据库中获取数据封装成搜索数据并存入到solr中

配置文件

|  |
| --- |
| **server**:  **port**: 8204 **spring**:  **application**:  **name**: search-server  **datasource**:  **driver-class-name**: com.mysql.cj.jdbc.Driver  **url**: jdbc:mysql://localhost:3306/my-shop-v3?serverTimezone=Asia/Shanghai  **username**: root  **password**: 123456  **type**: com.alibaba.druid.pool.DruidDataSource  *#solr:* **data**:  **solr**:  **host**: http://192.168.2.191:8983/solr/ik\_core *# 指明mapper映射文件的位置* **mybatis**:  **mapper-locations**: classpath:mapper/\*.xml **eureka**:  **client**:  **service-url**:  **defaultZone**: http://localhost:9201/eureka/ |

业务代码

|  |
| --- |
| **package** com.qf.my.shop.search.server.controller;  **import** com.qf.common.dto.ResultBean; **import** com.qf.my.shop.search.server.entity.SearchBean; **import** com.qf.my.shop.search.server.mapper.SearchBeanMapper; **import** org.apache.solr.client.solrj.SolrClient; **import** org.apache.solr.client.solrj.SolrServerException; **import** org.apache.solr.common.SolrInputDocument; **import** org.springframework.beans.factory.annotation.Autowired; **import** org.springframework.stereotype.Controller; **import** org.springframework.web.bind.annotation.RequestMapping; **import** org.springframework.web.bind.annotation.ResponseBody;  **import** java.io.IOException; **import** java.util.ArrayList; **import** java.util.List;  @Controller @RequestMapping(**"/search/admin"**) **public class** InitDataToSolrController {  @Autowired  **private** SearchBeanMapper **mapper**;   @Autowired  **private** SolrClient **solrClient**;   @RequestMapping(**"/initDataToSearch"**)  @ResponseBody  **public** ResultBean initData(){   *//从数据库中封装数据* List<SearchBean> searchBeans = **mapper**.selectAll();   *//向solr索引库中插入数据* List<SolrInputDocument> docs = **new** ArrayList<>(searchBeans.size());  **for** (SearchBean searchBean : searchBeans) {  SolrInputDocument doc = **new** SolrInputDocument();  doc.setField(**"id"**,searchBean.getId());  doc.setField(**"t\_product\_pname"**,searchBean.getTProductPname());  doc.setField(**"t\_product\_price"**,searchBean.getTProductPrice());  doc.setField(**"t\_product\_pimage"**,searchBean.getTProductPimage());  doc.setField(**"t\_product\_pdesc"**,searchBean.getTProductPdesc());   docs.add(doc);  }    **try** {  **solrClient**.add(docs);  **solrClient**.commit();*//提交* **return** ResultBean.*success*(**"添加数据成功"**,**null**);  } **catch** (SolrServerException e) {  e.printStackTrace();  } **catch** (IOException e) {  e.printStackTrace();  }    **return** ResultBean.*error*();  }  } |

## 搜solr库中检索数据并显示在前端页面上

业务代码：

|  |
| --- |
| **package** com.qf.my.shop.search.server.controller;  **import** com.qf.common.dto.ResultBean; **import** com.qf.my.shop.search.server.entity.SearchBean; **import** org.apache.solr.client.solrj.SolrClient; **import** org.apache.solr.client.solrj.SolrQuery; **import** org.apache.solr.client.solrj.SolrServerException; **import** org.apache.solr.client.solrj.response.QueryResponse; **import** org.apache.solr.common.SolrDocument; **import** org.apache.solr.common.SolrDocumentList; **import** org.springframework.beans.factory.annotation.Autowired; **import** org.springframework.stereotype.Controller; **import** org.springframework.web.bind.annotation.PathVariable; **import** org.springframework.web.bind.annotation.RequestMapping; **import** org.springframework.web.bind.annotation.ResponseBody;  **import** java.io.IOException; **import** java.util.ArrayList; **import** java.util.List;  @Controller @RequestMapping(**"/search"**) **public class** SearchController {    @Autowired  **private** SolrClient **solrClient**;   @RequestMapping(**"/action/{keyword}"**)  @ResponseBody  **public** ResultBean searchByKeywrod(@PathVariable(name = **"keyword"**)String keyword){  *//简单的搜索、高亮搜索* SolrQuery query = **new** SolrQuery();  *//封装query* query.setQuery(keyword);  query.setStart(0);  query.setRows(10);  query.set(**"df"**,**"tb\_item\_keywords"**);   List<SearchBean> products = **new** ArrayList<>();   **try** {  *//执行查询，获得结果* QueryResponse response = **solrClient**.query(query);  SolrDocumentList results = response.getResults();  */\*  <field name="t\_product\_pname" type="text\_ik" indexed="true" stored="true"/>  <field name="t\_product\_price" type="pfloat" indexed="true" stored="true"/>  <field name="t\_product\_pimage" type="text\_general" indexed="true" stored="true"/>  <field name="t\_product\_pdesc" type="text\_ik" indexed="true" stored="true"/>  \*/   //解析结果，生成集合* **for** (SolrDocument doc : results) {  SearchBean product = **new** SearchBean();  product.setId(Long.*parseLong*((String) doc.get(**"id"**)));  product.setTProductPname((String) doc.get(**"t\_product\_pname"**));  product.setTProductPrice(((Float) doc.get(**"t\_product\_price"**)).longValue());  product.setTProductPimage((String) doc.get(**"t\_product\_pimage"**));  product.setTProductPdesc((String) doc.get(**"t\_product\_pdesc"**));   products.add(product);   }    **return** ResultBean.*success*(products);   } **catch** (SolrServerException e) {  e.printStackTrace();  } **catch** (IOException e) {  e.printStackTrace();  }    **return** ResultBean.*error*();  }  } |