**微信商城**

# 项目介绍

微信商城+微信公众号开发，该项目根据上海某大型知名电商企业互联网互联网项目，改版而来，使用最新微服务技术，页面使用高仿小米界面。功能包含电商模块,会员、SSO、订单、商品、支付、消息、微信、H5和PC、移动端、优惠券、后台系统、任务调度等模块。



# 电商系统特点

1.技术新

2.技术范围广

3.分布式

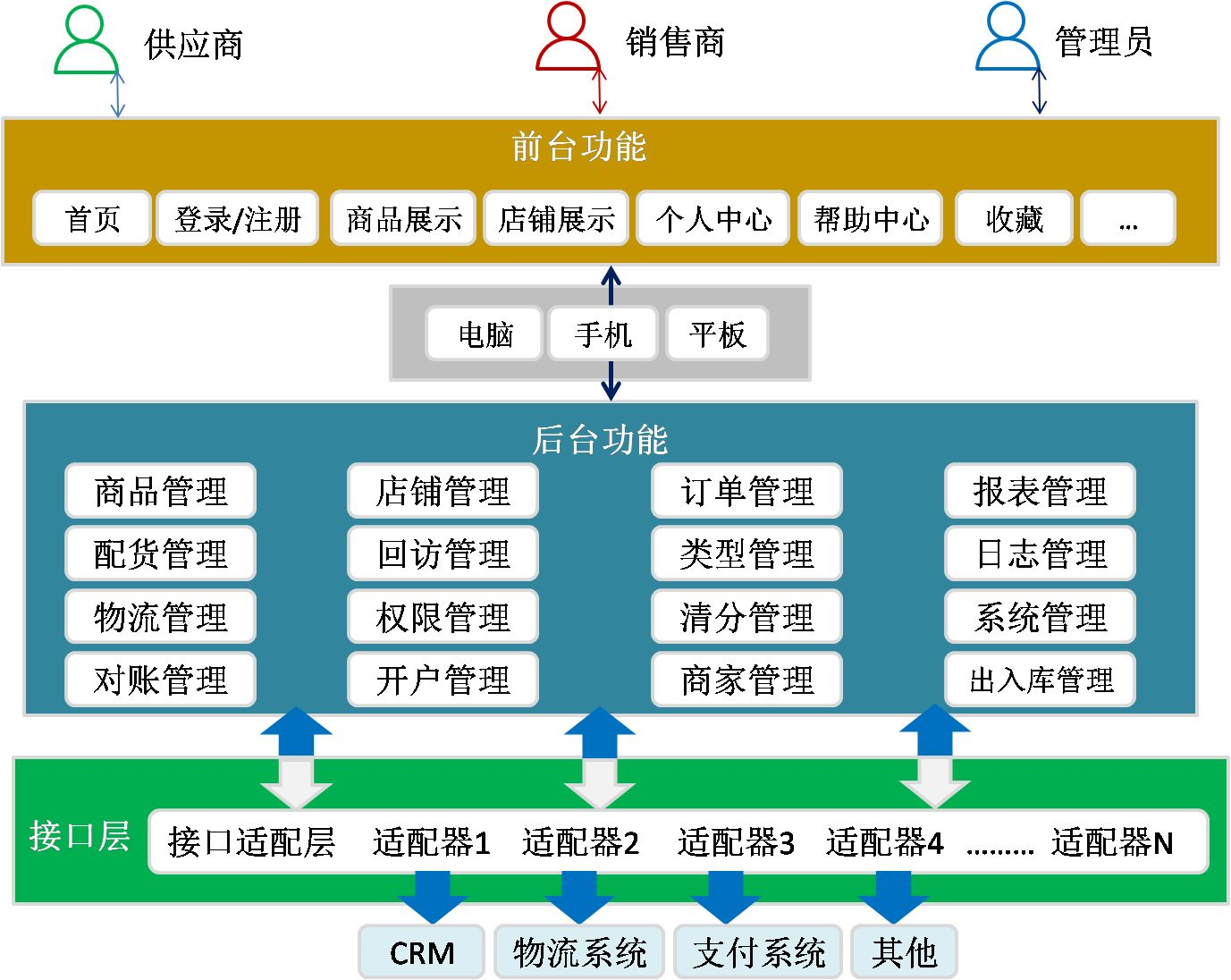
4.高并发、集群、负载均衡、高并发

5.海量数据

6.业务复杂

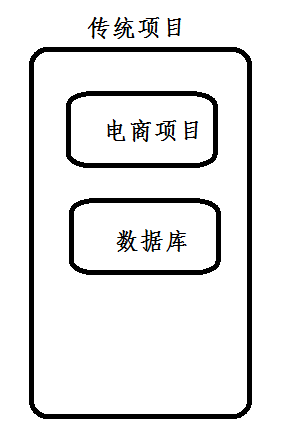
7.系统安全

# 电商系统架构

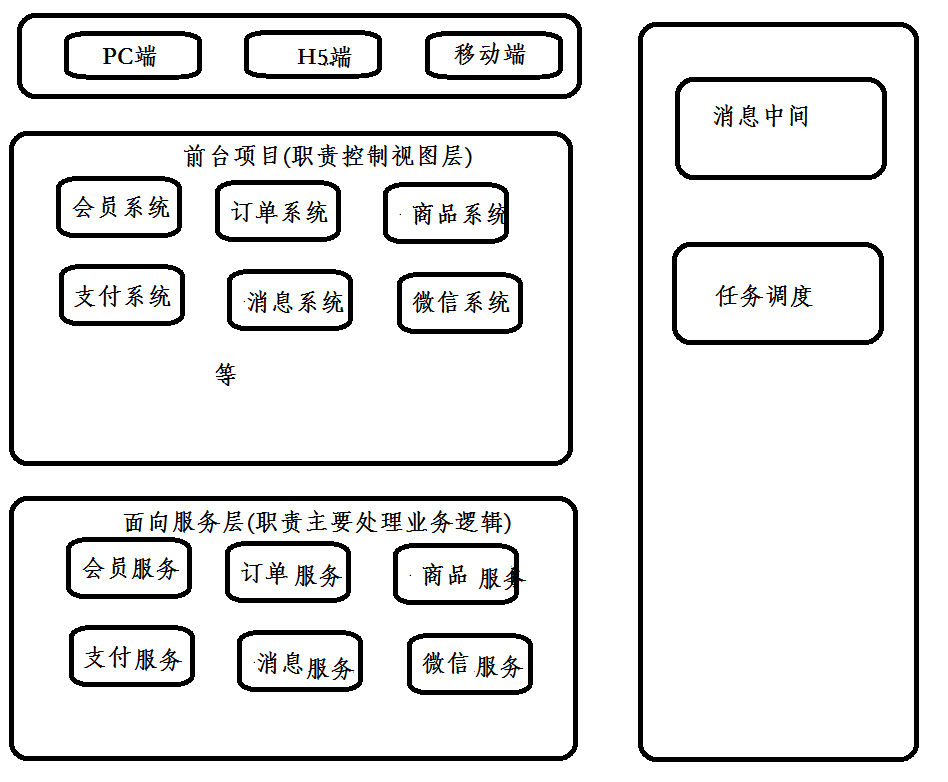


# 项目演变过程

## 传统架构



## 分布式架构



### 什么是分布式？

根据业务需求进行拆分成N个子系统，多个子系统相互协作才能完成业务流程子系统之间通讯使用RPC远程通讯技术。

### 什么是集群

同一个工程部署在多个不同的服务器上。

### 分布式架构优点

1.把模块拆分，使用接口通信，降低模块之间的耦合度。

2.把项目拆分成若干个子项目，不同的团队负责不同的子项目。

3.增加功能时只需要再增加一个子项目，调用其它系统的接口就可以。

4.可以灵活的进行分布式部署。

有优点就有缺点，缺点如下：

1.系统之间交互需要使用远程通信，接口开发增加工作量。

2.各个模块有一些通用的业务逻辑无法共用。

为了解决上面分布式架构的缺点，我们引入了soa架构，SOA：Service Oriented Architecture面向服务的架构。也就是把工程拆分成服务层、表现层两个工程。服务层中包含业务逻辑，只需要对外提供服务即可。表现层只需要处理和页面的交互，业务逻辑都是调用服务层的服务来实现。

# 项目拆分

在大型电商项目中,会将一个大的项目，拆分成N多个子模块，分配给不同的团队开发。

团队之间通讯采用RPC远程调用技术、使用Http+Restful+Json传输。

## 接口服务

会员服务、订单服务、商品服务、支付服务、消息服务、秒杀服务、优惠券服务等。

## 项目工程

H5项目、微信项目、后台管理等。

# 使用技术

### 电商项目

SpringBoot+SpringCloud +Maven+Redis+ActiveMQ+XXLJOB(分布式任务调度)+ Freemarker等。

使用 SpringCloud Eureka作为注册中心、Feign客户端调用工具、断路器Hystrix

视图展示使用Freemarker、数据库层使用Mybatis框架、缓存使用Redis、数据库使用MySQL

项目管理工具使用Maven、版本控制工具使用SVN、项目自动部署工具使用Jenkins

消息中间件使用ActiveMQ、分布式任务调度系统使用XXLJOB、反向代理工具使用Nginx

日志管理插件工具使用lombok、分布式日志收集使用Logstash、解析JSON框架使用FastJson

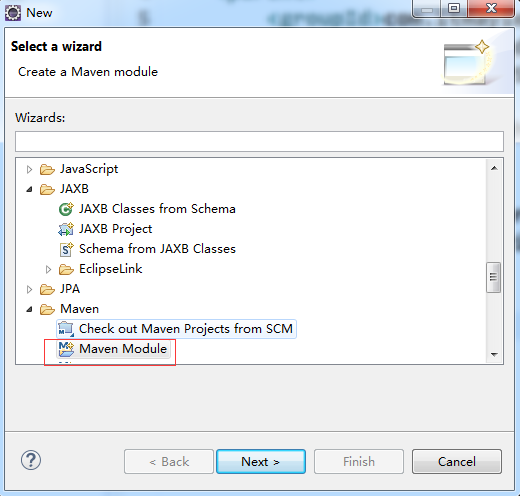
数据安全加密使用MD5加盐和Base64、RSA、分布式文件存储系统FastDFS等。

支付网关接口使用银联支付、第三方登录使用QQ授权等。

### 项目构建

|  |
| --- |
| itmayiedu-shopp-parent  ├─itmayiedu-shopp-common // 抽取的公共模块  ├─itmayiedu-shopp-eurekaserver // eureka注册中心  ├─itmayiedu-shopp-api // api服务  ├─itmayiedu-shopp-member-api // 会员服务  ├─itmayiedu-shopp-order-api // 订单服务  ├─itmayiedu-shopp-goods-api // 商品服务  ├─itmayiedu-shopp-pay-api // 支付服务  ├─itmayiedu-shopp-member // 会员系统  ├─itmayiedu-shopp-nessage // 消息系统  ├─itmayiedu-shopp-mobile-web // h5端工程 |

# 创建Modules



# 环境搭建

环境搭建步骤

1. 创建项目
2. 引入依赖
3. 搭建eureka注册中心
4. 封装控制层、日志、常量、常用工具、basedao

## itmayiedu-shopp-parent

|  |
| --- |
| **<parent>**  **<groupId>org.springframework.boot</groupId>**  **<artifactId>spring-boot-starter-parent</artifactId>**  **<version>1.5.3.RELEASE</version>**  **</parent>**  **<properties>**  **<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>**  **<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>**  **<java.version>1.8</java.version>**  **</properties>**  **<dependencies>**  **<!-- 集成commons工具类 -->**  **<dependency>**  **<groupId>org.apache.commons</groupId>**  **<artifactId>commons-lang3</artifactId>**  **<version>3.4</version>**  **</dependency>**  **<!-- 集成lombok 框架 -->**  **<dependency>**  **<groupId>org.projectlombok</groupId>**  **<artifactId>lombok</artifactId>**  **</dependency>**  **<!-- 集成redis -->**  **<dependency>**  **<groupId>org.springframework.boot</groupId>**  **<artifactId>spring-boot-starter-data-redis</artifactId>**  **</dependency>**  **<!-- 集成aop -->**  **<dependency>**  **<groupId>org.springframework.boot</groupId>**  **<artifactId>spring-boot-starter-aop</artifactId>**  **</dependency>**  **<!-- 集成web-->**  **<dependency>**  **<groupId>org.springframework.boot</groupId>**  **<artifactId>spring-boot-starter-web</artifactId>**  **</dependency>**  **<!-- springboot整合activemq -->**  **<dependency>**  **<groupId>org.springframework.boot</groupId>**  **<artifactId>spring-boot-starter-activemq</artifactId>**  **</dependency>**  **<!-- 集成发送邮件-->**  **<dependency>**  **<groupId>org.springframework.boot</groupId>**  **<artifactId>spring-boot-starter-mail</artifactId>**  **</dependency>**  **<!-- 集成mysql -->**  **<dependency>**  **<groupId>mysql</groupId>**  **<artifactId>mysql-connector-java</artifactId>**  **</dependency>**  **<dependency>**  **<groupId>org.springframework.boot</groupId>**  **<artifactId>spring-boot-starter-test</artifactId>**  **<scope>test</scope>**  **</dependency>**  **<dependency>**  **<groupId>org.mybatis</groupId>**  **<artifactId>mybatis</artifactId>**  **<version>3.4.0</version>**  **</dependency>**  **<dependency>**  **<groupId>tk.mybatis</groupId>**  **<artifactId>mapper</artifactId>**  **<version>3.3.7</version>**  **</dependency>**  **<!-- 阿里巴巴数据源 -->**  **<dependency>**  **<groupId>com.alibaba</groupId>**  **<artifactId>druid</artifactId>**  **<version>1.0.14</version>**  **</dependency>**  **<!-- httpclient -->**  **<dependency>**  **<groupId>commons-httpclient</groupId>**  **<artifactId>commons-httpclient</artifactId>**  **<version>3.1</version>**  **</dependency>**  **<dependency>**  **<groupId>org.apache.httpcomponents</groupId>**  **<artifactId>httpclient</artifactId>**  **</dependency>**  **<dependency>**  **<groupId>com.alibaba</groupId>**  **<artifactId>fastjson</artifactId>**  **<version>1.2.30</version>**  **</dependency>**  **<dependency>**  **<groupId>junit</groupId>**  **<artifactId>junit</artifactId>**  **</dependency>**  **<dependency>**  **<groupId>org.springframework</groupId>**  **<artifactId>spring-context-support</artifactId>**  **</dependency>**  **<dependency>**  **<groupId>commons-net</groupId>**  **<artifactId>commons-net</artifactId>**  **<version>3.3</version>**  **</dependency>**  **<dependency>**  **<groupId>org.springframework.cloud</groupId>**  **<artifactId>spring-cloud-starter-eureka-server</artifactId>**  **</dependency>**  **<dependency>**  **<groupId>org.springframework.cloud</groupId>**  **<artifactId>spring-cloud-starter-feign</artifactId>**  **</dependency>**  **</dependencies>**  **<dependencyManagement>**  **<dependencies>**  **<dependency>**  **<groupId>org.springframework.cloud</groupId>**  **<artifactId>spring-cloud-dependencies</artifactId>**  **<version>Dalston.RC1</version>**  **<type>pom</type>**  **<scope>import</scope>**  **</dependency>**  **</dependencies>**  **</dependencyManagement>**  **<build>**  **<plugins>**  **<plugin>**  **<groupId>org.springframework.boot</groupId>**  **<artifactId>spring-boot-maven-plugin</artifactId>**  **</plugin>**  **</plugins>**  **</build>**  **<repositories>**  **<repository>**  **<id>spring-milestones</id>**  **<name>Spring Milestones</name>**  **<url>https://repo.spring.io/milestone</url>**  **<snapshots>**  **<enabled>false</enabled>**  **</snapshots>**  **</repository>**  **</repositories>** |

## itmayiedu-shopp-eurekaserver

|  |
| --- |
| **@SpringBootApplication**  **@EnableEurekaServer**  **public class EureKaServer {**  **public static void main(String[] args) {**  **SpringApplication.*run*(EureKaServer.class, args);**  **}**  **}** |

|  |
| --- |
| **server:**  **port: 8761**  **eureka:**  **instance:**  **hostname: localhost**  **client:**  **registerWithEureka: false**  **fetchRegistry: false**  **serviceUrl:**  **defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/** |

## itmayiedu-shopp-member

|  |
| --- |
| **<dependencies>**  **<dependency>**  **<groupId>com.itmayiedu</groupId>**  **<artifactId>itmayiedu-shopp-member-api</artifactId>**  **<version>0.0.1-SNAPSHOT</version>**  **</dependency>**  **<dependency>**  **<groupId>com.itmayiedu</groupId>**  **<artifactId>itmayiedu-shopp-common</artifactId>**  **<version>0.0.1-SNAPSHOT</version>**  **</dependency>**  **<!-- springboot整合mybatis -->**  **<dependency>**  **<groupId>org.mybatis.spring.boot</groupId>**  **<artifactId>mybatis-spring-boot-starter</artifactId>**  **<version>1.3.0</version>**  **</dependency>**  **</dependencies>** |

### 配置文件

|  |
| --- |
| **server:**  **port: 8762**  **# context-path: /member**  **eureka:**  **client:**  **serviceUrl:**  **defaultZone: http://localhost:8761/eureka/**  **spring:**  **application:**  **name: member**  **redis:**  **host: 192.168.110.163**  **password: 123456**  **port: 6379**  **pool:**  **max-idle: 100**  **min-idle: 1**  **max-active: 1000**  **max-wait: -1**  **datasource:**  **name: test**  **url: jdbc:mysql://127.0.0.1:3306/itmayiedu-member**  **username: root**  **password: root**  **type: com.alibaba.druid.pool.DruidDataSource**  **driver-class-name: com.mysql.jdbc.Driver**  **filters: stat**  **maxActive: 20**  **initialSize: 1**  **maxWait: 60000**  **minIdle: 1**  **timeBetweenEvictionRunsMillis: 60000**  **minEvictableIdleTimeMillis: 300000**  **validationQuery: select 'x'**  **testWhileIdle: true**  **testOnBorrow: false**  **testOnReturn: false**  **poolPreparedStatements: true**  **maxOpenPreparedStatements: 20**  **activemq:**  **broker-url: tcp://localhost:61616**  **in-memory: true**  **pool:**  **enabled: false**  **messages:**  **queue: messages\_queue** |

# 工具类封装

## ListUtils封装

|  |
| --- |
| public class ListUtils {  //判断list集合是否为空  public List<?> emptyList(List<?> list) {  if (list == null || list.size() <= 0) {  return null;  }  return list;  }  //判断map集合是否为空  public Map<?,?> emptyMap(Map<?,?> map) {  if (map == null || map.size() <= 0) {  return null;  }  return map;  }  } |

## BaseApiService

|  |
| --- |
| public class BaseApiRedisService {  // 封装成功  public Map<String, Object> setResultSuccess() {  return setResult(Constants.*HTTP\_RES\_CODE\_200*, Constants.*HTTP\_RES\_CODE\_200\_VALUE*, null);  }  public Map<String, Object> setResultData(Object data) {  return setResult(Constants.*HTTP\_RES\_CODE\_200*, Constants.*HTTP\_RES\_CODE\_200\_VALUE*, data);  }  // 失败  public Map<String, Object> setResultError(String msg) {  return setResult(Constants.*HTTP\_RES\_CODE\_500*, msg, null);  }  // 自定义  public Map<String, Object> setResult(Integer code, String msg, Object data) {  Map<String, Object> resultMap = new HashMap<String, Object>();  resultMap.put(Constants.*HTTP\_RES\_CODE\_NAME*, code);  resultMap.put(Constants.*HTTP\_RES\_CODE\_MSG*, msg);  if (data != null)  resultMap.put(Constants.*HTTP\_RES\_CODE\_DATA*, data);  return resultMap;  }  } |

## Constants

|  |
| --- |
| public interface Constants {  // 响应code  String *HTTP\_RES\_CODE\_NAME* = "code";  // 响应msg  String *HTTP\_RES\_CODE\_MSG* = "msg";  // 响应data  String *HTTP\_RES\_CODE\_DATA* = "data";  // 响应请求成功  String *HTTP\_RES\_CODE\_200\_VALUE* = "success";  // 系统错误  String *HTTP\_RES\_CODE\_500\_VALUE* = "fial";  // 响应请求成功code  Integer *HTTP\_RES\_CODE\_200* = 200;  // 系统错误  Integer *HTTP\_RES\_CODE\_500* = 500;  } |

## BaseRedis

|  |
| --- |
| **@SuppressWarnings({ "rawtypes", "unchecked" })**  **@Component**  **public class BaseRedisService {**  **@Autowired**  **private StringRedisTemplate stringRedisTemplate;**  **public void setString(String key, Object data, Long timeout) {**  **if (data instanceof String) {**  **String value = (String) data;**  **stringRedisTemplate.opsForValue().set(key, value);**  **}**  **if (timeout != null) {**  **stringRedisTemplate.expire(key, timeout, TimeUnit.*SECONDS*);**  **}**  **}**  **public Object getString(String key) {**  **return stringRedisTemplate.opsForValue().get(key);**  **}**  **public void delKey(String key) {**  **stringRedisTemplate.delete(key);**  **}**  **}** |

### Linux环境下安装redis

#### linux 安装redis

Redis的官方下载网址是：  (这里下载的是Linux版的Redis源码包）

Redis服务器端的默认端口是6379。

**这里以虚拟机中的Linux系统如何安装Redis进行讲解。**

 在windows系统中下载好Redis的源码包。

1. 通过WinSCP工具，将Redis的源码包由windows上传到Linux系统的这个目录/opt/redis (即根目录下的lamp文件夹）。

2. 解压缩。

tar -zxf redis-2.6.17.tar.gz

3. 切换到解压后的目录。

cd redis-2.6.17            ( 一般来说，解压目录里的INSTALL文件或README文件里写有安装说明，可参考之）

4. 编译。

make

（注意，编译需要C语言编译器gcc的支持，如果没有，需要先安装gcc。可以使用rpm -q gcc查看gcc是否安装）

（利用yum在线安装gcc的命令    yum -y install gcc )

（如果编译出错，请使用make clean清除临时文件。之后，找到出错的原因，解决问题后再来重新安装。 ）

5. 进入到src目录。

cd src

6. 执行安装。

make install

到此就安装完成。但是，由于安装redis的时候，我们没有选择安装路径，故是默认位置安装。在此，我们可以将可执行文件和配置文件移动到习惯的目录。

cd /usr/local

mkdir -p /usr/local/redis/bin

mkdir -p /usr/local/redis/etc

cd /lamp/redis-2.6.17

mv ./redis.conf /usr/local/redis/etc

cd src

mv mkreleasehdr.sh redis-benchmark redis-check-aof redis-check-dump redis-cli redis-server redis-sentinel /usr/local/redis/bin

7．开放linux 6379 端口

1.编辑 /etc/sysconfig/iptables 文件：vi /etc/sysconfig/iptables  
加入内容并保存：-A RH-Firewall-1-INPUT -m state –state NEW -m tcp -p tcp –dport 6379 -j ACCEPT  
2.重启服务：/etc/init.d/iptables restart  
3.查看端口是否开放：/sbin/iptables -L -n

**比较重要的3个可执行文件：**

redis-server：Redis服务器程序

redis-cli：Redis客户端程序，它是一个命令行操作工具。也可以使用telnet根据其纯文本协议操作。

redis-benchmark：Redis性能测试工具，测试Redis在你的系统及配置下的读写性能。

#### Redis的启动命令：

/usr/local/redis/bin/redis-server

或

cd /usr/local/redis/bin

./redis-server /usr/local/redis/etc/redis.conf    为redis-server指定配置文

#### 修改 redis.conf文件

daemonize yes --- 修改为yes 后台启动

requirepass 123456  ----注释取消掉设置账号密码

ps aux | grep '6379' --- 查询端口

kill -15 9886 --- 杀死重置

kill -9 9886 --- 强制杀死

service iptables stop 停止防火强

#### redis命令连接方式

./redis-cli -h 127.0.0.1 -p 6379 -a "123456" --- redis 使用账号密码连接

PING 结果表示成功

### SpringBoot整合Redis

#### 步骤引入依赖

|  |
| --- |
| <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-data-redis</artifactId>  </dependency> |

#### Redis封装Java

在itmayiedu-shopp-common项目下创建一个BaseRedisService

|  |
| --- |
| **@SuppressWarnings({ "rawtypes", "unchecked" })**  **@Component**  **public class BaseRedisService {**  **@Autowired**  **private StringRedisTemplate stringRedisTemplate;**  **public void setString(String key, Object data, Long timeout) {**  **if (data instanceof String) {**  **String value = (String) data;**  **stringRedisTemplate.opsForValue().set(key, value);**  **}**  **if (timeout != null) {**  **stringRedisTemplate.expire(key, timeout, TimeUnit.*SECONDS*);**  **}**  **}**  **public Object getString(String key) {**  **return stringRedisTemplate.opsForValue().get(key);**  **}**  **public void delKey(String key) {**  **stringRedisTemplate.delete(key);**  **}**  **}** |

#### 配置文件新增redis连接

|  |
| --- |
| **spring:**  **redis:**  **host: 192.168.110.163**  **password: 123456**  **port: 6379**  **pool:**  **max-idle: 100**  **min-idle: 1**  **max-active: 1000**  **max-wait: -1** |

## 日志

### 安装lomBok插件

|  |
| --- |
| **1.下载lombok.jar包**[**https://projectlombok.org/download.html**](https://projectlombok.org/download.html)  **2.运行Lombok.jar:**[**Java**](http://lib.csdn.net/base/java) **-jar D:\software\lombok.jar D:\software\lombok.jar这是windows下lombok.jar所在的位置**  **数秒后将弹出一框，以确认eclipse的安装路径</code>**  **3.确认完eclipse的安装路径后，点击install/update按钮，即可安装完成**  **4.安装完成之后，请确认eclipse安装路径下是否多了一个lombok.jar包，并且其**  **配置文件eclipse.ini中是否 添加了如下内容: </code>**  **-javaagent:lombok.jar**  **-Xbootclasspath/a:lombok.jar**  **那么恭喜你已经安装成功，否则将缺少的部分添加到相应的位置即可 </code>**   1. **重启eclipse或myeclipse** |

### Lombok作用

**使用注解版本，生成get和set方法、简化重复代码**

### Maven坐标

|  |
| --- |
| **<!-- 集成lombok 框架 -->**  **<dependency>**  **<groupId>org.projectlombok</groupId>**  **<artifactId>lombok</artifactId>**  **</dependency>** |

### 生成get或set方法

|  |
| --- |
| **@Getter**  **public class TestEntity {**  **private String userId;**  **private String userName;**  **// public String getUserId() {**  **//**  **// return userId;**  **// }**  **// public void setUserId(String userId) {**  **//**  **// this.userId = userId;**  **// }**  **// public String getUserName() {**  **//**  **// return userName;**  **// }**  **// public void setUserName(String userName) {**  **//**  **// this.userName = userName;**  **// }**  **public static void main(String[] args) {**  **//使用**  **new TestEntity().getUserId();**  **}**    **}** |

**protected Logger log = Logger.getLogger(BaseApiService.class);**

**使用lomBok**

**@Slf4j**

### 打印日志

|  |
| --- |
| **@Slf4j**  **@RestController**  **public class MemberServiceImpl extends BaseApiRedisService implements MemberService {**    **@Autowired**  **private BaseRedisService baseRedisService;**  **@Override**  **public Map<String, Object> index() {**  **String result = "644064";**  ***log*.info("request .... index result:{}",result);**  **return setResultData(result);**  **}**  **}** |

### 使用aop打印请求参数

|  |
| --- |
| **@Aspect**  **// 申明是个spring管理的bean**  **@Component**  **@Slf4j**  **public class LogAspectServiceApi {**  **private JSONObject jsonObject = new JSONObject();**  **// 申明一个切点 里面是 execution表达式**  **@Pointcut("execution(public \* com.itmayiedu.service.\*.\*(..))")**  **private void controllerAspect() {**  **}**  **// 请求method前打印内容**  **@Before(value = "controllerAspect()")**  **public void methodBefore(JoinPoint joinPoint) {**  **ServletRequestAttributes requestAttributes = (ServletRequestAttributes) RequestContextHolder**  **.*getRequestAttributes*();**  **HttpServletRequest request = requestAttributes.getRequest();**  ***log*.info("===============请求内容===============");**  **try {**  **// 打印请求内容**  ***log*.info("请求地址:" + request.getRequestURL().toString());**  ***log*.info("请求方式:" + request.getMethod());**  ***log*.info("请求类方法:" + joinPoint.getSignature());**  ***log*.info("请求类方法参数:" + Arrays.*toString*(joinPoint.getArgs()));**  **} catch (Exception e) {**  ***log*.error("###LogAspectServiceApi.class methodBefore() ### ERROR:", e);**  **}**  ***log*.info("===============请求内容===============");**  **}**  **// 在方法执行完结后打印返回内容**  **@AfterReturning(returning = "o", pointcut = "controllerAspect()")**  **public void methodAfterReturing(Object o) {**  ***log*.info("--------------返回内容----------------");**  **try {**  ***log*.info("Response内容:" + jsonObject.*toJSONString*(o));**  **} catch (Exception e) {**  ***log*.error("###LogAspectServiceApi.class methodAfterReturing() ### ERROR:", e);**  **}**  ***log*.info("--------------返回内容----------------");**  **}**  **}** |

### 日志收集



## DateUtils

|  |
| --- |
| **@Slf4j**  **public class DateUtils {**  **/\*\* 年-月-日 时:分:秒 显示格式 \*/**  **// 备注:如果使用大写HH标识使用24小时显示格式,如果使用小写hh就表示使用12小时制格式。**  **public static String *DATE\_TO\_STRING\_DETAIAL\_PATTERN* = "yyyy-MM-dd HH:mm:ss";**  **/\*\* 年-月-日 显示格式 \*/**  **public static String *DATE\_TO\_STRING\_SHORT\_PATTERN* = "yyyy-MM-dd";**  **private static SimpleDateFormat *simpleDateFormat*;**  **/\*\***  **\* Date类型转为指定格式的String类型**  **\***  **\* @param source**  **\* @param pattern**  **\* @return**  **\*/**  **public static String DateToString(Date source, String pattern) {**  ***simpleDateFormat* = new SimpleDateFormat(pattern);**  **return *simpleDateFormat*.format(source);**  **}**  **/\*\***  **\***  **\* unix时间戳转为指定格式的String类型**  **\***  **\***  **\* System.currentTimeMillis()获得的是是从1970年1月1日开始所经过的毫秒数**  **\* unix时间戳:是从1970年1月1日（UTC/GMT的午夜）开始所经过的秒数,不考虑闰秒**  **\***  **\* @param source**  **\* @param pattern**  **\* @return**  **\*/**  **public static String timeStampToString(long source, String pattern) {**  ***simpleDateFormat* = new SimpleDateFormat(pattern);**  **Date date = new Date(source \* 1000);**  **return *simpleDateFormat*.format(date);**  **}**  **/\*\***  **\* 将日期转换为时间戳(unix时间戳,单位秒)**  **\***  **\* @param date**  **\* @return**  **\*/**  **public static long dateToTimeStamp(Date date) {**  **Timestamp timestamp = new Timestamp(date.getTime());**  **return timestamp.getTime() / 1000;**  **}**  **/\*\***  **\***  **\* 字符串转换为对应日期(可能会报错异常)**  **\***  **\* @param source**  **\* @param pattern**  **\* @return**  **\*/**  **public static Date stringToDate(String source, String pattern) {**  ***simpleDateFormat* = new SimpleDateFormat(pattern);**  **Date date = null;**  **try {**  **date = *simpleDateFormat*.parse(source);**  **} catch (ParseException e) {**  ***log*.error("字符串转换日期异常", e);**  **}**  **return date;**  **}**  **/\*\***  **\* 获得当前时间对应的指定格式**  **\***  **\* @param pattern**  **\* @return**  **\*/**  **public static String currentFormatDate(String pattern) {**  ***simpleDateFormat* = new SimpleDateFormat(pattern);**  **return *simpleDateFormat*.format(new Date());**  **}**  **/\*\***  **\* 获得当前unix时间戳(单位秒)**  **\***  **\* @return 当前unix时间戳**  **\*/**  **public static long currentTimeStamp() {**  **return System.*currentTimeMillis*() / 1000;**  **}**  **/\*\***  **\***  **\* @methodDesc: 功能描述:(获取当前系统时间戳)**  **\* @param: @return**  **\*/**  **public static Timestamp getTimestamp() {**  **return new Timestamp(new Date().getTime());**  **}**  **// 待补充** |

## MD5Util

|  |
| --- |
| **public class MD5Util {**    **public final static String MD5(String s) {**  **char hexDigits[]={'0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'};**  **try {**  **byte[] btInput = s.getBytes();**  **// 获得MD5摘要算法的 MessageDigest 对象**  **MessageDigest mdInst = MessageDigest.*getInstance*("MD5");**  **// 使用指定的字节更新摘要**  **mdInst.update(btInput);**  **// 获得密文**  **byte[] md = mdInst.digest();**  **// 把密文转换成十六进制的字符串形式**  **int j = md.length;**  **char str[] = new char[j \* 2];**  **int k = 0;**  **for (int i = 0; i < j; i++) {**  **byte byte0 = md[i];**  **str[k++] = hexDigits[byte0 >>> 4 & 0xf];**  **str[k++] = hexDigits[byte0 & 0xf];**  **}**  **return new String(str);**  **} catch (Exception e) {**  **e.printStackTrace();**  **return null;**  **}**  **}**  **public static void main(String[] args) {**  **System.*out*.println(MD5Util.*MD5*("20121221"));**  **System.*out*.println(MD5Util.*MD5*("加密"));**  **}**  **}** |

## BaseDao

### maven依赖

|  |
| --- |
| <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis</artifactId>  <version>3.4.0</version>  </dependency>  <dependency>  <groupId>tk.mybatis</groupId>  <artifactId>mapper</artifactId>  <version>3.3.7</version>  </dependency>  <!-- springboot整合mybatis -->  <dependency>  <groupId>org.mybatis.spring.boot</groupId>  <artifactId>mybatis-spring-boot-starter</artifactId>  <version>1.3.0</version>  </dependency> |

### 数据源连接

|  |
| --- |
| **spring:**  **application:**  **name: member**  **#redis连接信息**  **redis:**  **host: 192.168.110.163**  **password: 123456**  **port: 6379**  **pool:**  **max-idle: 100**  **min-idle: 1**  **max-active: 1000**  **max-wait: -1**  **#数据库连接信息**  **datasource:**  **name: test**  **url: jdbc:mysql://127.0.0.1:3306/itmayiedu-member**  **username: root**  **password: root**  **# 使用druid数据源**  **type: com.alibaba.druid.pool.DruidDataSource**  **driver-class-name: com.mysql.jdbc.Driver**  **filters: stat**  **maxActive: 20**  **initialSize: 1**  **maxWait: 60000**  **minIdle: 1**  **timeBetweenEvictionRunsMillis: 60000**  **minEvictableIdleTimeMillis: 300000**  **validationQuery: select 'x'**  **testWhileIdle: true**  **testOnBorrow: false**  **testOnReturn: false**  **poolPreparedStatements: true**  **maxOpenPreparedStatements: 20** |

### 创建Baseentity

|  |
| --- |
| **@Getter**  **@Setter**  **public class BaseEntity {**  **/\*\***  **\* 主键ID**  **\*/**  **private Long id;**  **/\*\***  **\* 创建时间**  **\*/**  **private Timestamp created;**  **/\*\***  **\* 修改时间**  **\*/**  **private Timestamp updated;**  **}** |

### 创建Basedao

|  |
| --- |
| **/\*\***  **\***  **\* @classDesc: 功能描述:(BaseDao)**  **\* @author: 蚂蚁课堂创始人-余胜军**  **\* @QQ: 644064779**  **\* @QQ粉丝群: 116295598**  **\* @createTime: 2017年10月21日 下午9:10:25**  **\* @version: v1.0**  **\* @copyright:每特学院(蚂蚁课堂)上海每特教育科技有限公司**  **\*/**  **public interface BaseDao {**  **/\*\***  **\***  **\***  **\* @methodDesc: 功能描述:(增加持久化对象)**  **\* @param: @param**  **\* t**  **\*/**  **@InsertProvider(type = BaseProvider.class, method = "save")**  **public void save(@Param("oj") Object oj, @Param("table") String table);**  **/\*\***  **\***  **\***  **\* @methodDesc: 功能描述:(修改持久化对象)**  **\* @param: @param**  **\* t**  **\* @return**  **\*/**  **@InsertProvider(type = BaseProvider.class, method = "update")**  **public void update(@Param("oj") Object oj, @Param("table") String table, @Param("idKey") Long idKey);**  **}** |

### 创建BaseProvider

|  |
| --- |
| **public class BaseProvider {**  **public String save(Map<String, Object> para) {**  **final Object oj = para.get("oj");**  **final String table = (String) para.get("table");**  **String sql = new SQL() {**  **{**  **INSERT\_INTO(table);**  **String columns = ReflectionUtils.*getInsertFields*(oj);**  **String values = ReflectionUtils.*getInsertDeclaredFieldsValue*(oj);**  **VALUES(columns, values);**  **}**  **}.toString();**  **return sql;**  **}**  **public String update(Map<String, Object> para) {**  **final Object oj = para.get("oj");**  **final String table = (String) para.get("table");**  **final Long idKey = (Long) para.get("idKey");**  **String sql = new SQL() {**  **{**  **UPDATE(table);**  **SET(ReflectionUtils.*updateAllSerField*(oj));**  **WHERE("id="+idKey + "");**  **}**  **}.toString();**  **return sql;**  **}**  **}** |

### 创建ReflectionUtils工具类

|  |
| --- |
| **/\*\***  **\***  **\* @classDesc: 功能描述:(Java反射工具类)**  **\* @author: 蚂蚁课堂创始人-余胜军**  **\* @QQ: 644064779**  **\* @QQ粉丝群: 116295598**  **\* @createTime: 2017年10月21日 下午9:51:54**  **\* @version: v1.0**  **\* @copyright:每特学院(蚂蚁课堂)上海每特教育科技有限公司**  **\*/**  **@Slf4j**  **public class ReflectionUtils {**  **/\*\***  **\***  **\* @methodDesc: 功能描述:(获取类的属性,拼接成字符串)**  **\* @param: @return**  **\*/**  **public static String getInsertFields(Object oj) {**  **if (oj == null) {**  **return null;**  **}**  **Class cl = oj.getClass();**  **// 获取所有的属性**  **Field[] declaredFields = cl.getDeclaredFields();**  **String sb1 = *appendFields*(declaredFields);**  **Class superclass = cl.getSuperclass();**  **Field[] superField = superclass.getDeclaredFields();**  **String sb2 = *appendFields*(superField);**  **return sb1 + "," + sb2;**  **}**  **/\*\***  **\***  **\* @methodDesc: 功能描述:(获取类的属性,拼接成字符串的值)**  **\* @param: @return**  **\* @throws IllegalAccessException**  **\* @throws IllegalArgumentException**  **\* @throws InstantiationException**  **\*/**  **public static String getInsertDeclaredFieldsValue(Object oj) {**  **if (oj == null) {**  **return null;**  **}**  **Class cl = oj.getClass();**  **// 获取所有的属性**  **Field[] declaredFields = cl.getDeclaredFields();**  **String sb1 = *appendFieldValues*(declaredFields, oj);**  **Class superclass = cl.getSuperclass();**  **Field[] superField = superclass.getDeclaredFields();**  **String sb2 = *appendFieldValues*(superField, oj);**  **return sb1 + "," + sb2;**  **}**  **/\*\***  **\***  **\* @methodDesc: 功能描述:(获取类的属性,拼接成字符串值)**  **\* @param: @param**  **\* oj**  **\* @param: @return**  **\*/**  **public static String updateAllSerField(Object oj) {**  **if (oj == null) {**  **return null;**  **}**  **Class cl = oj.getClass();**  **// 获取所有的属性**  **Field[] declaredFields = cl.getDeclaredFields();**  **String sb1 = *updateSerField*(declaredFields, oj);**  **Field[] supDeclaredFields = cl.getSuperclass().getDeclaredFields();**  **String sb2 = *updateSerField*(supDeclaredFields, oj);**  **return sb1 + "," + sb2;**  **}**  **public static String updateSerField(Field[] declaredFields, Object oj) {**  **StringBuffer sb = new StringBuffer();**  **try {**  **for (int i = 0; i < declaredFields.length; i++) {**  **String name = declaredFields[i].getName();**  **if (name.equals("id")) {**  **continue;**  **}**  **declaredFields[i].setAccessible(true);**  **Object value = declaredFields[i].get(oj);**  **if (value == null) {**  **continue;**  **}**  **sb.append(name + "=" + "'" + value + "'");**  **if ((i < declaredFields.length - 1)) {**  **sb.append(",");**  **}**  **}**  **} catch (Exception e) {**  ***log*.error("###updateSerField() ERROR:", e);**  **}**  **return sb.toString();**  **}**  **public static String appendFieldValues(Field[] declaredFields, Object oj) {**  **StringBuffer sf = new StringBuffer();**  **try {**  **for (int i = 0; i < declaredFields.length; i++) {**  **Field field = declaredFields[i];**  **String name = field.getName();**  **if (name == "id") {**  **continue;**  **}**  **field.setAccessible(true);// 设置私有权限访问**  **sf.append("'" + field.get(oj) + "'");**  **if (i < declaredFields.length - 1) {**  **sf.append(",");**  **}**  **}**  **} catch (Exception e) {**  ***log*.error("###ERROR:getDeclaredFieldsValue方法出现异常:", e);**  **}**  **return sf.toString();**  **}**  **public static String appendFields(Field[] declaredFields) {**  **StringBuffer sf = new StringBuffer();**  **// 获取到子类的**  **for (int i = 0; i < declaredFields.length; i++) {**  **Field field = declaredFields[i];**  **String name = field.getName();**  **if (name == "id") {**  **continue;**  **}**  **sf.append(field.getName());**  **if (i < declaredFields.length - 1) {**  **sf.append(",");**  **}**  **}**  **return sf.toString();**  **}**  **}** |

### 代码测试

@EnableFeignClients

# 数据库表设计

数据库分表分库为垂直和水平拆分。

什么是垂直拆分 按照不同业务模块进行划分数据库，例如 会员数据库、订单数据库。

什么是水平拆分 将表进行一个分类，存放在一个数据库中,例如 会员表1、会员表2。

## 会员服务表设计

|  |
| --- |
| CREATE TABLE `mb\_user` (  `id` bigint(20) NOT NULL AUTO\_INCREMENT,  `username` varchar(50) NOT NULL COMMENT '用户名',  `password` varchar(32) NOT NULL COMMENT '密码，加密存储',  `phone` varchar(20) DEFAULT NULL COMMENT '注册手机号',  `email` varchar(50) DEFAULT NULL COMMENT '注册邮箱',  `created` datetime NOT NULL,  `updated` datetime NOT NULL,  PRIMARY KEY (`id`),  UNIQUE KEY `username` (`username`) USING BTREE,  UNIQUE KEY `phone` (`phone`) USING BTREE,  UNIQUE KEY `email` (`email`) USING BTREE  ) ENGINE=InnoDB AUTO\_INCREMENT=37 DEFAULT CHARSET=utf8 COMMENT='用户表';  INSERT INTO `mb\_user` (username,password,phone,email,created,updated) VALUES ('yushengjun2', 'e10adc3949ba59abbe56e057f20f883e', '15527339673', 'aa1@a', '2015-04-06 17:03:55', '2015-04-06 17:03:55'); |

# 移动端模块

### 注册功能

需要会员团队需要提供登录和注册接口。

##### 会员系统提供注册接口

1. 使用手机号码查找该用户是否存在
2. 进行注册准备,密码使用MD5加密
3. 如果不存在,则可以注册
4. 注册成功后,使用MQ调用消息服务接口,推送一条邮箱注册成功通知。

###### 代码

|  |
| --- |
| **@Slf4j**  **@RestController**  **public class MbUserServiceImpl extends BaseApiService implements MbUserService {**  **@Autowired**  **private MemberDao memberDao;**  **@Autowired**  **private RegisterMailboxProducer registerMailboxProducer;**  **@Value("${messages.queue}")**  **private String MESSAGES\_QUEUE;**  **@Override**  **public Map<String, Object> register(@RequestBody MbUser mbUser) {**  **try {**  **// ① 使用手机号码查找该用户是否存在**  **MbUser resultMbUser = memberDao.getMbUser(mbUser);**  **if (resultMbUser != null) {**  **return setResultError("该手机号码已经被注册");**  **}**  **// 密码加密并且加盐**  **String userName = mbUser.getUsername();**  **String password = mbUser.getPassword();**  **String newPassWord = MD5Util.*MD5*(userName + password);**  **mbUser.setPassword(newPassWord);**  **// ② 如果不存在,则可以注册**  **mbUser.setCreated(DateUtils.*getTimestamp*());**  **mbUser.setUpdated(DateUtils.*getTimestamp*());**  **memberDao.save(mbUser, ConstantsTables.*TABLE\_MB\_USER*);**  **// ③ 注册成功后,调用消息服务接口,推送一条邮箱注册成功通知。**  **String email = mbUser.getEmail();**  **String message = message(email);**  ***log*.info("###register()调用消息中间 发送注册成功邮箱通知结果 message:{}",message);**  **sendMess(message);**  **} catch (Exception e) {**  ***log*.error("####register()####,ERROR:", e);**  **return setResultError("系统错误!");**  **}**  **return setResultSuccess();**  **}**  **private String message(String mail) {**  **JSONObject root = new JSONObject();**  **JSONObject header = new JSONObject();**  **header.put("interfaceType", "sms\_mail");**  **JSONObject content = new JSONObject();**  **content.put("mail", mail);**  **root.put("header", header);**  **root.put("content", content);**  **return root.toJSONString();**  **}**  **private void sendMess(String json) {**  **Destination activeMQQueue = new ActiveMQQueue(MESSAGES\_QUEUE);**  **registerMailboxProducer.sendMess(activeMQQueue, json);**  **}**  **}** |

##### 消息系统提供注册消息服务

###### 整合SpringBoot整合ActiveMQ

|  |
| --- |
| <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-activemq</artifactId>  </dependency> |

###### 自定义消息报文头

|  |
| --- |
| {  "header": {  "interfaceType": "接口类型"  },  "content": {}  } |

###### 生产者

引入配置文件

|  |
| --- |
| **server:**  **port: 8762**  **context-path: /member**  **eureka:**  **client:**  **serviceUrl:**  **defaultZone: http://localhost:8761/eureka/**  **spring:**  **application:**  **name: member**  **#redis连接信息**  **redis:**  **host: 192.168.110.163**  **password: 123456**  **port: 6379**  **pool:**  **max-idle: 100**  **min-idle: 1**  **max-active: 1000**  **max-wait: -1**  **#数据库连接信息**  **datasource:**  **name: test**  **url: jdbc:mysql://127.0.0.1:3306/itmayiedu-member**  **username: root**  **password: root**  **# 使用druid数据源**  **type: com.alibaba.druid.pool.DruidDataSource**  **driver-class-name: com.mysql.jdbc.Driver**  **filters: stat**  **maxActive: 20**  **initialSize: 1**  **maxWait: 60000**  **minIdle: 1**  **timeBetweenEvictionRunsMillis: 60000**  **minEvictableIdleTimeMillis: 300000**  **validationQuery: select 'x'**  **testWhileIdle: true**  **testOnBorrow: false**  **testOnReturn: false**  **poolPreparedStatements: true**  **maxOpenPreparedStatements: 20**  **##activemq连接信息**  **activemq:**  **broker-url: tcp://localhost:61616**  **in-memory: true**  **pool:**  **enabled: false**  **##队列**  **messages:**  **queue: messages\_queue** |

代码

|  |
| --- |
| **/\*\***  **\***  **\* @classDesc: 功能描述:(使用MQ调用消息服务,发送注册邮件)**  **\* @author: 蚂蚁课堂创始人-余胜军**  **\* @QQ: 644064779**  **\* @QQ粉丝群: 116295598**  **\* @createTime: 2017年10月22日 下午9:13:18**  **\* @version: v1.0**  **\* @copyright:每特学院(蚂蚁课堂)上海每特教育科技有限公司**  **\*/**  **@Service("registerMailboxProducer")**  **public class RegisterMailboxProducer {**  **@Autowired**  **private JmsMessagingTemplate jmsMessagingTemplate;**  **/\*\***  **\***  **\* @methodDesc: 功能描述:(消息服务发送者)**  **\* @param: @param**  **\* destination**  **\* @param: @param**  **\* json**  **\*/**  **public void sendMess(Destination destination, String json) {**  **jmsMessagingTemplate.convertAndSend(destination, json);**  **}**  **}** |

|  |
| --- |
| **private String message(String mail) {**  **JSONObject root = new JSONObject();**  **JSONObject header = new JSONObject();**  **header.put("interfaceType", "sms\_mail");**  **JSONObject content = new JSONObject();**  **content.put("mail", mail);**  **root.put("header", header);**  **root.put("content", content);**  **return root.toJSONString();**  **}**  **private void sendMess(String json) {**  **Destination activeMQQueue = new ActiveMQQueue(MESSAGES\_QUEUE);**  **registerMailboxProducer.sendMess(activeMQQueue, json);**  **}** |

###### 消费者

创建itmayiedu-shopp-message消息工程项目

配置文件引入

|  |
| --- |
| **server:**  **port: 8763**  **context-path: /nessage**  **eureka:**  **client:**  **serviceUrl:**  **defaultZone: http://localhost:8761/eureka/**  **spring:**  **application:**  **name: message**  **activemq:**  **broker-url: tcp://localhost:61616**  **in-memory: true**  **pool:**  **enabled: false**  **mail:**  **host: smtp.163.com**  **username: yushengjun644064@163.com**  **password: itmayiedu644**  **enable: true**  **smtp:**  **auth: true**  **starttls:**  **enable: true**  **required: true** |

创建mq进行分发消息

|  |
| --- |
| **/\*\***  **\***  **\* @classDesc: 功能描述:(mq分发)**  **\* @author: 蚂蚁课堂创始人-余胜军**  **\* @QQ: 644064779**  **\* @QQ粉丝群: 116295598**  **\* @createTime: 2017年10月22日 下午9:00:45**  **\* @version: v1.0**  **\* @copyright:每特学院(蚂蚁课堂)上海每特教育科技有限公司**  **\*/**  **@Slf4j**  **@Component**  **public class ConsumerDistribute {**  **@Autowired**  **private SMSMailboxService smsMailboxService;**  **/\*\***  **\***  **\***  **\* @methodDesc: 功能描述:(接受生产者消息)**  **\* @param: 报文json**  **\*/**  **@SuppressWarnings("static-access")**  **@JmsListener(destination = "messages\_queue")**  **public void distribute(String json) {**  **if (StringUtils.*isEmpty*(json)) {**  **return;**  **}**  ***log*.info("###接受到消息内容json:{}", json);**  **JSONObject root = new JSONObject().*parseObject*(json);**  **JSONObject header = root.getJSONObject("header");**  **String interfaceType = header.getString("interfaceType");**  **if (StringUtils.*isEmpty*(interfaceType)) {**  **return;**  **}**  **MessageAdapter messageAdapter = null;**  **switch (interfaceType) {**  **// 发送邮箱**  **case Constants.*SMS\_MAIL*:**  **messageAdapter = smsMailboxService;**  **break;**  **default:**  **break;**  **}**  **// 执行消息服务**  **JSONObject content = root.getJSONObject("content");**  **messageAdapter.distribute(content);**  **}**  **}** |

创建适配器接口

|  |
| --- |
| **public interface MessageAdapter {**  **public void distribute( JSONObject jsonObject);**  **}** |

发送邮件实现

|  |
| --- |
| **@Component**  **@Slf4j**  **public class SMSMailboxService implements MessageAdapter {**  **@Autowired**  **private JavaMailSender mailSender; // 自动注入的Bean**  **@Value("${spring.mail.username}")**  **private String sender; // 读取配置文件中的参数**  **@Override**  **public void distribute(JSONObject jsonObject) {**  **String mail = jsonObject.getString("mail");**  **// 开始发送邮件**  **SimpleMailMessage message = new SimpleMailMessage();**  **message.setFrom(mail);**  **message.setTo(sender); // 自己给自己发送邮件**  **message.setSubject("恭喜您成为微信商城的信用...");**  **message.setText("恭喜您今天成为了微信商城的用户,谢谢您的关注!");**  ***log*.info("###发送短信邮箱 mail:{}", mail);**  **mailSender.send(message);**  **}**  **}** |

启动

|  |
| --- |
| **@SpringBootApplication**  **@EnableDiscoveryClient**  **public class AppMessage {**  **public static void main(String[] args) {**  **SpringApplication.*run*(AppMessage.class, args);**  **}**  **}** |

###### 测试邮箱账号密码

[yushengjun644064@163.com](mailto:yushengjun644064@163.com)

密码 123456w

授权码itmayiedu644

##### 前端页面调用会员注册接口

###### 创建一个移动端工程

###### SpringBoot整合freemarker

|  |
| --- |
| **<dependency>**  **<groupId>org.springframework.boot</groupId>**  **<artifactId>spring-boot-starter-freemarker</artifactId>**  **</dependency>**  **<!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->**  **<dependency>**  **<groupId>javax.servlet</groupId>**  **<artifactId>javax.servlet-api</artifactId>**  **<scope>provided</scope>**  **</dependency>** |

###### 配置文件

|  |
| --- |
| server:  port: 8764  context-path: /mobile  eureka:  client:  serviceUrl:  defaultZone: http://localhost:8761/eureka/  spring:  freemarker:  suffix: .ftl  templateEncoding: UTF-8  templateLoaderPath: classpath:/templates/  application:  name: mobile |

###### 新建一个ftl文件

在 resources目录下创建 templates文件夹专门存放ftl文件

###### Eclipse ftl文件如何提亮

<http://bbs.itmayiedu.com/article/1508740165679>

###### 代码

使用@FeignClient调用

引入**itmayiedu-shopp-member-api依赖**

|  |
| --- |
| **<dependency>**  **<groupId>com.itmayiedu</groupId>**  **<artifactId>itmayiedu-shopp-member-api</artifactId>**  **<version>0.0.1-SNAPSHOT</version>**  **</dependency>** |

|  |
| --- |
| **@FeignClient(name = "member")**  **public interface RefMbUserService extends MbUserService {**  **}** |

控制层代码

|  |
| --- |
| **@Slf4j**  **@Controller**  **public class RegisterController extends BaseController {**  **// 跳转到注册**  **private static final String *LOCAREGISTER* = "locaRegister";**  **// 登录页面**  **private static final String *LOGIN* = "login";**  **@Autowired**  **private RefMbUserService refMbUserService;**  **@RequestMapping("/locaRegister")**  **public String locaRegister() {**  **return *LOCAREGISTER*;**  **}**  **@RequestMapping("/register")**  **public String register(@ModelAttribute("mbUser") MbUser mbUser, HttpServletRequest request) {**  **try {**  **Map<String, Object> registerResult = refMbUserService.register(mbUser);**  ***log*.info("####refMbUserService.register result:" + registerResult);**  **Integer code = (Integer) registerResult.get("code");**  **Integer httpResCode200 = Constants.*HTTP\_RES\_CODE\_200*;**  **if (!code.equals(httpResCode200)) {**  **String msg = (String) registerResult.get("msg");**  **return setError(msg, request, *LOCAREGISTER*);**  **}**  **return "redirect:/" + *LOGIN*;**  **} catch (Exception e) {**  ***log*.error("###register() 发送错误###", e);**  **return setError("注册失败!请稍后重试!", request, *LOCAREGISTER*);**  **}**  **}**  **public String setError(String errorMsg, HttpServletRequest request, String addres) {**  **request.setAttribute("error", errorMsg);**  **return addres;**  **}**  **public static void main(String[] args) {**  **Integer code = 200;**  **Integer httpResCode200 = 200;**  **System.*out*.println(code.equals(httpResCode200));**  **}**  **}** |

页面展示

|  |
| --- |
| **<!DOCTYPE html>**  **<html>**  **<head>**  **<title>注册微信商城账户</title>**  **<meta content=*"yes"* name=*"apple-mobile-web-app-capable"* />**  **<!-- ios系统的私有标签，它指定在web app状态下，ios设备中顶端的状态条的颜色 -->**  **<meta content=*"black"* name=*"apple-mobile-web-app-status-bar-style"* />**  **<!-- 设备浏览网页时对数字不启用电话功能 -->**  **<meta content=*"telephone=no,email=no"* name=*"format-detection"* />**  **<meta charset=*"UTF-8"*>**  **<meta name=*"viewport"***  **content=*"width=device-width,initial-scale=1.0,minimum-scale=1.0, maximum-scale=1.0,user-scalable=no"*>**  **<link rel=*"stylesheet"* href=*"css/common.css"*>**  **<!-- 自适应样式单 -->**  **<link rel=*"stylesheet"* href=*"css/adaptive.css"*>**  **<link rel=*"stylesheet"* href=*"css/login.css"*>**  **<body class=*"zh\_CN"*>**  **<div class=*"layout"*>**  **<div class=*"nl-content"*>**  **<div class=*"nl-logo-area"* id=*"custom\_display\_1"*>**  **<a href=*"javascript:void(0)"*> <img id=*"logo-img"***  **src=*"data:image/png;base64,"***  **data-src=*"/static/img/passport/nl-logo.png"***  **onerror=*"this.src=&#39;/static/img/passport/nl-logo.png&#39;"***  **width=*"60"*>**  **</a>**  **</div>**  **<h1 class=*"nl-login-title"* id=*"custom\_display\_256"*>**  **<span id=*"message\_LOGIN\_TITLE"*>注册微信商城账户</span>**  **</h1>**  **<h2 class=*"nl-login-title lsrp-appname display-custom-hide"***  **id=*"lsrp\_appName"*></h2>**  **<div class=*"nl-phone-tip"*>**  **<div id=*"message\_LOGIN\_PHONETIP"*>非大陆地区请在手机号码前面添加当地的国际代码前缀(如香港**  **+852 台湾 +886)</div>**  **<div id=*"select\_country\_code"*>帮助</div>**  **</div>**  **<div class=*"nl-frame-container"*>**  **<div class=*"ng-form-area show-place"* id=*"form-area"*>**  **<form method=*"post"* action=*"register"* id=*"miniLogin"***  **onsubmit="return MiniLogin.validateForm();">**  **<div class=*"shake-area"* id=*"shake\_area"* style="z-index: *30*;">**  **<div class=*"enter-area display-custom-hide"* id=*"revalidate\_user"*>**  **<p class=*"revalidate-user-name"* id=*"revalidate\_user\_name"*></p>**  **</div>**  **<div class=*"enter-area"* id=*"enter\_user"*>**  **<input type=*"text"* class=*"enter-item first-enter-item"***  **data-rule=*"(^[\w.\-]+@(?:[A-Za-z0-9]+(?:-[A-Za-z0-9]+)\*\.)+[A-Za-z]{2,6}$)|(^1\d{10}$)|(^\d{3,}$)|(^\++\d{2,}$)"***  **id=*"miniLogin\_username"* name=*"username"* autocomplete=*"off"***  **placeholder=*"用户名称"*> <i class=*"placeholder hide"***  **id=*"message\_INPUT\_IDENTITY"*>用户名称</i> <span class=*"error-tip"*><em**  **class=*"error-ico"*></em><span class=*"error-msg"*></span></span>**  **</div>**  **<div class=*"enter-area"* id=*"enter\_user"*>**  **<input type=*"text"* name=*"email"***  **class=*"enter-item last-enter-item"***  **data-rule=*"(^[\w.\-]+@(?:[A-Za-z0-9]+(?:-[A-Za-z0-9]+)\*\.)+[A-Za-z]{2,6}$)|(^1\d{10}$)|(^\d{3,}$)|(^\++\d{2,}$)"***  **id=*"miniLogin\_username"* name=*"phone"* autocomplete=*"off"***  **placeholder=*"邮&nbsp;&nbsp;&nbsp;&nbsp;箱"*> <i**  **class=*"placeholder hide"* id=*"message\_INPUT\_IDENTITY"*>邮&nbsp;&nbsp;&nbsp;&nbsp;箱</i>**  **<span class=*"error-tip"*><em class=*"error-ico"*></em><span**  **class=*"error-msg"*></span></span>**  **</div>**  **</div>**  **<div class=*"shake-area"* id=*"shake\_area"* style="z-index: *30*;">**  **<div class=*"enter-area display-custom-hide"* id=*"revalidate\_user"*>**  **<p class=*"revalidate-user-name"* id=*"revalidate\_user\_name"*></p>**  **</div>**  **<div class=*"enter-area"* id=*"enter\_user"*>**  **<input type=*"text"* name=*"phone"***  **class=*"enter-item first-enter-item"***  **data-rule=*"(^[\w.\-]+@(?:[A-Za-z0-9]+(?:-[A-Za-z0-9]+)\*\.)+[A-Za-z]{2,6}$)|(^1\d{10}$)|(^\d{3,}$)|(^\++\d{2,}$)"***  **id=*"miniLogin\_username"* name=*"phone"* autocomplete=*"off"***  **placeholder=*"手机号码"*> <i class=*"placeholder hide"***  **id=*"message\_INPUT\_IDENTITY"*>手机号码</i> <span class=*"error-tip"*><em**  **class=*"error-ico"*></em><span class=*"error-msg"*></span></span>**  **</div>**  **<div class=*"enter-area"* style="z-index: *20*;">**  **<input type=*"password"* class=*"enter-item last-enter-item"***  **id=*"miniLogin\_pwd"* name=*"password"* autocomplete=*"off"***  **data-rule=*""* placeholder=*"密码"*> <i**  **class=*"placeholder hide"* id=*"message\_INPUT\_PASSWORD"*>密码</i> <span**  **class=*"error-tip"*><em class=*"error-ico"*></em><span**  **class=*"error-msg"*></span></span>**  **</div>**  **</div>**  **<#if error??>**  **<div>**  **<span id=*"message\_LOGIN\_TOO\_MUCH"* style="color: *red*">**  **${error}</span>**  **</div>**  **</#if>**  **<div class=*"miniLogin\_forbidden"* id=*"miniLogin\_forzen"*>**  **<div>**  **<span id=*"message\_LOGIN\_FORZEN"*>此帐号已被冻结，暂时无法登录</span>**  **</div>**  **</div>**  **<input class=*"button orange"* type=*"submit"***  **id=*"message\_LOGIN\_IMMEDIATELY"* value=*"立即注册"*> <a**  **style="display: *none*" id=*"redirectLink"* href=*""* target=*"\_top"*></a>**  **<a style="display: *none*" id=*"redirectTwoPhraseLoginLink"* href=*""*></a>**  **</form>**  **</div>**  **</div>**  **<div class=*"web-info"*>**  **<p class=*"web-info-content"* id=*"web\_info\_content"*></p>**  **</div>**  **</div>**  **<div class=*"nl-footer"*>**  **<div class=*"nl-f-nav"* id=*"nl\_f\_nav"*>**  **<span id=*"custom\_display\_4"*> <a href=*"javascript:void(0);"***  **onclick="change\_lang(&#39;zh\_CN&#39;); return false;" class=*"zh-cn"*>简体</a>|**  **<a href=*"javascript:void(0);"***  **onclick="change\_lang(&#39;zh\_TW&#39;); return false;" class=*"zh-tw"*>繁体</a>|**  **<a href=*"javascript:void(0);"***  **onclick="change\_lang(&#39;en&#39;); return false;" class=*"zh-en"*>English</a>**  **</span> <span id=*"custom\_display\_32"*> <span class=*"n\_common\_line"*>|</span>**  **<span id=*"message\_FAQLIST"*><a**  **href=*"http://static.account.xiaomi.com/html/faq/faqList.html"***  **target=*"\_blank"*>常见问题</a></span>**  **</span>**  **</div>**  **<p class=*"nl-f-copyright"* id=*"message\_COPYRIGHT"*>上海每特教育科技有限公司版权所有-京ICP备10046444-京公网安备1101080212535-京ICP证110507号</p>**  **</div>**  **<div id=*"modal-mask"* class=*"modal-mask display-custom-hide"*></div>**  **<div class=*"modal-container display-custom-hide"***  **id=*"qrlogin-container"*>**  **<div id=*"qrlogin-close"* class=*"modal-close"*>Close</div>**  **<iframe id=*"qrlogin-iframe"* frameborder=*"0"* border=*"0"***  **class=*"qrlogin-iframe"***  **style="width: *340px*; height: *340px*; margin-top: *-170px*; margin-left: *-170px*;"></iframe>**  **</div>**  **</div>**  **<a target=*"\_blank"*></a>**  **</body>**  **</html>** |

### 登录和QQ授权登录

#### 登录步骤

1. 使用手机和密码数据库查找对应信息
2. 如果用户信息存在, 生成对应的token作为key,value为用户userId存放在Redis中
3. 前台项目获取到token,存放在Cookie中，其他页面直接通过Cookie获取token、

#### Cookie工具类

|  |
| --- |
| **/\*\***  **\* Cookie工具类**  **\***  **\*/**  **public class CookieUtil {**  **private CookieUtil() {**  **}**  **/\*\***  **\* 添加cookie**  **\***  **\* @param response**  **\* @param name**  **\* @param value**  **\* @param maxAge**  **\*/**  **public static void addCookie(HttpServletResponse response, String name, String value, int maxAge) {**  **Cookie cookie = new Cookie(name, value);**  **cookie.setPath("/");**  **if (maxAge > 0) {**  **cookie.setMaxAge(maxAge);**  **}**  **response.addCookie(cookie);**  **}**  **/\*\***  **\* 删除cookie**  **\***  **\* @param response**  **\* @param name**  **\*/**  **public static void removeCookie(HttpServletResponse response, String name) {**  **Cookie uid = new Cookie(name, null);**  **uid.setPath("/");**  **uid.setMaxAge(0);**  **response.addCookie(uid);**  **}**  **/\*\***  **\* 获取cookie值**  **\***  **\* @param request**  **\* @return**  **\*/**  **public static String getUid(HttpServletRequest request,String cookieName) {**  **Cookie cookies[] = request.getCookies();**  **for (Cookie cookie : cookies) {**  **if (cookie.getName().equals(cookieName)) {**  **return cookie.getValue();**  **}**  **}**  **return null;**  **}**  **}** |

#### QQ授权网站

QQ授权登录流程

<http://wiki.connect.qq.com/%e7%bd%91%e7%ab%99%e5%ba%94%e7%94%a8%e6%8e%a5%e5%85%a5%e6%b5%81%e7%a8%8b>

QQ登录授权步骤：

<http://wiki.connect.qq.com/%e4%bd%bf%e7%94%a8authorization_code%e8%8e%b7%e5%8f%96access_token>

#### 底部公共引入

<#include "common/bottom.ftl">

#### java对接QQ登录

#### 将Sdk4J.jar打包到maven本地仓库或私服中

mvn install:install-file -Dfile=F:/Sdk4J.jar -DgroupId=com.sdk4j -DartifactId=sdk4j -Dversion=1.0 -Dpackaging=jar

#### maven依赖

|  |
| --- |
| **<dependency>**  **<groupId>com.sdk4j</groupId>**  **<artifactId>sdk4j</artifactId>**  **<version>1.0</version>**  **</dependency>** |

#### 将QQ授权配置文件



修改信息app\_ID、app\_KEY、redirect\_URI

#### 封装跳转QQ登录地址

|  |
| --- |
| **/\*\***  **\* 跳转到QQ授权地址**  **\* @param request**  **\* @return**  **\* @throws QQConnectException**  **\*/**  **@RequestMapping("/locaQQLogin")**  **public String locaQQLogin(HttpServletRequest request) throws QQConnectException {**  **String authorizeURL = new Oauth().getAuthorizeURL(request);**  **return "redirect:"+authorizeURL;**  **}** |

#### 封装QQ授权回调地址

|  |
| --- |
| **/\*\***  **\* QQ回调地址**  **\***  **\* @param request**  **\* @return**  **\*/**  **@RequestMapping("/qqLoginCallback")**  **public String qqLoginCallback(HttpServletRequest request, HttpSession httpSession, HttpServletResponse response) {**  **try {**  **// 先获取accessToken**  **AccessToken accessTokenObj = new Oauth().getAccessTokenByRequest(request);**  **if (accessTokenObj == null) {**  **return setError(request, "qq授权失败!", *ERROR*);**  **}**  **String accessToken = accessTokenObj.getAccessToken();**  **if (StringUtils.*isEmpty*(accessToken)) {**  **return setError(request, "qq授权失败!", *ERROR*);**  **}**  **// 获取openid**  **OpenID openIdObj = new OpenID(accessToken);**  **String userOpenID = openIdObj.getUserOpenID();**  **// 使用openid查找用户信息是否绑定**  **Map<String, Object> openIdMap = userFeign.findLogin(userOpenID);**  **Integer code = (Integer) openIdMap.get(BaseApiConstants.*HTTP\_CODE\_NAME*);**  **// 判断是否绑定会员**  **if (code.equals(BaseApiConstants.*HTTP\_200\_CODE*)) {**  **// 已经授权过,自动登录**  **String token = (String) openIdMap.get("data");**  **CookieUtil.*addCookie*(response, Constants.*USER\_TOKEN*, token,**  **Constants.*WEBUSER\_COOKIE\_TOKEN\_TERMVALIDITY*);**  **return *INDEX*;**  **}**  **// 没有绑定openid**  **httpSession.setAttribute("openid", userOpenID);**  **} catch (Exception e) {**  **}**  **// 跳转到关联页面**  **return *ASSOCIATEDACCOUNT*;**  **}** |

#### 服务登录代码

|  |
| --- |
| **@Override**  **public Map<String, Object> login(UserEntity userEntity) {**  **// 往数据进行查找数据**  **String phone = userEntity.getPhone();**  **String password = userEntity.getPassword();**  **String newPassWord = md5PassSalt(phone, password);**  **UserEntity userPhoneAndPwd = userDao.getUserPhoneAndPwd(phone, newPassWord);**  **if (userPhoneAndPwd == null) {**  **return setResutError("账号或密码错误");**  **}**  **String token = setLoginToken(userEntity.getId());**  **String openId = userEntity.getOpenId();**  **if (!StringUtils.*isEmpty*(openId)) {**  **// 关联到数据库中,update**  **UserEntity newUserEntity = new UserEntity();**  **newUserEntity.setOpenId(openId);**  **newUserEntity.setUpdated(DateUtils.*getTimestamp*());**  **userDao.update(newUserEntity, DBTableName.*TABLE\_MB\_USER*, userPhoneAndPwd.getId());**  **}**  **return setResutSuccessData(token);**  **}** |