**微服务电商系统**

# 课程安排

电商系统架构

使用新技术介绍

项目环境搭建

工具类的封装

# 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

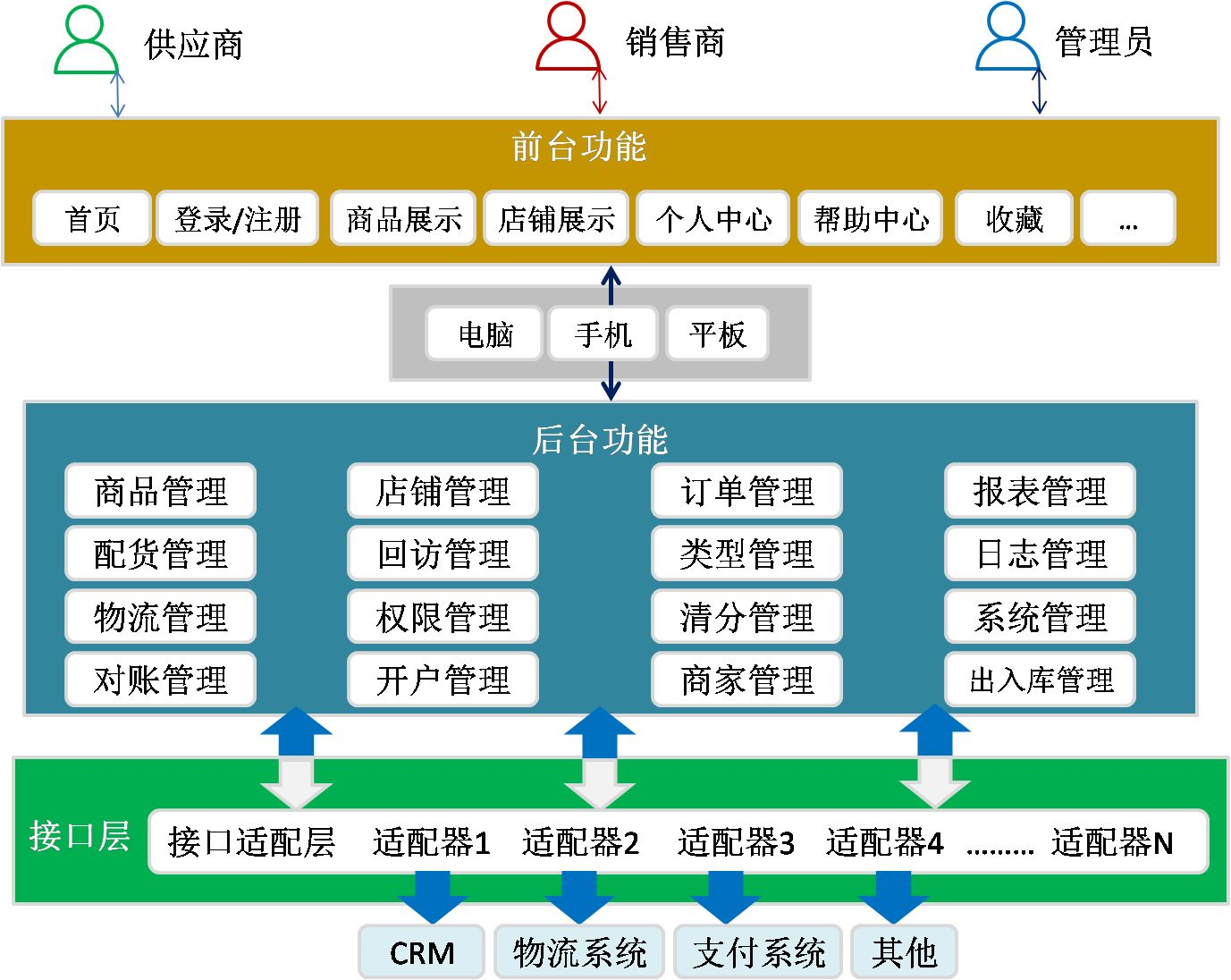
4.高并发、集群、负载均衡、高并发

5.海量数据

6.业务复杂

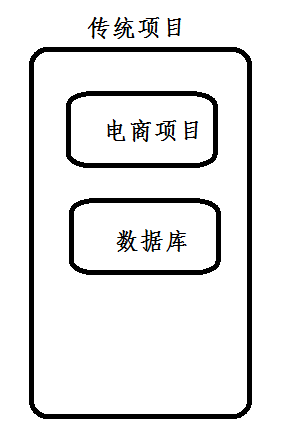
7.系统安全

# 电商系统架构

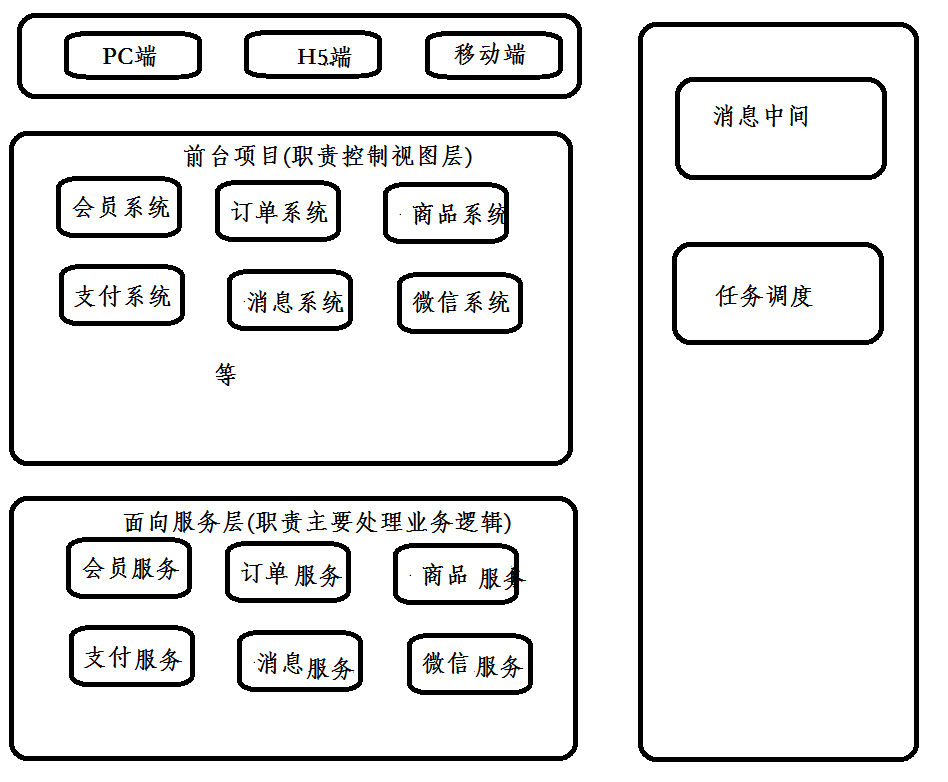


# 项目演变过程

## 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

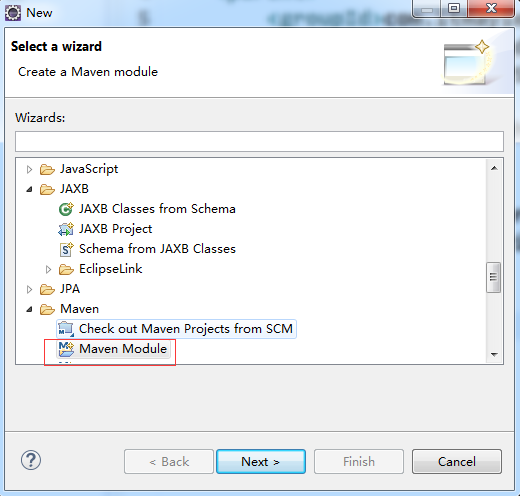
数据安全加密使用MD5加盐和Base64、RSA、分布式文件存储系统FastDFS等。

支付网关接口使用支付宝、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

创建项目

引入依赖

搭建eureka注册中心

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-common

### Maven依赖

|  |
| --- |
| <parent>  <groupId>com.itmayiedu</groupId>  <artifactId>itmayiedu-shopp-parent</artifactId>  <version>0.0.1-SNAPSHOT</version>  </parent>  <artifactId>itmayiedu-shopp-common</artifactId>  <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-data-redis</artifactId>  </dependency>  </dependencies> |

## itmayiedu-shopp-eurekaserver

### Maven依赖

|  |
| --- |
| <parent>  <groupId>com.itmayiedu</groupId>  <artifactId>itmayiedu-shopp-parent</artifactId>  <version>0.0.1-SNAPSHOT</version>  </parent>  <groupId>com.itmayiedu</groupId>  <artifactId>itmayiedu-shopp-eurekaservers</artifactId>  <version>0.0.1-SNAPSHOT</version> |

### 启动EureKaServer

创建eureka作为服务注册中心

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

### application.yml

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

## itmayiedu-shopp-api

创建itmayiedu-shopp-api工程，存放所有的需要提供给外部接口项

目

### itmayiedu-shopp-member-api

#### 创建MemberService测试接口

|  |
| --- |
| @RequestMapping("/member")  **public** **interface** MemberService {  @RequestMapping("/testRest")  **public** Map<String, Object> testRest();  } |

## itmayiedu-shopp-member

### Maven依赖

|  |
| --- |
| **<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** |

### 实现接口MemberServiceImpl

|  |
| --- |
| @RestController  **public** **class** MemberServiceImpl **implements** MemberService {  @Override  **public** Map<String, Object> testRest() {  Map<String, Object> result = **new** HashMap<>();  result.put("errorCode", "200");  result.put("errorMsg", "success");  **return** result;  }  } |

# 工具类封装

## 创建ResponseBase

|  |
| --- |
| **public** **class** ResponseBase {  // 响应code  **private** Integer code;  // 消息内容  **private** String msg;  // 返回data  **private** Object data;  **public** ResponseBase() {  }  **public** ResponseBase(Integer code, String msg, Object data) {  **super**();  **this**.code = code;  **this**.msg = msg;  **this**.data = data;  }  **public** Integer getCode() {  **return** code;  }  **public** **void** setCode(Integer code) {  **this**.code = code;  }  **public** String getMsg() {  **return** msg;  }  **public** **void** setMsg(String msg) {  **this**.msg = msg;  }  **public** Object getData() {  **return** data;  }  **public** **void** setData(Object data) {  **this**.data = data;  }  } |

## BaseController

|  |
| --- |
| **public** **class** BaseController {  // 返回成功 ,data值为null  **public** ResponseBase setResultSuccess() {  **return** setResult(Constants.***HTTP\_RES\_CODE\_200***, Constants.***HTTP\_RES\_CODE\_200\_VALUE***, **null**);  }  // 返回成功 ,data可传  **public** ResponseBase setResultSuccess(Object data) {  **return** setResult(Constants.***HTTP\_RES\_CODE\_200***, Constants.***HTTP\_RES\_CODE\_200\_VALUE***, data);  }  // 返回失败  **public** ResponseBase setResultError(String msg){  **return** setResult(Constants.***HTTP\_RES\_CODE\_500***,msg, **null**);  }  // 自定义返回结果  **public** ResponseBase setResult(Integer code, String msg, Object data) {  ResponseBase responseBase = **new** ResponseBase();  responseBase.setCode(code);  responseBase.setMsg(msg);  **if** (data != **null**)  responseBase.setData(data);  **return** responseBase;  }  } |

## 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;  }  } |

## 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：[http://redis.io/download](http://redis.io/download" \t "_blank)  (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

|  |
| --- |
| @Component  **public** **class** BaseRedisService {  @Autowired  **private** StringRedisTemplate stringRedisTemplate;  **public** **void** setString(String key, Object data) {  setString(key, data, **null**);  }  **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** String getString(String key) {  **return** (String) stringRedisTemplate.opsForValue().get(key);  }  **public** **void** delKey(String key) {  stringRedisTemplate.delete(key);  }  } |

#### 配置文件新增redis连接

|  |
| --- |
| **spring:**  **redis:**  **host: 127.0.0.1**  **password: 123456**  **port: 6379**  **pool:**  **max-idle: 100**  **min-idle: 1**  **max-active: 1000**  **max-wait: -1** |

#### 测试结果

##### MemberService

|  |
| --- |
| @RequestMapping("/setRedisTest")  **public** ResponseBase setRedisTest(String key, String value);  @RequestMapping("/getRedis")  **public** ResponseBase getRedis(String key); |

##### MemberServiceImpl

|  |
| --- |
| @Override  **public** ResponseBase setRedisTest(String key, String value) {  baseRedisService.setString(key, value);  **return** setResultSuccess();  }  @Override  **public** ResponseBase getRedis(String key) {  String value = baseRedisService.getString(key);  **return** setResultSuccess(value);  } |

## 日志

### 安装lomBok插件

|  |
| --- |
| **1.下载lombok.jar包[https://projectlombok.org/download.html](https://projectlombok.org/download.html" \t "http://blog.csdn.net/arkblue/article/details/_blank)**  **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();**  **}**    **}** |

### 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"));**  **}**  **}** |

## 整合MySQL数据库

### Maven依赖

|  |
| --- |
| <!-- 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** |

### 数据库表结构

|  |
| --- |
| 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 ('yushengjun3', 'e10adc3949ba59abbe56e057f20f883e', '15527339674', 'aa1@a', '2015-04-06 17:03:55', '2015-04-06 17:03:55'); |

### itmayiedu-shopp-member-api 创建实体类

|  |
| --- |
| @Getter  @Setter  **public** **class** UserEntity {  **private** Integer id;  **private** String username;  **private** String password;  **private** String phone;  **private** String email;  **private** Date created;  **private** Date updated;  } |

### itmayiedu-shopp-member 创建Dao层

|  |
| --- |
| @Mapper  **public** **interface** UserDao {  @Select("select id,username,password,phone,email,created,updated from mb\_user where id =#{userId}")  UserEntity findByID(@Param("userId") Long userId);  @Insert("INSERT INTO `mb\_user` (username,password,phone,email,created,updated) VALUES (#{username}, #{password},#{phone},#{email},#{created},#{updated});")  Integer insertUser(UserEntity userEntity);  } |

### 提供接口测试数据库访问层

|  |
| --- |
| **public** ResponseBase findUser(Long userId) {  UserEntity userEntity = userDao.findByID(userId);  **if**(userEntity==**null**){  **return** setResultError("为查找到该用户信息");  }  **return** setResultSuccess(userEntity);  } |

# 会员服务接口

## 会员注册

技术要求:会员注册成功后，调用消息服务平台采用异步方式发送邮件通知

### 新增用户注册接口

|  |
| --- |
| **public** ResponseBase register(@RequestBody UserEntity user) {  String passWord=user.getPassword();  String newPassWord=MD5Util.*MD5*(passWord);  user.setPassword(newPassWord);  Integer insertUser = userDao.insertUser(user);  **if** (insertUser <= 0) {  **return** setResultError("注册失败!");  }  **return** setResultSuccess();  } |

### 测试请求参数

|  |  |
| --- | --- |
| 请求地址 | http://127.0.0.1:8762/member/register |
| 请求参数 | {  "username":"644069",  "password":"test",  "email":"yushengjun2018@163.com",  "created":"2018-03-14",  "updated":"2018-03-14"    } |

### 调用消息服务平台发送邮件

#### MQ接口协议定义

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

#### Maven新增依赖

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

#### 配置文件新增信息

|  |
| --- |
| ##activemq连接信息  activemq:  broker-url: tcp://localhost:61616  in-memory: **true**  pool:  enabled: **false**  ##队列  messages:  queue: messages\_queue |

#### 生产者管理接口

|  |
| --- |
| @Service  **public** **class** RegisterMailboxProducer {  @Autowired  **private** JmsMessagingTemplate jmsMessagingTemplate;  **public** **void** sendMsg(Destination destination, String json) {  jmsMessagingTemplate.convertAndSend(destination, json);  }  } |

#### 生产者发送邮件

|  |
| --- |
| @Override  **public** ResponseBase register(@RequestBody UserEntity user) {  String passWord = user.getPassword();  String newPassWord = MD5Util.*MD5*(passWord);  user.setPassword(newPassWord);  Integer insertUser = userDao.insertUser(user);  **if** (insertUser <= 0) {  **return** setResultError("注册失败!");  }  // 采用MQ异步发送邮件  String email = user.getEmail();  String messAageJson = message(email);  ***log***.info("######email:{},messAageJson:{}",email,messAageJson);  sendMsg(messAageJson);  **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** sendMsg(String json) {  ActiveMQQueue activeMQQueue = **new** ActiveMQQueue(MESSAGESQUEUE);  registerMailboxProducer.sendMsg(activeMQQueue, json);  }  } |

# 消息服务平台

## 创建itmayiedu-shopp-message项目

### Maven依赖

|  |
| --- |
| <dependencies>  <dependency>  <groupId>com.itmayiedu</groupId>  <artifactId>itmayiedu-shopp-common</artifactId>  <version>0.0.1-SNAPSHOT</version>  </dependency>  </dependencies> |

### 配置文件

|  |
| --- |
| 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: yushengjun2018@163.com  password: itmayiedu644  enable: **true**  smtp:  auth: **true**  starttls:  enable: **true**  required: **true**  ########################################  msg:  subject: 蚂蚁课堂|每特教育提示您，会员注册成功邮件。  text: 恭喜您的账户{},注册成功!蚂蚁课堂|每特教育 上海每特教育科技有限公司 www.itmayiedu.com |

### 创建适配接口

|  |
| --- |
| **public** **interface** MessageAdapter {  **public** **void** sendMsg(JSONObject body);  } |

### 消息管理接口

|  |
| --- |
| @Slf4j  @Component  **public** **class** ConsumerDistribute {  @Autowired  **private** MailService mailService;  **private** MessageAdapter messageAdapter;  @JmsListener(destination = "messages\_queue")  **public** **void** distribute(String json) {  ***log***.info("####ConsumerDistribute###distribute() 消息服务平台接受 json参数:" + json);  **if** (StringUtils.*isEmpty*(json)) {  **return**;  }  JSONObject jsonObecjt = **new** JSONObject().*parseObject*(json);  JSONObject header = jsonObecjt.getJSONObject("header");  String interfaceType = header.getString("interfaceType");  **if** (StringUtils.*isEmpty*(interfaceType)) {  **return**;  }  **if** (interfaceType.equals(Constants.***SMS\_MAIL***)) {  messageAdapter = mailService;  }  **if** (messageAdapter == **null**) {  **return**;  }  JSONObject body = jsonObecjt.getJSONObject("content");  messageAdapter.sendMsg(body);  }  } |

### 发送邮件具体实现

|  |
| --- |
| @Service  **public** **class** MailService **implements** MessageAdapter {  @Value("${msg.subject}")  **private** String subject;  @Value("${msg.text}")  **private** String text;  @Autowired  **private** JavaMailSender mailSender; // 自动注入的Bean  @Override  **public** **void** sendMsg(JSONObject body) {  String mail = body.getString("mail");  **if** (StringUtils.*isEmpty*(mail)) {  **return**;  }  SimpleMailMessage simpleMailMessage = **new** SimpleMailMessage();  // 发送  simpleMailMessage.setFrom(mail);  simpleMailMessage.setTo(mail);  // 标题  simpleMailMessage.setSubject(subject);  // 内容  simpleMailMessage.setText(text.replace("{}", mail));  mailSender.send(simpleMailMessage);  }  } |