# 会员系统

## 总体目标

搭建环境

会员登录注册

发起众筹项目

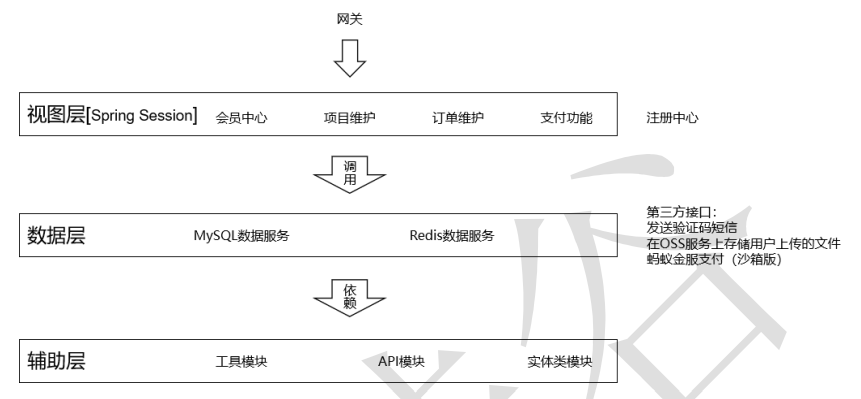
展示众筹项目

支持众筹项目

订单

支付

## 整体架构



## 工程概述

父工程、聚合工程：唯一的pom工程 crowd-07-member-parent

注册中心：crowd-08-member-eureka

实体类模块：crowd-09-member-entity

MySQL 数据服务：crowd-10-member-mysql-provider

Redis 数据服务：crowd-11-member-redis-provider

会员中心：crowd-12-member-authentication-consumer

项目维护：crowd-13-member-project-consumer

订单维护：crowd-14-member-order-consumer

支付功能：crowd-15-member-pay-consumer

网关：crowd-16-member-zuul

API 模块：crowd-17-member-api

# 环境搭建

## 约定

1.包名约定：新创建的包都作为 com.atguigu.crowd 的子包

2.主启动类类名：CrowdMainClass

3.端口号：

eureka 1000

mysql-provider 2000

redis-provider 3000

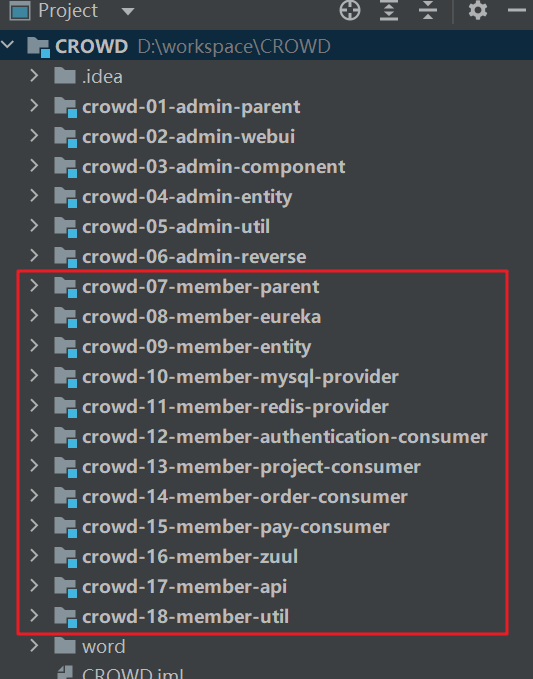
authentication-consumer 4000

project-consumer 5000

order-consumer 7000

pay-consumer 8000

zuul 80



## eureka工程

### 依赖

<dependencies>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>  
 </dependency>  
</dependencies>

### 主启动类

加上@EnableEurekaServer注解

@EnableEurekaServer  
@SpringBootApplication  
public class CrowdMainClass {  
 public static void main(String[] args) {  
 SpringApplication.*run*(CrowdMainClass.class);  
 System.*out*.println("Hello world!");  
 }  
}

### application.yml

server:  
 port: 1000  
  
spring:  
 application:  
 name: eureka  
  
eureka:  
 instance:  
 *#配置当前注册中心的服务主机地址* hostname: localhost  
 client:  
 *# 自己就是注册中心，所以自己不注册自己* register-with-eureka: false  
 *# 自己就是注册中心，所以不需要“从注册中心取回信息”* fetch-registry: false  
 service-url:  
 *#客户端（consumer和provider）访问 Eureka 时使用的地址* defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka

## entity工程

### 实体类的进一步划分

1.VO：View Object 视图对象

用途 1：接收浏览器发送过来的数据

用途 2：把数据发送给浏览器去显示 

2.PO：Persistent Object 持久化对象

用途 1：将数据封装到 PO 对象存入数据库

用途 2：将数据库数据查询出来存入 PO 对象

所以 PO 对象是和数据库表对应，一个数据库表对应一个PO 对象

3.DO：Data Object 数据对象

用途 1：从 Redis 查询得到数据封装为 DO 对象

用途 2：从 ElasticSearch 查询得到数据封装为 DO 对象

用途 3：从 Solr 查询得到数据封装为 DO 对象 ……

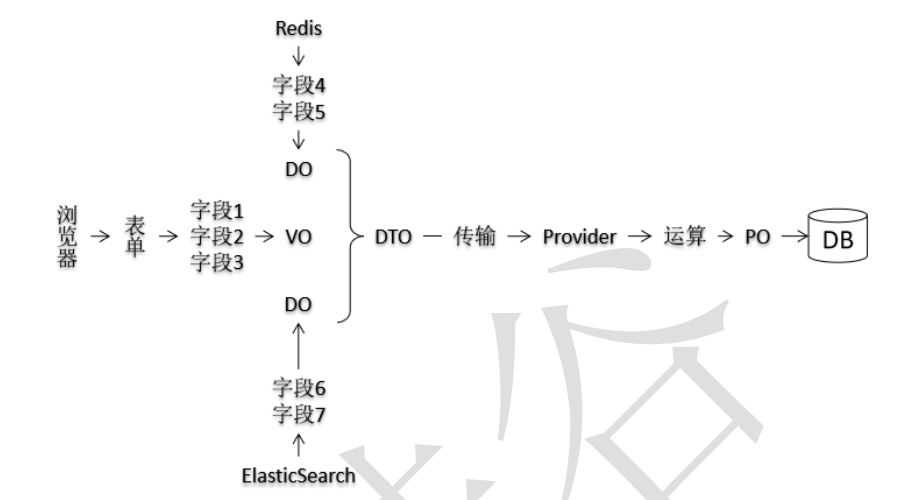
从中间件或其他第三方接口查询到的数据封装为 DO 对象

4.DTO：Data Transfer Object 数据传输对象

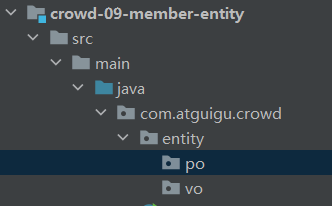
用途 1：从 Consumer 发送数据到 Provider

用途 2：Provider 返回数据给 Consumer

使用 org.springframework.beans.BeanUtils.copyProperties(Object, Object)在不同实体类之间复制属性。 MemberVO→复制属性→MemberPO



### 创建包



## MySQL工程

抽取整个项目中所有针对数据库的操作。

### 建表语句

成员信息表

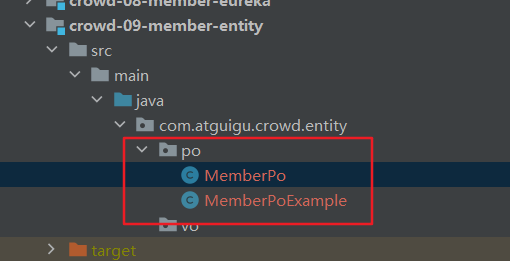
create table t\_member(  
 id int(11) not null auto\_increment,  
 loginacct varchar(255) not null,  
 userpswd varchar(255) not null,  
 username varchar(255),  
 email varchar(255),  
 authstatus int(4) comment '实名认证状态 0-未认证，1-申请中，2-已认证',  
 usertype int(4) comment '0-个人，1-企业',  
 realname varchar(255),  
 cardnum varchar(255),  
 accttype int(4) comment '0-企业，1-个体，2-个人，3-政府',  
 primary key(id)  
);

### 逆向工程

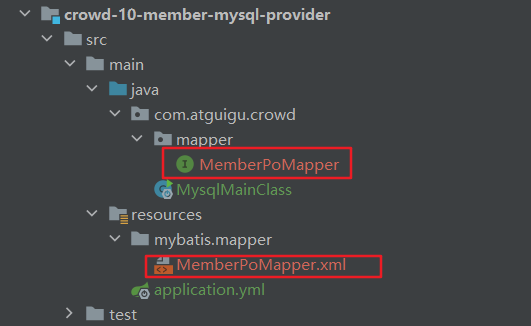
1.逆向工程，generatorConfig.xml配置文件修改表名和类名



2.实体类放到entity工程的po包下



3.mapper接口和mapper.xml放到mybatis包下



### 依赖

<dependencies>  
 <!-- 指定groupId和artifactId即可，版本已在父工程中定义 -->  
  
 <!--SpringBoot测试-->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 </dependency>  
 <!--MyBatis -->  
 <dependency>  
 <groupId>org.mybatis.spring.boot</groupId>  
 <artifactId>mybatis-spring-boot-starter</artifactId>  
 </dependency>  
 <!--mysql驱动-->  
 <dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
 </dependency>  
 <!--连接池-->  
 <dependency>  
 <groupId>com.alibaba</groupId>  
 <artifactId>druid</artifactId>  
 </dependency>  
 <!-- 对外暴露服务 -->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
 <!-- 作为客户端访问 Eureka 注册中心 -->  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>  
 </dependency>  
  
 <!-- 为了能够使用实体类 -->  
 <dependency>  
 <groupId>com.atguigu.crowd</groupId>  
 <artifactId>crowd-09-member-entity</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 </dependency>  
 <!-- 为了能够使用工具类 -->  
 <dependency>  
 <groupId>com.atguigu.crowd</groupId>

<artifactId>crowd-18-member-util</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 </dependency>  
  
</dependencies>

### 主启动类

注意加上@MapperScan注解，扫描mapper接口所在包

//扫描mapper接口所在包  
@MapperScan("com.atguigu.crowd.mapper")  
@SpringBootApplication  
public class CrowdMainClass {  
 public static void main(String[] args) {  
 SpringApplication.*run*(CrowdMainClass.class);  
 System.*out*.println("Hello world!");  
 }  
}

### application.yml

配置mybatis、数据源、eureka和日志等

server:  
 port: 2000  
spring:  
 application:  
 *#配置当前微服务名称，在Eureka注册中心，是通过微服务名称来查找调用微服务* name: mysql  
  
 *#数据源配置* datasource:  
 name: mydb  
 driver-class-name: com.mysql.cj.jdbc.Driver  
 type: com.alibaba.druid.pool.DruidDataSource  
 url: jdbc:mysql://localhost:3306/crowd?useUnicode=true&characterEncoding=UTF-8&serverTimezone=UTC  
 username: root  
 password: 123456  
*#MyBatis配置*mybatis:  
 mapper-locations: classpath:/mybatis/mapper/\*Mapper.xml  
*#日志*logging:  
 level:  
 com.atguigu.crowd.mapper: debug  
 com.atguigu.crowd.test: debug  
*#eureka配置*eureka:  
 client:  
 service-url:  
 *#Eureka客户端访问Eureka服务器时使用的地址* defaultZone: http://localhost:1000/eureka/

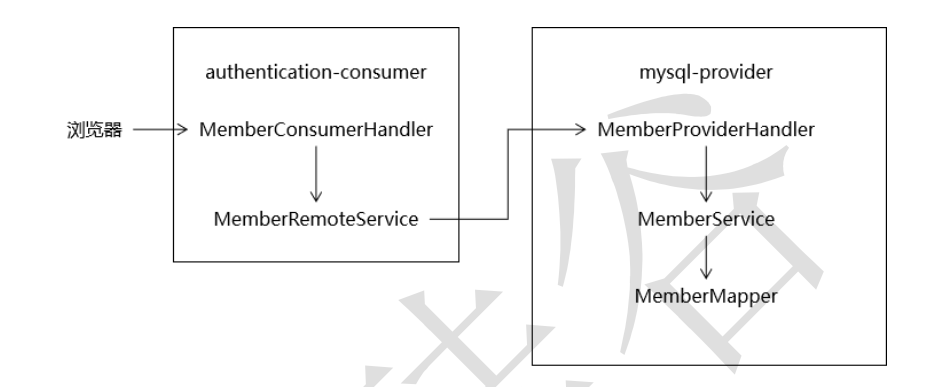
### 测试类

@RunWith(SpringRunner.class)  
@SpringBootTest  
public class MyBatisTest {  
 Logger logger = LoggerFactory.*getLogger*(MyBatisTest.class);  
  
 @Autowired  
 private DataSource dataSource;  
  
 @Autowired  
 private MemberPoMapper mapper;  
  
 @Test  
 public void testConnetion() throws SQLException {  
 Connection connection = dataSource.getConnection();  
 logger.info("============"+connection.toString());  
 }  
  
 @Test  
 public void testDatabase(){  
 MemberPo memberPo = mapper.selectByPrimaryKey(1001);  
 logger.info("======="+memberPo.toString());  
 }  
  
}

### MySQL工程对外服务

1.对外的服务放在mysql工程

2.接口和实现放在api工程



#### api工程

1.引入依赖

引入util工程是为了ResultEntity类，引入entity工程是为了MemberPo类

<dependencies>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-openfeign</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>com.atguigu.crowd</groupId>  
 <artifactId>crowd-05-admin-util</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 </dependency>  
 <dependency>  
 <groupId>com.atguigu.crowd</groupId>  
 <artifactId>crowd-09-member-entity</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 </dependency>  
</dependencies>

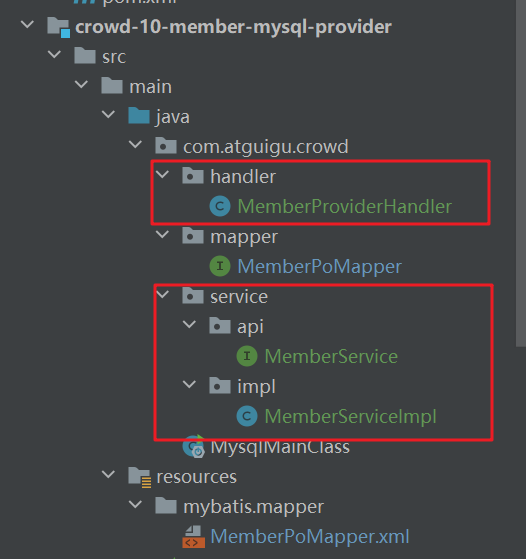
2.service类

注意要加 @FeignClient 注解，远程接口要和provider工程的方法保持一致

即要和mysql-provider工程中的控制器的方法一致

//@FeignClient注解表示和一个provider对应，value属性指定要调用的Provider的微服务名称;  
// fallbackFactory属性指定 consumer 调用 provider 时如果失败所采取的备用方案  
@FeignClient(value = "mysql-provider")  
public interface MySQLRemoteService {  
 //远程调用的接口方法  
 //注意：这里的方法要和 Provider中的具体实现的方法的声明要一致  
 //@RequestMapping注解的url地址、@RequestParam、@RequestBody、@PathVariable两边一致  
 @RequestMapping("/get/member/by/loginacct/remote")  
 ResultEntity<MemberPo> getMemberByLoginacctRemote(@RequestParam("loginacct") String loginacct);  
}

#### mysql工程



1.控制器方法

@RestController  
public class MemberProviderHandler {  
 @Autowired  
 private MemberService memberService;  
  
 //provider工程的远程方法，要和api工程有 @FeignClient 注解标记的方法一致  
 @RequestMapping("/get/member/by/loginacct/remote")  
 ResultEntity<MemberPo> getMemberByLoginacctRemote(@RequestParam("loginacct") String loginacct){  
 //*todo 异常处理* MemberPo member = memberService.getMemberByLoginacct(loginacct);  
 return ResultEntity.*successWithData*(member);  
 }  
}

2.service接口

public interface MemberService {  
  
 MemberPo getMemberByLoginacct(String loginacct);  
}

3.service实现类，加上事务控制

// 在类上使用@Transactional(readOnly = true)针对查询操作设置事务属性  
@Transactional(readOnly = true)  
@Service  
public class MemberServiceImpl implements MemberService {  
 @Autowired  
 private MemberPoMapper mapper;  
  
 @Override  
 public MemberPo getMemberByLoginacct(String loginacct) {  
 return mapper.selectByLoginAcct(loginacct);  
 }  
}

4.mapper接口

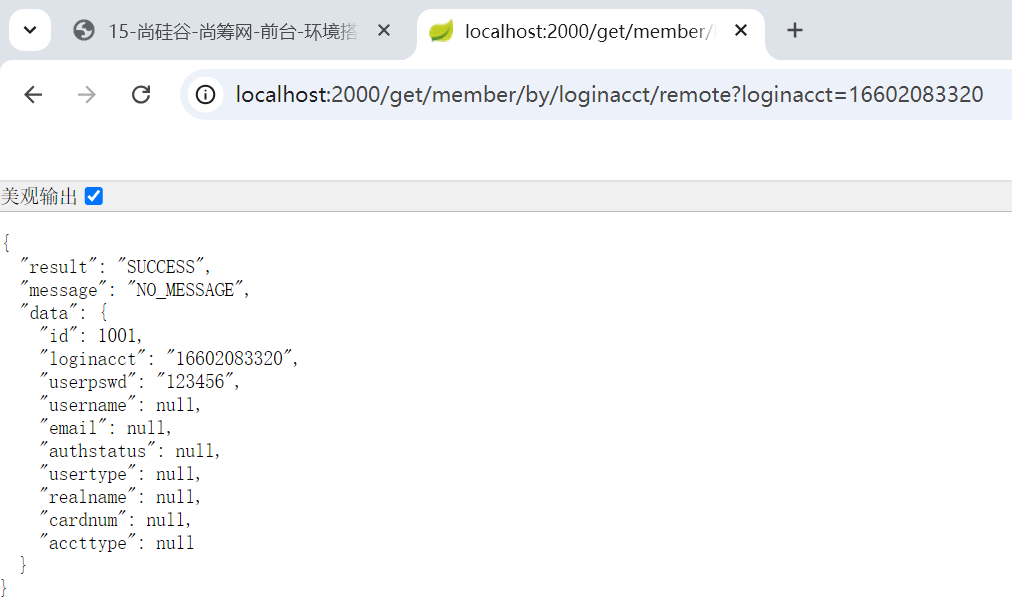
*/\*\*  
\* 根据登录账号查询成员信息  
\** ***@param*** *loginAcct  
\** ***@return****\*/*MemberPo selectByLoginAcct(@Param("loginAcct") String loginAcct);

5.mapper.xml的sql

<select id="selectByLoginAcct" parameterType="java.lang.String" resultMap="BaseResultMap">  
 select  
 <include refid="Base\_Column\_List" />  
 from t\_member  
 where loginacct = #{loginAcct}  
</select>

#### 测试

http://localhost:2000/get/member/by/loginacct/remote?loginacct=16602083320



## Redis工程

抽取项目中所有访问 Redis 的操作。

### 依赖

<dependencies>  
 <!-- 整合 Redis -->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-redis</artifactId>  
 </dependency>  
 <!-- 测试 -->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
 <!-- 对外暴露服务 -->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
 <!-- 作为客户端访问 Eureka 注册中心 -->  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>  
 </dependency>  
 <!-- 为了能够使用实体类 -->  
 <dependency>  
 <groupId>com.atguigu.crowd</groupId>  
 <artifactId>crowd-09-member-entity</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 </dependency>  
 <!-- 为了能够使用工具类 -->  
 <dependency>  
 <groupId>com.atguigu.crowd</groupId>  
 <artifactId>crowd-18-member-util</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 </dependency>  
</dependencies>

### 主启动类

@SpringBootApplication  
public class CrowdMainClass {  
 public static void main(String[] args) {  
 SpringApplication.*run*(CrowdMainClass.class);  
 System.*out*.println("Hello world!");  
 }  
}

### application.yml

主要配置：微服务名称、redis、eureka

server:  
 port: 3000  
  
spring:  
 application:  
 *#配置当前微服务名称，在Eureka注册中心，是通过微服务名称来查找调用微服务* name: redis-provider  
  
 *#redis相关配置* redis:  
 *#默认使用 0 号数据库* database: 0  
 *#redis服务器地址，如果redis的配置文件的 bind属性绑定了地址，需要用该地址* host: 192.168.64.133  
 *#rdis默认端口6379* port: 6379  
 password: 123456  
 timeout: 5000  
 jedis:  
 pool:  
 *#连接池中最大空闲连接，默认值是8* max-idle: 500  
 *#连接池中的最小空闲连接，默认值是0* min-idle: 50  
 *#如果赋值为 -1，则表示不限制；如果pool已经分配了maxActive个jedis实例，则此时pool的状态为exhansted（耗尽）* max-active: 1000  
 *#等待可用连接的最大时间，单位毫秒，默认值为 -1，表示永不超时。如果超过等待时间，则直接抛出 JedisConnectionException* max-wait: 2000  
  
*#eureka配置*eureka:  
 client:  
 service-url:  
 *#Eureka客户端访问Eureka服务器时使用的地址* defaultZone: http://localhost:1000/eureka/

### 测试类

@RunWith(SpringRunner.class)  
@SpringBootTest  
public class RedisTest {  
 Logger logger = LoggerFactory.*getLogger*(RedisTest.class);  
 @Autowired  
 private StringRedisTemplate redisTemplate;  
 @Test  
 public void testRedis(){  
 ValueOperations<String, String> operations = redisTemplate.opsForValue();  
 operations.set("name1","shuyun");  
 }  
}

### redis工程对外服务

#### api工程接口

提供外部服务：string类型的get、set方法，分带超时时间和不带超时时间的方法，以及删除key的操作

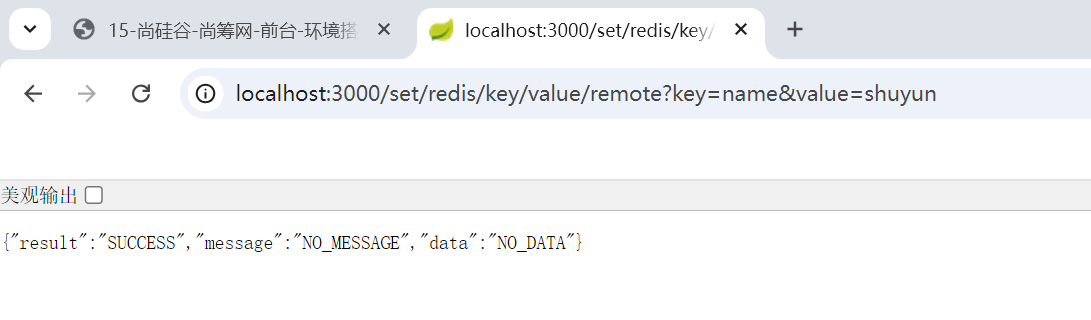
*/\*\*  
\** ***@author*** *shuyun  
\** ***@date*** *2024-10-11 08:52:27  
\** ***@FeignClient注解表示和一个provider对应，value属性指定要调用的Provider的微服务名称;****\* fallbackFactory属性指定 consumer 调用 provider 时如果失败所采取的备用方案  
 \*/*@FeignClient(value = "redis-provider")  
public interface RedisRemoteService {  
  
 */\*\*  
 \* redis操作，不带超时的set方法  
 \*  
 \** ***@param*** *key  
 \** ***@param*** *value  
 \** ***@return*** *\*/* @RequestMapping("/set/redis/key/value/remote")  
 ResultEntity<String> setRedisKeyValueRemote(@RequestParam("key") String key, @RequestParam("value") String value);  
  
  
 */\*\*  
 \* redis操作，带超时的set方法  
 \** ***@param*** *key  
 \** ***@param*** *value  
 \** ***@param*** *time  
 \** ***@param*** *timeUnit  
 \** ***@return*** *\*/* @RequestMapping("/set/redis/key/value/remote/with/timeout")  
 ResultEntity<String> setRedisKeyValueRemoteWithTimeout(  
 @RequestParam("key") String key,  
 @RequestParam("value") String value,  
 @RequestParam("time") long time,  
 @RequestParam("timeUnit") TimeUnit timeUnit);  
  
 */\*\*  
 \* reddis操作，根据key获取value  
 \** ***@param*** *key  
 \** ***@return*** *\*/* @RequestMapping("/get/redis/value/by/key/remote")  
 ResultEntity<String> getRedisValueByKeyRemote(@RequestParam("key") String key);  
  
 */\*\*  
 \* redis操作，删除key  
 \** ***@param*** *key  
 \** ***@return*** *\*/* @RequestMapping("/remove/redis/key/remote")  
 ResultEntity<String> removeRedisKeyRemote(@RequestParam("key") String key);  
}

#### redis工程handler

@RestController  
public class RedisHandler {  
  
 @Autowired  
 private StringRedisTemplate redisTemplate;  
  
 */\*\*  
 \* redis操作，不带超时的set方法  
 \*  
 \** ***@param*** *key  
 \** ***@param*** *value  
 \** ***@return*** *\*/* @RequestMapping("/set/redis/key/value/remote")  
 ResultEntity<String> setRedisKeyValueRemote(@RequestParam("key") String key, @RequestParam("value") String value){  
 try{  
 ValueOperations<String, String> operations = redisTemplate.opsForValue();  
 operations.set(key,value);  
 }catch (Exception e){  
 return ResultEntity.*failed*(e.getMessage());  
 }  
 return ResultEntity.*successWithoutData*();  
 }  
  
  
 */\*\*  
 \* redis操作，带超时的set方法  
 \** ***@param*** *key  
 \** ***@param*** *value  
 \** ***@param*** *time  
 \** ***@param*** *timeUnit  
 \** ***@return*** *\*/* @RequestMapping("/set/redis/key/value/remote/with/timeout")  
 ResultEntity<String> setRedisKeyValueRemoteWithTimeout(  
 @RequestParam("key") String key,  
 @RequestParam("value") String value,  
 @RequestParam("time") long time,  
 @RequestParam("timeUnit") TimeUnit timeUnit){  
  
 try{  
 ValueOperations<String, String> operations = redisTemplate.opsForValue();  
 operations.set(key,value,time,timeUnit);  
 }catch (Exception e){  
 return ResultEntity.*failed*(e.getMessage());  
 }  
 return ResultEntity.*successWithoutData*();  
 }  
  
 */\*\*  
 \* reddis操作，根据key获取value  
 \** ***@param*** *key  
 \** ***@return*** *\*/* @RequestMapping("/get/redis/value/by/key/remote")  
 ResultEntity<String> getRedisValueByKeyRemote(@RequestParam("key") String key){  
 try{  
 ValueOperations<String, String> operations = redisTemplate.opsForValue();  
 String value = operations.get(key);  
 return ResultEntity.*successWithData*(value);  
 }catch (Exception e){  
 return ResultEntity.*failed*(e.getMessage());  
 }  
 }  
  
 */\*\*  
 \* redis操作，删除key  
 \** ***@param*** *key  
 \** ***@return*** *\*/* @RequestMapping("/remove/redis/key/remote")  
 ResultEntity<String> removeRedisKeyRemote(@RequestParam("key") String key){  
 try{  
 redisTemplate.delete(key);  
 return ResultEntity.*successWithoutData*();  
 }catch (Exception e){  
 return ResultEntity.*failed*(e.getMessage());  
 }  
 }  
}

#### 测试

http://localhost:3000/set/redis/key/value/remote?key=name&value=shuyun



## 认证工程

## 网关

# 会员登录

# 发起项目

# 展示项目

# 支付案例

# 订单

# 支付

# 部署

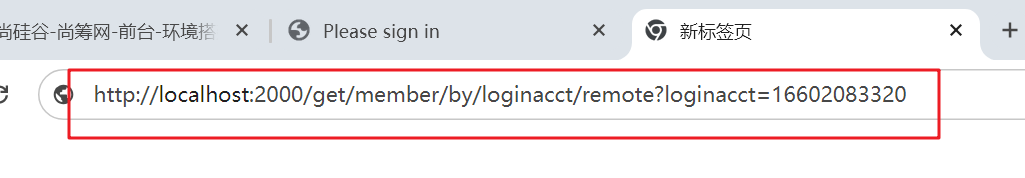
# 总结

# 出错汇总

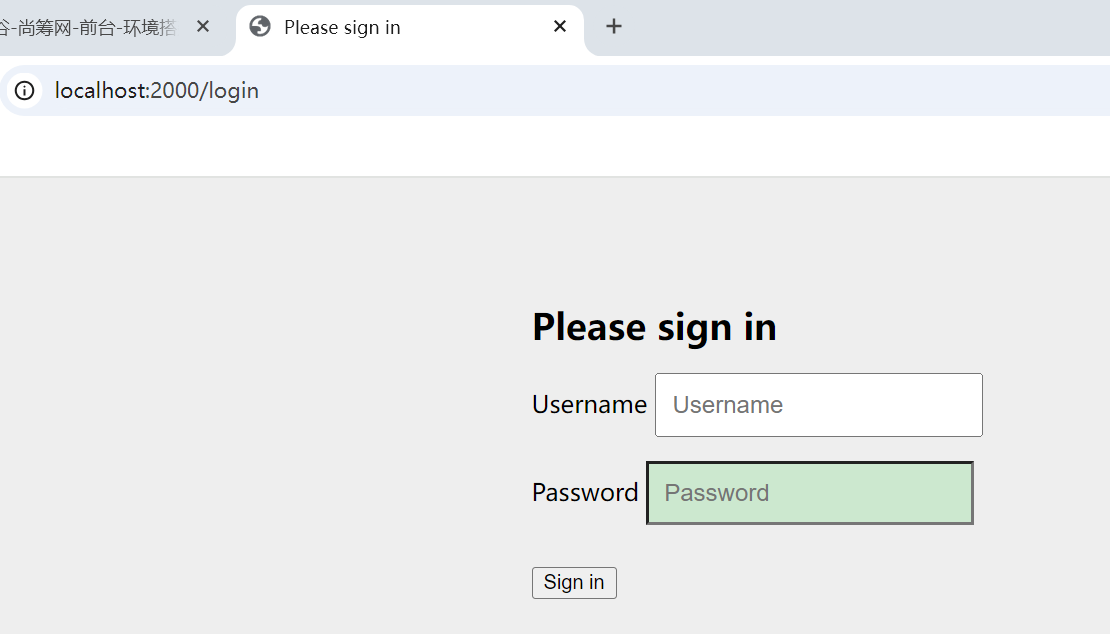
## 环境搭建

搭建mysql和redis工程的时候，单元测试没问题，在浏览器通过控制器方法测试时，请求被拦截了，要求登录

输入这个地址



跳转到了这里



出错原因：Mysql工程和redis工程引入的resultentity所在的util工程是后台的工程，这个util工程的依赖是全依赖于其父工程依赖（完全引用），导致引入SpringSecurity，所以就拦截请求，要求登录后访问。

修改：新建一个util工程，新建resultentity类，Mysql工程和redis工程引入新的util工程

