尚筹网

[15-尚硅谷-尚筹网-前台-环境搭建]

1 忽略 IDE 工具导入 Maven 工程

1.1 为什么要忽略 IDE 环境

开发的时候团队成员会因为不同的习惯、喜好使用不同 IDE 工具。而不同 IDE 环境之间又存在一定差异。

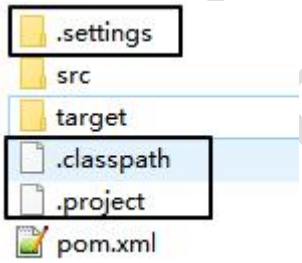
所以在不同 IDE 工具之间导入工程有可能发生错误。所以最好能够不受具体 IDE 工具的干扰。

1.2 操作

以 Eclipse 为例。

1.2.1 擦除痕迹

导入工程前先删除以前的 Eclipse 工程的痕迹。



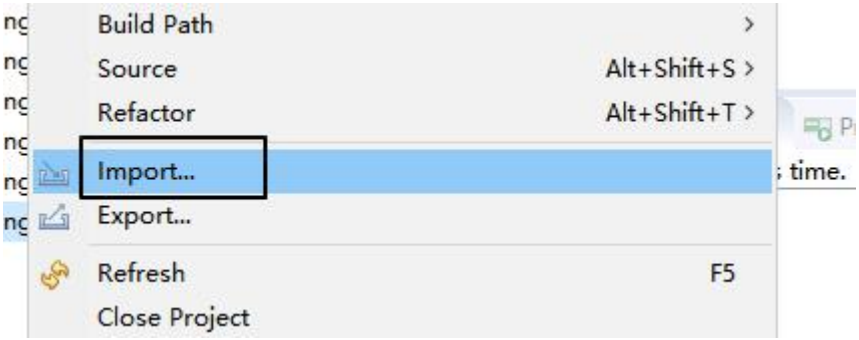
target 目录也可以删除，让工程导入进来以后重新编译。

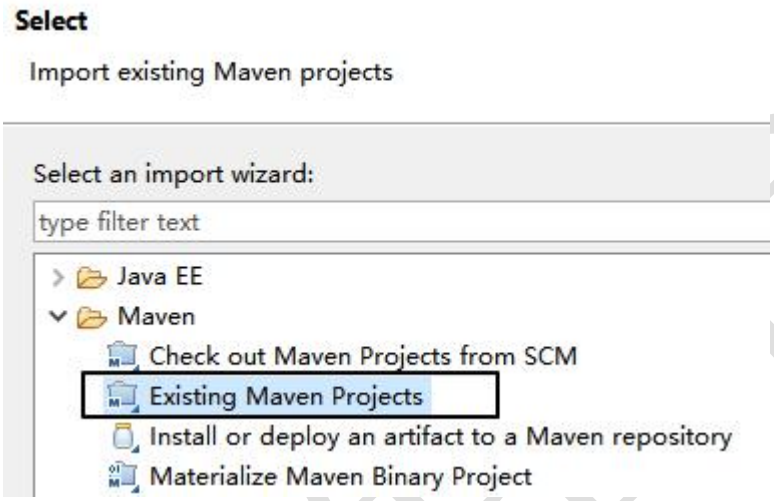
1.2.2 复制

把要导入的工程复制到当前工作区。原因是：以 Maven 方式导入工程不会自动复制到工作区，真正的目录和文件还是在原来的地方。

1.2.3 导入

然后以 Maven 工程方式导入。



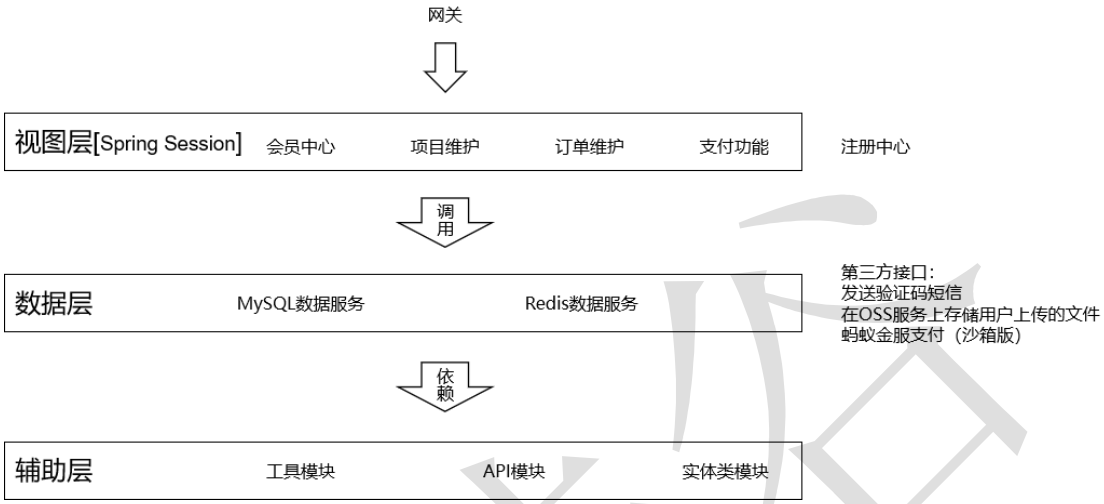


2 尚筹网会员系统总目标

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

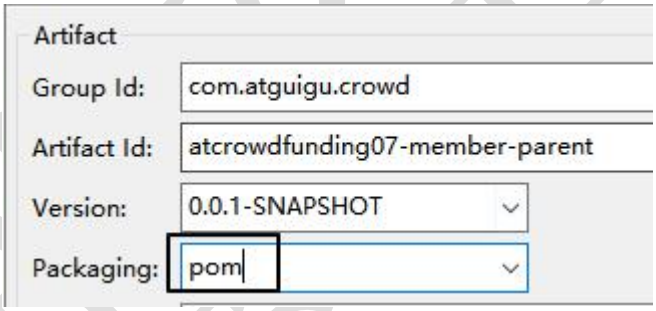
3 会员系统架构

3.1 架构图



3.2 需要创建的工程

父工程、聚合工程：atcrowdfunding07-member-parent（唯一的 pom 工程）



注册中心：atcrowdfunding08-member-eureka
实体类模块：atcrowdfunding09-member-entity
MySQL 数据服务：atcrowdfunding10-member-mysql-provider
Redis 数据服务：atcrowdfunding11-member-redis-provider
会员中心：atcrowdfunding12-member-authentication-consumer
项目维护：atcrowdfunding13-member-project-consumer
订单维护：atcrowdfunding14-member-order-consumer
支付功能：atcrowdfunding15-member-pay-consumer
网关：atcrowdfunding16-member-zuul
API 模块：atcrowdfunding17-member-api

4 parent 工程配置 pom.xml

```
<!-- 配置在父工程中要管理的依赖 -->
<dependencyManagement>
    <dependencies>
        <!-- 导入 SpringCloud 需要使用的依赖信息 -->
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>Greenwich.SR2</version>
            <type>pom</type>
            <!-- import 依赖范围表示将 spring-cloud-dependencies 包中的依赖信息导入 -->
            <scope>import</scope>
        </dependency>
        <!-- 导入 SpringBoot 需要使用的依赖信息 -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-dependencies</artifactId>
            <version>2.1.6.RELEASE</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
        <dependency>
            <groupId>org.mybatis.spring.boot</groupId>
            <artifactId>mybatis-spring-boot-starter</artifactId>
            <version>2.1.0</version>
        </dependency>
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>druid</artifactId>
            <version>1.0.5</version>
        </dependency>
    </dependencies>
</dependencyManagement>
```

5 搭建环境约定

5.1 包名约定

新创建的包都作为 com.atguigu.crowd 的子包

5.2 主启动类类名

CrowdMainClass

5.3 端口号

atcrowdfunding08-member-eureka 1000
atcrowdfunding10-member-mysql-provider 2000
atcrowdfunding11-member-redis-provider 3000
atcrowdfunding12-member-authentication-consumer 4000
atcrowdfunding13-member-project-consumer 5000
atcrowdfunding14-member-order-consumer 7000
atcrowdfunding15-member-pay-consumer 8000
atcrowdfunding16-member-zuul 80

6 eureka 工程

6.1 依赖

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

6.2 主启动类

```
@EnableEurekaServer
@SpringBootApplication
public class CrowdMainClass {

    public static void main(String[] args) {
        SpringApplication.run(CrowdMainClass.class, args);
    }
```

```
}
```

6.3 application.yml

```
server:
  port: 1000
spring:
  application:
    name: atguigu-crowd-eureka
eureka:
  instance:
    hostname: localhost
  client:
    register-with-eureka: false
    fetch-registry: false
    service-url:
      defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/
```

7 entity 工程

7.1 实体类的进一步细分

- 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

- 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

- 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

  ……

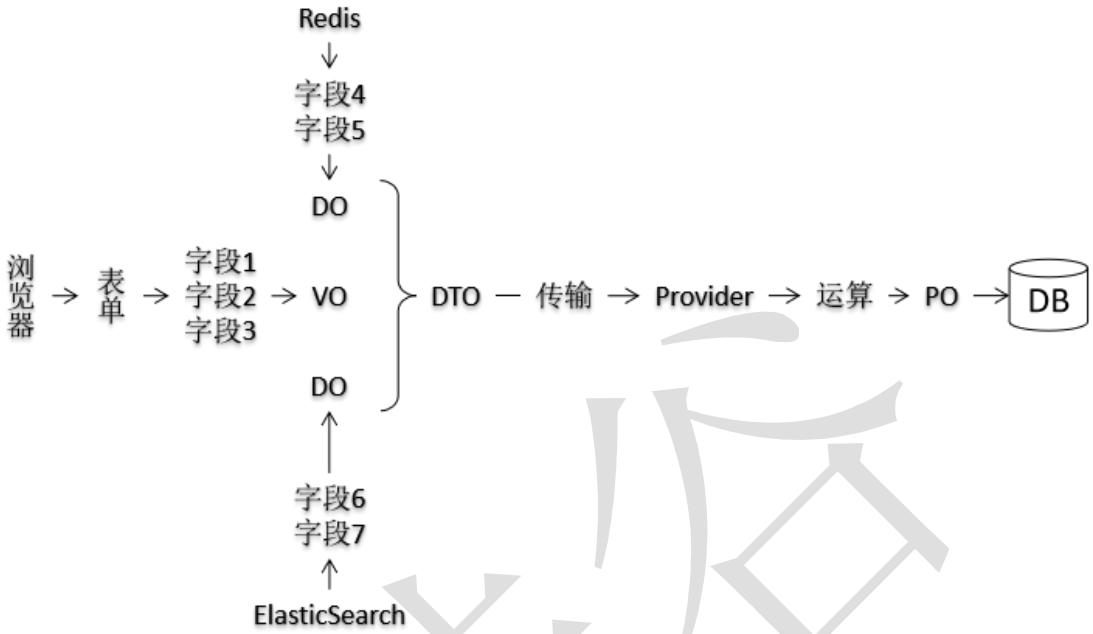
  从中间件或其他第三方接口查询到的数据封装为 DO 对象

- 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

7.2 创建包

com.atguigu.crowd.entity.po

com.atguigu.crowd.entity.vo

7.3 lombok

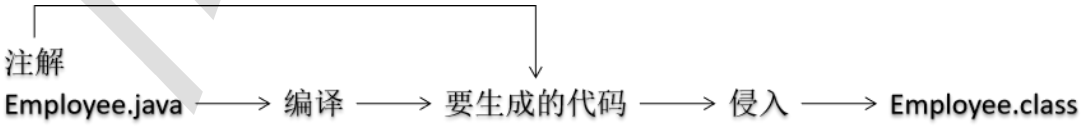
7.3.1 效果

```java
@Data
@NoArgsConstructor
@AllArgsConstructor
@EqualsAndHashCode
public class Employee {

    private Integer empId;
    private String empName;
    private Double empPrice;

    public static void main(String[] args) {
        Employee employee = new Employee();
        employee = new Employee(1, "tom", 100.00);
        employee.setEmpName("Jerry");
        employee.getEmpPrice();
        employee.hashCode();
        employee.equals(employee);
    }

}
```

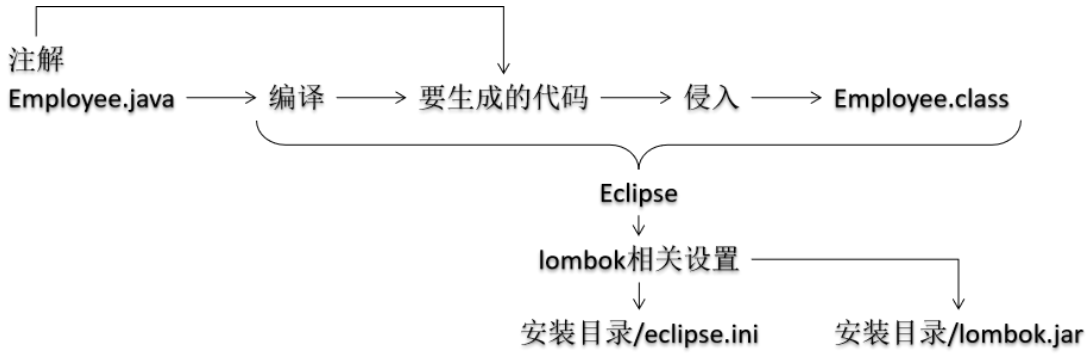
让我们在开发时不必编写 getXxx()、setXxx()、有参构造器、无参构造器等等这样具备固定模式的代码。

7.3.2 lombok 原理

根据注解确定要生成的代码，然后将要生成的代码侵入到字节码文件中。



7.3.3 Eclipse 设置



7.3.3.1 原生 Eclipse

双击执行 lombok-1.16.22.jar

7.3.3.2 STS

- 第一步：找到 STS 安装根目录/STS.ini
  - 在最后一行加入-javaagent:lombok.jar
- 第二步：将 lombok 的 jar 包复制到 STS 的根目录下
- 第三步：将 lombok 的 jar 包重命名为 lombok.jar
- 第四步：重启 STS

7.3.4 注解

- @Data：每一个字段都加入 getXxx()、setXxx()方法
- @NoArgsConstructor：无参构造器
- @AllArgsConstructor：全部字段都包括的构造器
- @EqualsAndHashCode：equals 和 hashCode 方法
- @Getter
  - 类：所有字段都加入 getXxx()方法
  - 字段：当前字段加入 getXxx()方法
- @Setter
  - 类：所有字段都加入 setXxx()方法
  - 字段：当前字段加入 setXxx()方法

8 MySQL 工程基础环境

8.1 目标

抽取整个项目中所有针对数据库的操作。

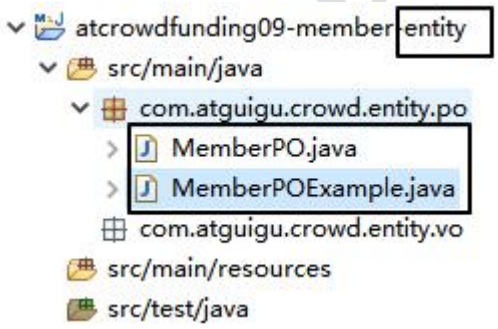
8.2 创建数据库表

```
create table t_member
(
    id                          int(11) not null auto_increment,
```

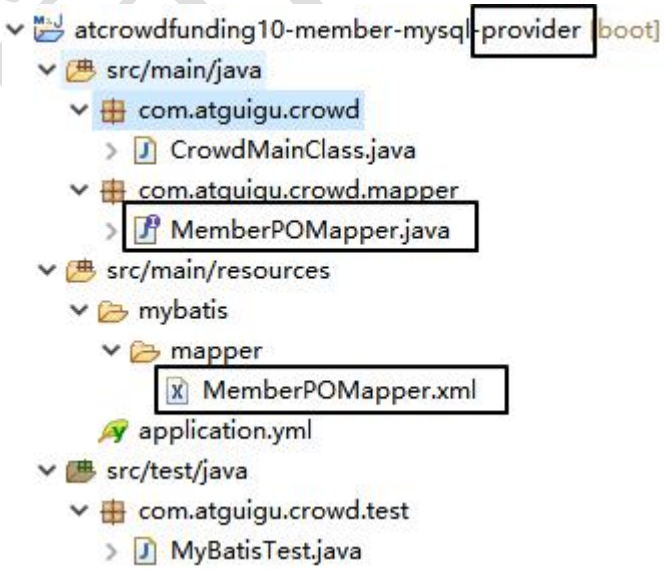
```
    loginacct          varchar(255) not null,
    userpswd           char(200) not null,
    username           varchar(255),
    email              varchar(255),
    authstatus         int(4) comment '实名认证状态 0 - 未实名认证， 1 - 实名认证申请中， 2 - 已实名认证',
    usertype           int(4) comment ' 0 - 个人， 1 - 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)
);
```

8.3 逆向生成

8.3.1 实体类归位



8.3.2 Mapper 相关归位



8.4 依赖

```
<!-- 整合 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>

<!-- 数据库连接池 -->
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>druid</artifactId>
</dependency>

<!-- SpringBoot 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>atcrowdfunding09-member-entity</artifactId>
```

```
        <version>0.0.1-SNAPSHOT</version>
</dependency>

<!-- 为了能够使用工具类 -->
<dependency>
        <groupId>com.atguigu.crowd</groupId>
        <artifactId>atcrowdfunding05-common-util</artifactId>
        <version>0.0.1-SNAPSHOT</version>
</dependency>
```

8.5 创建主启动类

```
// 扫描 MyBatis 的 Mapper 接口所在的包
@MapperScan("com.atguigu.crowd.mapper")
@SpringBootApplication
public class CrowdMainClass {

        public static void main(String[] args) {
                SpringApplication.run(CrowdMainClass.class, args);
        }

}
```

8.6 application.yml

```
server:
    port: 2000
spring:
    application:
        name: atguigu-crowd-mysql
    datasource:
        name: mydb
        type: com.alibaba.druid.pool.DruidDataSource
        url: jdbc:mysql://127.0.0.1:3306/project_crowd?serverTimezone=UTC
        username: root
        password: root
        driver-class-name: com.mysql.cj.jdbc.Driver
eureka:
    client:
        service-url:
            defaultZone: http://localhost:1000/eureka
mybatis:
```

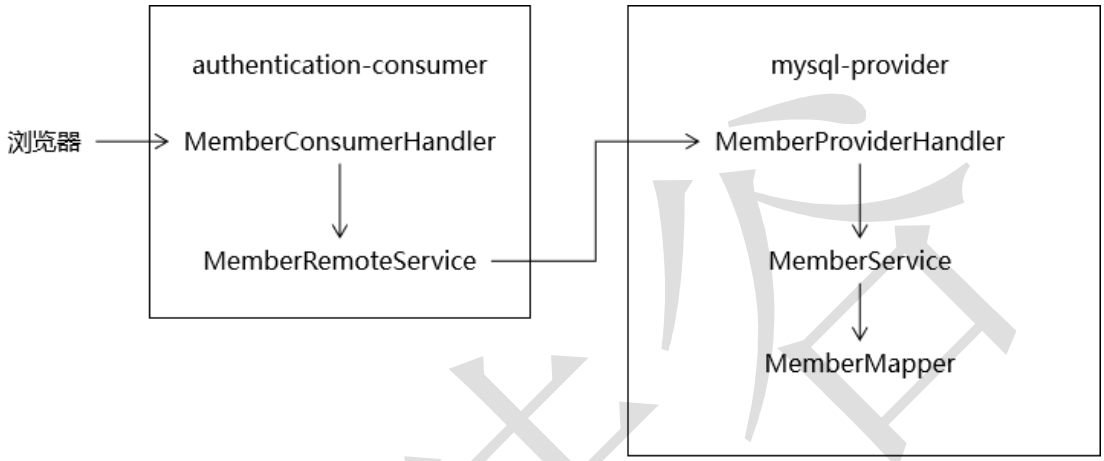
```
  mapper-locations: classpath*:/mybatis/mapper/*Mapper.xml
logging:
  level:
    com.atguigu.crowd.mapper: debug
    com.atguigu.crowd.test: debug
```

8.7 测试类

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class MyBatisTest {

    @Autowired
    private DataSource dataSource;

    @Autowired
    private MemberPOMapper memberPOMapper;

    private Logger logger = LoggerFactory.getLogger(MyBatisTest.class);

    @Test
    public void testMapper() {

        BCryptPasswordEncoder passwordEncoder = new BCryptPasswordEncoder();

        String source = "123123";

        String encode = passwordEncoder.encode(source);

        MemberPO memberPO = new MemberPO(null, "jack", encode, "杰克",
"jack@qq.com", 1, 1, "杰克", "123123", 2);

        memberPOMapper.insert(memberPO);
    }

    @Test
    public void testConnection() throws SQLException {

        Connection connection = dataSource.getConnection();

        logger.debug(connection.toString());
    }
```

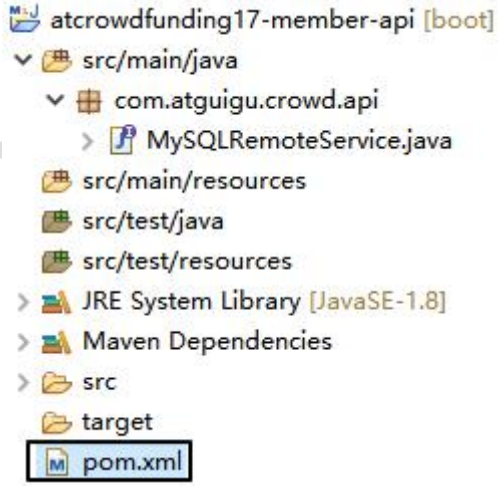
```
}
```

9 MySQL 工程对外暴露服务



9.1 api 工程

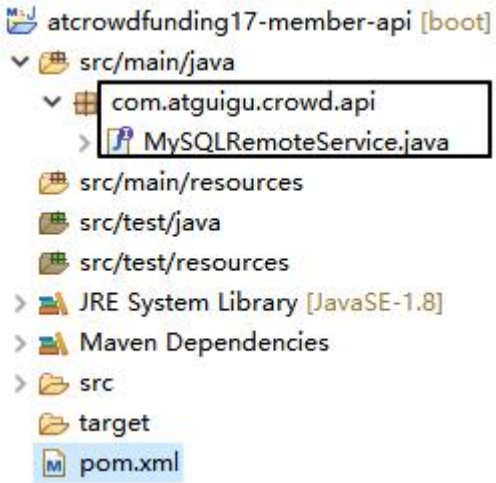
9.1.1 依赖




```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-openfeign</artifactId>
</dependency>
<dependency>
    <groupId>com.atguigu.crowd</groupId>
    <artifactId>atcrowdfunding05-common-util</artifactId>
```

```
        <version>0.0.1-SNAPSHOT</version>
</dependency>
<dependency>
        <groupId>com.atguigu.crowd</groupId>
        <artifactId>atcrowdfunding09-member-entity</artifactId>
        <version>0.0.1-SNAPSHOT</version>
</dependency>
```

9.1.2 创建接口




```
@FeignClient("atguigu-crowd-mysql")
public interface MySQLRemoteService {

        @RequestMapping("/get/memberpo/by/login/acct/remote")
        ResultEntity<MemberPO>    getMemberPOByLoginAcctRemote(@RequestParam("loginacct")
String loginacct);

}
```

9.2 MySQL 工程

9.2.1 创建组件

9.2.2 Handler 代码

```
@RestController
public class MemberProviderHandler {

    @Autowired
    private MemberService memberService;

    @RequestMapping("/get/memberpo/by/login/acct/remote")
    public ResultEntity<MemberPO> getMemberPOByLoginAcctRemote(@RequestParam("loginacct") String loginacct) {

        try {

            // 1.调用本地 Service 完成查询
            MemberPO memberPO = memberService.getMemberPOByLoginAcct(loginacct);

            // 2.如果没有抛异常，那么就返回成功的结果
            return ResultEntity.successWithData(memberPO);
        } catch (Exception e) {
            e.printStackTrace();

            // 3.如果捕获到异常则返回失败的结果
            return ResultEntity.failed(e.getMessage());
        }

    }
```

```
}
```

9.2.3 Service 代码

```
// 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 memberPOMapper;

    @Override
    public MemberPO getMemberPOByLoginAcct(String loginacct) {

        // 1.创建 Example 对象
        MemberPOExample example = new MemberPOExample();

        // 2.创建 Criteria 对象
        Criteria criteria = example.createCriteria();

        // 3.封装查询条件
        criteria.andLoginacctEqualTo(loginacct);

        // 4.执行查询
        List<MemberPO> list = memberPOMapper.selectByExample(example);

        // 5.获取结果
        return list.get(0);
    }

}
```

10 Redis 工程基础环境

10.1 目标

抽取项目中所有访问 Redis 的操作。

10.2 依赖

```
<!-- 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>atcrowdfunding09-member-entity</artifactId>
    <version>0.0.1-SNAPSHOT</version>
</dependency>

<!-- 为了能够使用工具类 -->
<dependency>
    <groupId>com.atguigu.crowd</groupId>
    <artifactId>atcrowdfunding05-common-util</artifactId>
    <version>0.0.1-SNAPSHOT</version>
</dependency>
```

10.3 主启动类

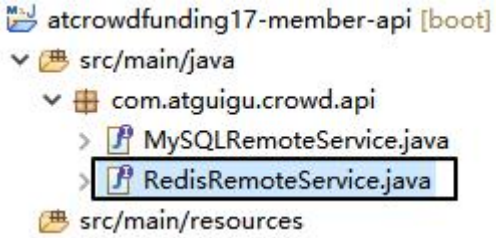
```
@SpringBootApplication
public class CrowdMainClass {

	public static void main(String[] args) {
		SpringApplication.run(CrowdMainClass.class, args);
	}

}
```

10.4 测试类

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class RedisTest {

	// private Logger logger = LoggerFactory.getLogger(RedisTest.class);

	@Autowired
	private StringRedisTemplate redisTemplate;

	@Test
	public void testSet() {
		ValueOperations<String, String> operations = redisTemplate.opsForValue();

		operations.set("apple", "red");
	}

}
```

11 Redis 工程对外暴露服务

11.1 api 工程创建接口



```
@FeignClient("atguigu-crowd-redis")
public interface RedisRemoteService {

    @RequestMapping("/set/redis/key/value/remote")
    ResultEntity<String> setRedisKeyValueRemote(
            @RequestParam("key") String key,
            @RequestParam("value") String value);

    @RequestMapping("/set/redis/key/value/remote/with/timeout")
    ResultEntity<String> setRedisKeyValueRemoteWithTimeout(
            @RequestParam("key") String key,
            @RequestParam("value") String value,
            @RequestParam("time") long time,
            @RequestParam("timeUnix") TimeUnit timeUnit);

    @RequestMapping("/get/redis/string/value/by/key")
    ResultEntity<String> getRedisStringValueByKeyRemote(@RequestParam("key") String key);

    @RequestMapping("/remove/redis/key/remote")
    ResultEntity<String> removeRedisKeyRemote(@RequestParam("key") String key);

}
```

11.2 Redis 工程 handler 代码

```
@RestController
public class RedisHandler {

    @Autowired
    private StringRedisTemplate redisTemplate;

    @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);

            return ResultEntity.successWithoutData();
```

```
        } catch (Exception e) {
            e.printStackTrace();

            return ResultEntity.failed(e.getMessage());
        }

    }

    @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);

            return ResultEntity.successWithoutData();
        } catch (Exception e) {
            e.printStackTrace();

            return ResultEntity.failed(e.getMessage());
        }
    }

    @RequestMapping("/get/redis/string/value/by/key")
    ResultEntity<String> getRedisStringValueByKeyRemote(@RequestParam("key") String key) {

        try {
            ValueOperations<String, String> operations = redisTemplate.opsForValue();

            String value = operations.get(key);

            return ResultEntity.successWithData(value);
        } catch (Exception e) {
            e.printStackTrace();

            return ResultEntity.failed(e.getMessage());
        }
```

```
    }

    @RequestMapping("/remove/redis/key/remote")
    ResultEntity<String> removeRedisKeyRemote(@RequestParam("key") String key) {
        try {

            redisTemplate.delete(key);

            return ResultEntity.successWithoutData();

        } catch (Exception e) {
            e.printStackTrace();

            return ResultEntity.failed(e.getMessage());
        }
    }

}
```

12 认证工程显示首页

12.1 依赖


```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
```

```
        <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
    </dependency>
    <dependency>
        <groupId>com.atguigu.crowd</groupId>
        <artifactId>atcrowdfunding17-member-api</artifactId>
        <version>0.0.1-SNAPSHOT</version>
    </dependency>
```

12.2 主启动类

```
@EnableDiscoveryClient //  当前版本可以不写
@SpringBootApplication
public class CrowdMainClass {

    public static void main(String[] args) {
        SpringApplication.run(CrowdMainClass.class, args);
    }

}
```

12.3 application.yml

```
server:
  port: 4000
spring:
  application:
    name: atguigu-crowd-auth
  thymeleaf:
    prefix: classpath:/templates/
    suffix: .html
eureka:
  client:
    service-url:
      defaultZone: http://localhost:1000/eureka
```

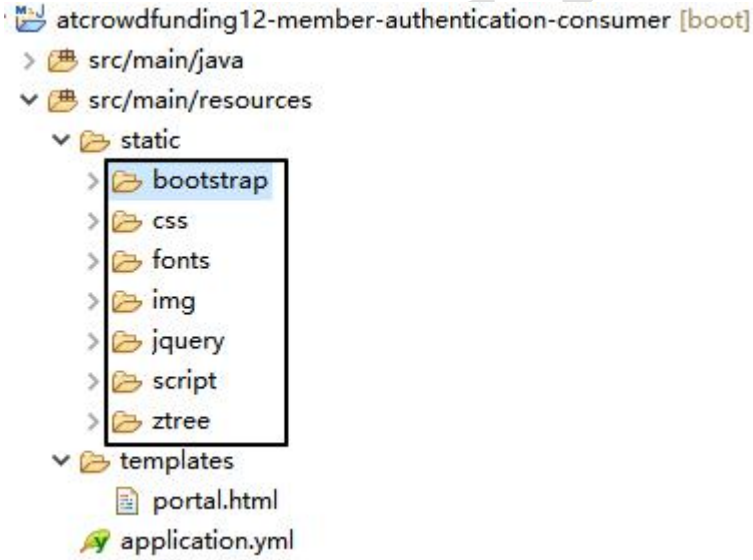
12.4 显示首页的 handler

```
@Controller
```

```
public class PortalHandler {

    @RequestMapping("/")
    public String showPortalPage() {

        // 这里实际开发中需要加载数据……

        return "portal";
    }

}
```

12.5加入静态资源

12.5.1静态资源加入的位置



SpringBoot 要求在 static 目录下存放静态资源。

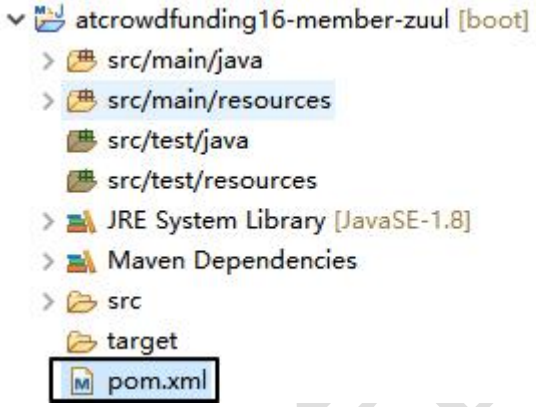
12.5.2调整 portal.html 页面

```
<html lang="zh-CN" xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1">
<meta name="description" content="">
<meta name="author" content="">
<base th:href="@{/}"/>
<link rel="stylesheet" href="bootstrap/css/bootstrap.min.css">
```

```
<link rel="stylesheet" href="css/font-awesome.min.css">
<link rel="stylesheet" href="css/carousel.css">
```

13 网关

13.1 依赖




```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-zuul</artifactId>
</dependency>
```

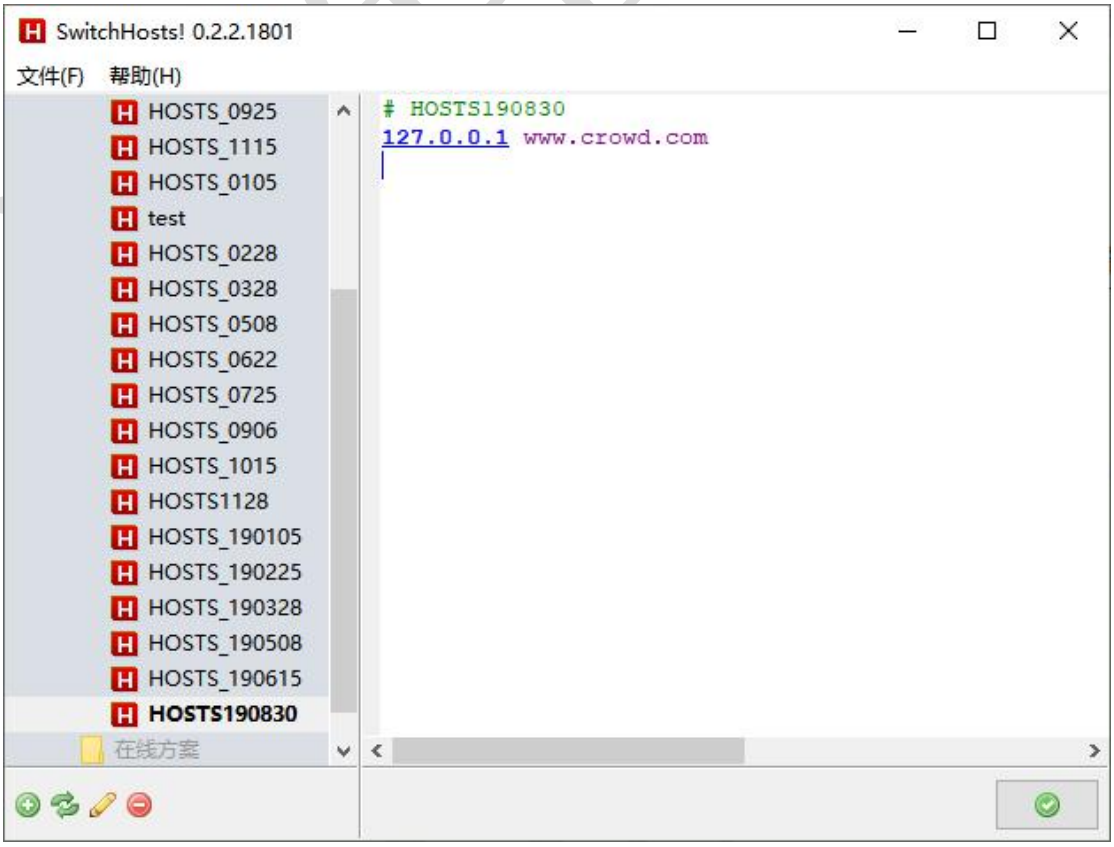
13.2 主启动类

```
@EnableZuulProxy
@SpringBootApplication
public class CrowdMainClass {

    public static void main(String[] args) {
        SpringApplication.run(CrowdMainClass.class, args);
    }

}
```

13.3application.yml

```
server:
  port: 80
spring:
  application:
    name: atguigu-crowd-zuul
eureka:
  client:
    service-url:
      defaultZone: http://localhost:1000/eureka
zuul:
  ignored-services: "*"
  sensitive-headers: "*"   # 在 Zuul 向其他微服务重定向时保持原本头信息（请求头、响应头）
  routes:
    crowd-portal:
      service-id: atguigu-crowd-auth
      path: /** # 这里一定要使用两个“*”号，不然“/”路径后面的多层路径将无法访问
```

13.4配置域名（可选）



13.5访问效果

13.6