

课程内容

- 使用Docker安装MySQL
- Mybatis插件MybatisPlus的入门
- 搭建后台服务系统
- 实现新增房源服务
- 前后端进行整合，实现新增房源功能

1、使用Docker安装MySQL

好客租房项目的底层数据库采用MySQL，而MySQL采用衍生版本Percona，并且采用docker容器化的方式进行部署。

1.1、什么是percona？

Percona 为 MySQL 数据库服务器进行了改进，在功能和性能上较 MySQL 有着很显著的提升。该版本提升了在高负载情况下的 InnoDB 的性能、为 DBA 提供一些非常有用的性能诊断工具；另外有更多的参数和命令来控制服务器行为。

Percona Server 只包含 MySQL 的服务器版，并没有提供相对应 MySQL 的 Connector 和 GUI 工具进行改进。

Percona Server 使用了一些 google-mysql-tools, Proven Scaling, Open Query 对 MySQL 进行改造。

官网：<https://www.percona.com/software/mysql-database>

1.2、安装部署

```
1 #镜像地址：https://hub.docker.com/_/percona/
2 #拉取镜像
3 docker pull percona:5.7.23
4
5 #创建容器
6 docker create --name percona -v /data/mysql-data:/var/lib/mysql -p 3306:3306 -e
  MYSQL_ROOT_PASSWORD=root percona:5.7.23
7
8 #参数解释：
9 --name：percona 指定是容器的名称
10 -v：      /data/mysql-data:/var/lib/mysql  将主机目录/data/mysql-data挂载到容器
  的/var/lib/mysql上
11 -p：      3306:3306 设置端口映射，主机端口是3306，容器内部端口3306
12 -e：      MYSQL_ROOT_PASSWORD=root 设置容器参数，设置root用户的密码为root
13 percona:5.7.23： 镜像名:版本
14
15 #启动容器
16 docker start percona
```

测试：



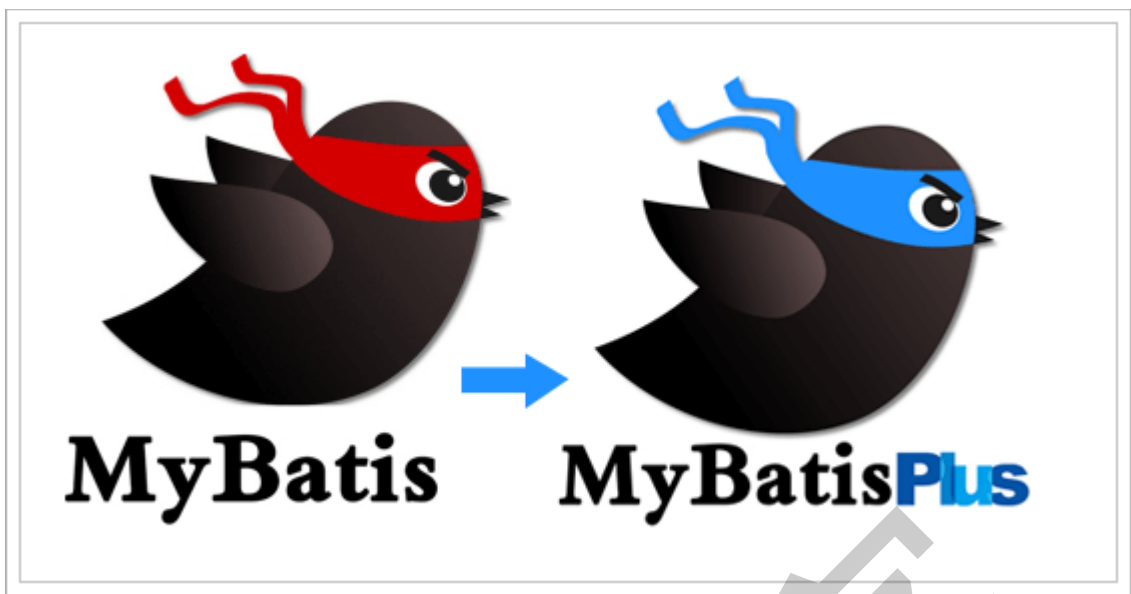
2、MybatisPlus入门

在后台系统服务的开发过程中，必然要和数据库进行交互，在本套课程中，ORM这一层的技术选型，我们采用Mybatis框架作为持久层框架，原因是Mybatis对SQL语句编写更加的灵活。

为了提升开发的效率，所以选用MybatisPlus作为mybatis的插件，以提升开发的效率。下面我们来学习下MybatisPlus插件的使用。

2.1、简介

MyBatis-Plus (简称 MP) 是一个 MyBatis 的增强工具，在 MyBatis 的基础上只做增强不做改变，为简化开发、提高效率而生。



我们的愿景是成为 MyBatis 最好的搭档，就像 魂斗罗 中的 1P、2P，基友搭配，效率翻倍。

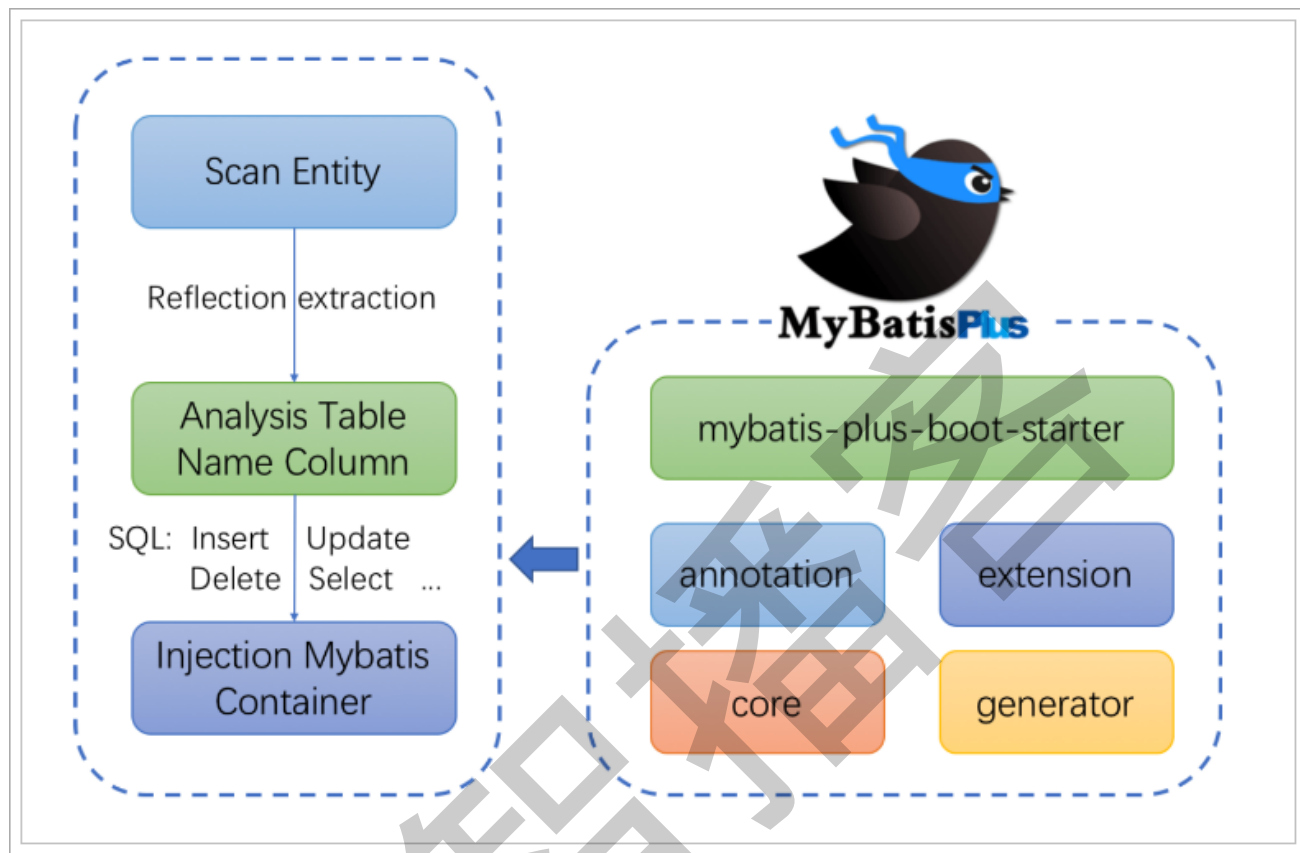


2.2、特性

- **无侵入**：只做增强不做改变，引入它不会对现有工程产生影响，如丝般顺滑
- **损耗小**：启动即会自动注入基本 CURD，性能基本无损耗，直接面向对象操作
- **强大的 CRUD 操作**：内置通用 Mapper、通用 Service，仅仅通过少量配置即可实现单表大部分 CRUD 操作，更有强大的条件构造器，满足各类使用需求
- **支持 Lambda 形式调用**：通过 Lambda 表达式，方便的编写各类查询条件，无需再担心字段写错
- **支持多种数据库**：支持 MySQL、MariaDB、Oracle、DB2、H2、HSQL、SQLite、Postgre、SQLServer2005、SQLServer 等多种数据库
- **支持主键自动生成**：支持多达 4 种主键策略（内含分布式唯一 ID 生成器 - Sequence），可自由配置，完美解决主键问题
- **支持 XML 热加载**：Mapper 对应的 XML 支持热加载，对于简单的 CRUD 操作，甚至可以无 XML 启动
- **支持 ActiveRecord 模式**：支持 ActiveRecord 形式调用，实体类只需继承 Model 类即可进行强大的 CRUD 操作
- **支持自定义全局通用操作**：支持全局通用方法注入（Write once, use anywhere）
- **支持关键词自动转义**：支持数据库关键词（order、key.....）自动转义，还可自定义关键词
- **内置代码生成器**：采用代码或者 Maven 插件可快速生成 Mapper、Model、Service、Controller 层代码，支持模板引擎，更有超多自定义配置等您来使用
- **内置分页插件**：基于 MyBatis 物理分页，开发者无需关心具体操作，配置好插件之后，写分页等同于普通 List 查询

- **内置性能分析插件**：可输出 Sql 语句以及其执行时间，建议开发测试时启用该功能，能快速揪出慢查询
- **内置全局拦截插件**：提供全表 delete、update 操作智能分析阻断，也可自定义拦截规则，预防误操作
- **内置 Sql 注入剥离器**：支持 Sql 注入剥离，有效预防 Sql 注入攻击

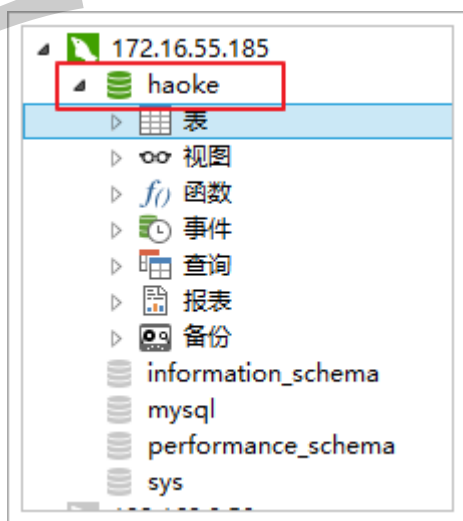
2.3、架构



2.4、快速入门

2.4.1、创建表

首先，创建数据库：haoke:



```
1 CREATE TABLE `user` (
```



```
2  `id` bigint(20) NOT NULL COMMENT '主键ID',
3  `name` varchar(30) DEFAULT NULL COMMENT '姓名',
4  `age` int(11) DEFAULT NULL COMMENT '年龄',
5  `email` varchar(50) DEFAULT NULL COMMENT '邮箱',
6  PRIMARY KEY (`id`)
7 ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
8
9  -- 插入数据
10 INSERT INTO `user` (`id`, `name`, `age`, `email`) VALUES ('1', 'Jone', '18',
11 'test1@baomidou.com');
12 INSERT INTO `user` (`id`, `name`, `age`, `email`) VALUES ('2', 'Jack', '20',
13 'test2@baomidou.com');
14 INSERT INTO `user` (`id`, `name`, `age`, `email`) VALUES ('3', 'Tom', '28',
15 'test3@baomidou.com');
16 INSERT INTO `user` (`id`, `name`, `age`, `email`) VALUES ('4', 'Sandy', '21',
17 'test4@baomidou.com');
18 INSERT INTO `user` (`id`, `name`, `age`, `email`) VALUES ('5', 'Billie', '24',
19 'test5@baomidou.com');
```

2.4.2、创建工程以及导入依赖

New Project

GroupId: cn.itcast.mybatisplus ☒ Inherit

ArtifactId: itcast-mybatis-plus

Version: 1.0-SNAPSHOT ☒ Inherit

Previous Next Cancel Help

导入依赖：



```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
5     <modelVersion>4.0.0</modelVersion>
6
7     <parent>
8         <groupId>org.springframework.boot</groupId>
9         <artifactId>spring-boot-starter-parent</artifactId>
10        <version>2.1.0.RELEASE</version>
11    </parent>
12
13    <groupId>cn.itcast.mybatisplus</groupId>
14    <artifactId>itcast-mybatis-plus</artifactId>
15    <version>1.0-SNAPSHOT</version>
16
17    <dependencies>
18        <dependency>
19            <groupId>org.springframework.boot</groupId>
20            <artifactId>spring-boot-starter</artifactId>
21        </dependency>
22        <dependency>
23            <groupId>org.springframework.boot</groupId>
24            <artifactId>spring-boot-starter-test</artifactId>
25            <scope>test</scope>
26        </dependency>
27
28        <!--简化代码的工具包-->
29        <dependency>
30            <groupId>org.projectlombok</groupId>
31            <artifactId>lombok</artifactId>
32            <optional>true</optional>
33        </dependency>
34        <!--mybatis-plus的springboot支持-->
35        <dependency>
36            <groupId>com.baomidou</groupId>
37            <artifactId>mybatis-plus-boot-starter</artifactId>
38            <version>3.0.5</version>
39        </dependency>
40        <!--mysql驱动-->
41        <dependency>
42            <groupId>mysql</groupId>
43            <artifactId>mysql-connector-java</artifactId>
44            <version>5.1.47</version>
45        </dependency>
46    </dependencies>
47
48    <build>
49        <plugins>
50            <plugin>
51                <groupId>org.springframework.boot</groupId>
52                <artifactId>spring-boot-maven-plugin</artifactId>
```



```
53         </plugin>
54     </plugins>
55 </build>
56
57
58 </project>
```

2.4.3、编写application.properties文件

```
1 spring.application.name = itcast-mybatis-plus
2
3 spring.datasource.driver-class-name=com.mysql.jdbc.Driver
4 spring.datasource.url=jdbc:mysql://172.16.55.185:3306/haoke?
  useUnicode=true&characterEncoding=utf8&autoReconnect=true&allowMultiQueries=true&useS
  SL=false
5 spring.datasource.username=root
6 spring.datasource.password=root
```

2.4.4、创建User对象

```
1 package cn.itcast.mybatisplus.pojo;
2
3 public class User {
4     private Long id;
5     private String name;
6     private Integer age;
7     private String email;
8
9     public Long getId() {
10         return id;
11     }
12
13     public void setId(Long id) {
14         this.id = id;
15     }
16
17     public String getName() {
18         return name;
19     }
20
21     public void setName(String name) {
22         this.name = name;
23     }
24
25     public Integer getAge() {
26         return age;
27     }
28
29     public void setAge(Integer age) {
30         this.age = age;
31     }
32 }
```




```
33     public String getEmail() {
34         return email;
35     }
36
37     public void setEmail(String email) {
38         this.email = email;
39     }
40
41     @Override
42     public String toString() {
43         return "User{" +
44             "id=" + id +
45             ", name='" + name + '\'' +
46             ", age=" + age +
47             ", email='" + email + '\'' +
48             '}';
49     }
50 }
```

2.4.5、编写UserMapper

```
1 package cn.itcast.mybatisplus.mapper;
2
3 import cn.itcast.mybatisplus.pojo.User;
4 import com.baomidou.mybatisplus.core.mapper.BaseMapper;
5
6 public interface UserMapper extends BaseMapper<User> {
7
8 }
```

2.4.6、编写SpringBoot启动类

```
1 package cn.itcast.mybatisplus;
2
3 import org.mybatis.spring.annotation.MapperScan;
4 import org.springframework.boot.SpringApplication;
5 import org.springframework.boot.WebApplicationType;
6 import org.springframework.boot.autoconfigure.SpringBootApplication;
7 import org.springframework.boot.builder.SpringApplicationBuilder;
8
9 @MapperScan("cn.itcast.mybatisplus.mapper") //设置mapper接口的扫描包
10 @SpringBootApplication
11 public class MyApplication {
12
13     public static void main(String[] args) {
14         SpringApplication.run(MyApplication.class, args);
15     }
16 }
17 }
18 }
```




2.4.7、编写单元测试用例

```
1 package cn.itcast.mybatisplus.mapper;
2
3 import cn.itcast.mybatisplus.pojo.User;
4 import org.junit.Test;
5 import org.junit.runner.RunWith;
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.boot.test.context.SpringBootTest;
8 import org.springframework.test.context.junit4.SpringRunner;
9
10 import java.util.List;
11
12 @RunWith(SpringRunner.class)
13 @SpringBootTest
14 public class UserMapperTest {
15
16     @Autowired
17     private UserMapper userMapper;
18
19     @Test
20     public void testSelect() {
21         System.out.println("----- selectAll method test -----");
22         List<User> userList = userMapper.selectAll();
23         for (User user : userList) {
24             System.out.println(user);
25         }
26     }
27
28
29 }
30
```

测试结果：

```
Tests passed: 1 of 1 test - 263 ms
----- selectAll method test -----
User{id=1, name='Jone', age=18, email='test1@baomidou.com'}
User{id=2, name='Jack', age=20, email='test2@baomidou.com'}
User{id=3, name='Tom', age=28, email='test3@baomidou.com'}
User{id=4, name='Sandy', age=21, email='test4@baomidou.com'}
User{id=5, name='Billie', age=24, email='test5@baomidou.com'}
```

可以看到，通过简单的一些代码的编写即可实现数据的查询操作。

2.5、BaseMapper

在MybatisPlus中，BaseMapper中定义了一些常用的CRUD方法，当我们自定义的Mapper接口继承BaseMapper后即可拥有了这些方法。

需要说明的是：这些方法仅适合单表操作。



```
1  /**
2   * <p>
3   * Mapper 继承该接口后, 无需编写 mapper.xml 文件, 即可获得CRUD功能
4   * </p>
5   * <p>
6   * 这个 Mapper 支持 id 泛型
7   * </p>
8   *
9   * @author hubin
10  * @since 2016-01-23
11  */
12  public interface BaseMapper<T> {
13
14      /**
15       * <p>
16       * 插入一条记录
17       * </p>
18       *
19       * @param entity 实体对象
20       */
21      int insert(T entity);
22
23      /**
24       * <p>
25       * 根据 ID 删除
26       * </p>
27       *
28       * @param id 主键ID
29       */
30      int deleteById(Serializable id);
31
32      /**
33       * <p>
34       * 根据 columnMap 条件, 删除记录
35       * </p>
36       *
37       * @param columnMap 表字段 map 对象
38       */
39      int deleteByMap(@Param(Constants.COLUMN_MAP) Map<String, Object> columnMap);
40
41      /**
42       * <p>
43       * 根据 entity 条件, 删除记录
44       * </p>
45       *
46       * @param queryWrapper 实体对象封装操作类 (可以为 null)
47       */
48      int delete(@Param(Constants.WRAPPER) Wrapper<T> queryWrapper);
49
50      /**
51       * <p>
52       * 删除 (根据ID 批量删除)
53       * </p>
```



```
54      *
55      * @param idList 主键ID列表(不能为 null 以及 empty)
56      */
57      int deleteBatchIds(@Param(Constants.COLLECTION) Collection<? extends
Serializable> idList);
58
59      /**
60      * <p>
61      * 根据 ID 修改
62      * </p>
63      *
64      * @param entity 实体对象
65      */
66      int updateById(@Param(Constants.ENTITY) T entity);
67
68      /**
69      * <p>
70      * 根据 whereEntity 条件, 更新记录
71      * </p>
72      *
73      * @param entity 实体对象 (set 条件值,不能为 null)
74      * @param updateWrapper 实体对象封装操作类 (可以为 null,里面的 entity 用于生成 where
语句)
75      */
76      int update(@Param(Constants.ENTITY) T entity, @Param(Constants.WRAPPER)
Wrapper<T> updateWrapper);
77
78      /**
79      * <p>
80      * 根据 ID 查询
81      * </p>
82      *
83      * @param id 主键ID
84      */
85      T selectById(Serializable id);
86
87      /**
88      * <p>
89      * 查询 (根据ID 批量查询)
90      * </p>
91      *
92      * @param idList 主键ID列表(不能为 null 以及 empty)
93      */
94      List<T> selectBatchIds(@Param(Constants.COLLECTION) Collection<? extends
Serializable> idList);
95
96      /**
97      * <p>
98      * 查询 (根据 columnMap 条件)
99      * </p>
100     *
101     * @param columnMap 表字段 map 对象
102     */
```



```
103     List<T> selectByMap(@Param(Constants.COLUMN_MAP) Map<String, Object>  
104     columnMap);  
105  
106     /**  
107     * <p>  
108     * 根据 entity 条件，查询一条记录  
109     * </p>  
110     * @param queryWrapper 实体对象  
111     */  
112     T selectOne(@Param(Constants.WRAPPER) Wrapper<T> queryWrapper);  
113  
114     /**  
115     * <p>  
116     * 根据 wrapper 条件，查询总记录数  
117     * </p>  
118     *  
119     * @param queryWrapper 实体对象  
120     */  
121     Integer selectCount(@Param(Constants.WRAPPER) Wrapper<T> queryWrapper);  
122  
123     /**  
124     * <p>  
125     * 根据 entity 条件，查询全部记录  
126     * </p>  
127     *  
128     * @param queryWrapper 实体对象封装操作类（可以为 null）  
129     */  
130     List<T> selectList(@Param(Constants.WRAPPER) Wrapper<T> queryWrapper);  
131  
132     /**  
133     * <p>  
134     * 根据 wrapper 条件，查询全部记录  
135     * </p>  
136     *  
137     * @param queryWrapper 实体对象封装操作类（可以为 null）  
138     */  
139     List<Map<String, Object>> selectMaps(@Param(Constants.WRAPPER) Wrapper<T>  
140     queryWrapper);  
141  
142     /**  
143     * <p>  
144     * 根据 wrapper 条件，查询全部记录  
145     * 注意：只返回第一个字段的值  
146     * </p>  
147     *  
148     * @param queryWrapper 实体对象封装操作类（可以为 null）  
149     */  
150     List<Object> selectObjs(@Param(Constants.WRAPPER) Wrapper<T> queryWrapper);  
151  
152     /**  
153     * <p>  
154     * 根据 entity 条件，查询全部记录（并翻页）
```



```
154      * </p>
155      *
156      * @param page          分页查询条件 ( 可以为 RowBounds.DEFAULT )
157      * @param queryWrapper 实体对象封装操作类 ( 可以为 null )
158      */
159      IPage<T> selectPage(IPage<T> page, @Param(Constants.WRAPPER) wrapper<T>
queryWrapper);
160
161      /**
162      * <p>
163      * 根据 wrapper 条件，查询全部记录 ( 并翻页 )
164      * </p>
165      *
166      * @param page          分页查询条件
167      * @param queryWrapper 实体对象封装操作类
168      */
169      IPage<Map<String, Object>> selectMapsPage(IPage<T> page,
@param(Constants.WRAPPER) wrapper<T> queryWrapper);
170  }
171
```

2.5.1、like查询

```
1      @Test
2      public void testSelectByLike(){
3          QueryWrapper queryWrapper = new QueryWrapper(new User());
4          //查询名字中包含"o"的用户
5          queryWrapper.like("name", "o");
6
7          List<User> users = this.userMapper.selectList(queryWrapper);
8          for (User user : users) {
9              System.out.println(user);
10         }
11     }
```

✓ Tests passed: 1 of 1 test - 333 ms

```
User{id=1, name='Jone', age=18, email='test1@baomidou.com'}
User{id=3, name='Tom', age=28, email='test3@baomidou.com'}
|
```

2.5.2、条件查询



```
1  @Test
2  public void testSelectByLe(){
3      QueryWrapper queryWrapper = new QueryWrapper(new User());
4      // 查询年龄小于等于20的用户
5      queryWrapper.le("age", 20);
6
7      List<User> users = this.userMapper.selectList(queryWrapper);
8      for (User user : users) {
9          System.out.println(user);
10     }
11 }
```

Tests passed: 1 of 1 test – 345 ms

```
User{id=1, name='Jone', age=18, email='test1@baomidou.com'}
User{id=2, name='Jack', age=20, email='test2@baomidou.com'}
```

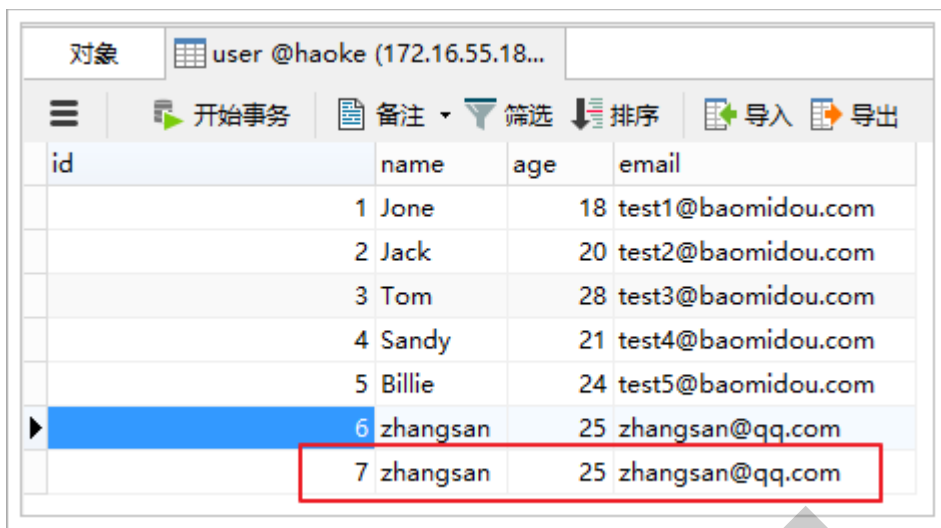
更多查看：<http://mp.baomidou.com/guide/wrapper.html#abstractwrapper>

2.5.3、插入数据

```
1  @Test
2  public void testSave(){
3      User user = new User();
4      user.setAge(25);
5      user.setEmail("zhangsan@qq.com");
6      user.setName("zhangsan");
7      int count = this.userMapper.insert(user);
8      System.out.println("新增数据成功! count => " + count);
9  }
```

设置自增长id：

```
1  public class User {
2
3      @TableId(value = "ID", type = IdType.AUTO)
4      private Long id;
5      private String name;
6      private Integer age;
7      private String email;
```



id	name	age	email
1	Jone	18	test1@baomidou.com
2	Jack	20	test2@baomidou.com
3	Tom	28	test3@baomidou.com
4	Sandy	21	test4@baomidou.com
5	Billie	24	test5@baomidou.com
6	zhangsan	25	zhangsan@qq.com
7	zhangsan	25	zhangsan@qq.com

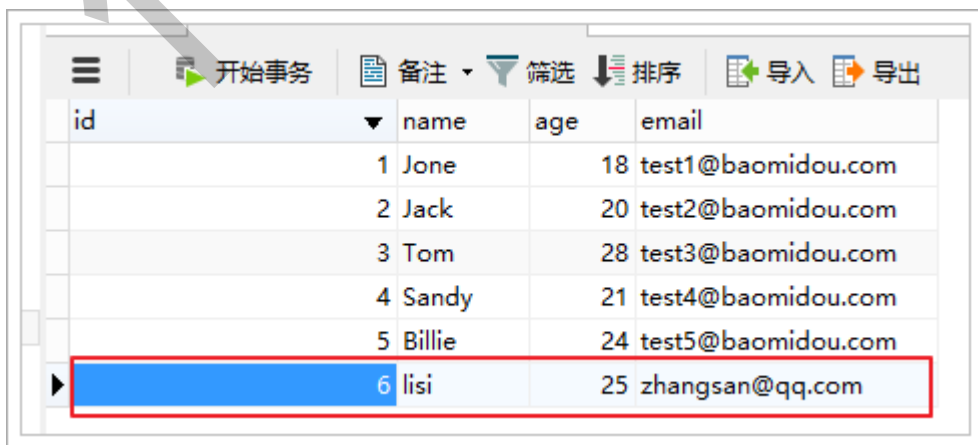
2.5.4、删除数据

```
1 @Test
2 public void testDelete(){
3     this.userMapper.deleteById(7L);
4     System.out.println("删除成功!");
5 }
```

2.5.5、修改数据

根据id修改数据，修改不为null的字段。

```
1 @Test
2 public void testUpdate(){
3     User user = new User();
4     user.setId(6L);
5     user.setName("lisi");
6     this.userMapper.updateById(user);
7     System.out.println("修改成功!");
8 }
```



id	name	age	email
1	Jone	18	test1@baomidou.com
2	Jack	20	test2@baomidou.com
3	Tom	28	test3@baomidou.com
4	Sandy	21	test4@baomidou.com
5	Billie	24	test5@baomidou.com
6	lisi	25	zhangsan@qq.com

2.5.6、分页查询



首先需要配置分页插件：

```
1 package cn.itcast.mybatisplus;
2
3 import com.baomidou.mybatisplus.extension.plugins.PaginationInterceptor;
4 import org.mybatis.spring.annotation.MapperScan;
5 import org.springframework.boot.SpringApplication;
6 import org.springframework.boot.WebApplicationType;
7 import org.springframework.boot.autoconfigure.SpringBootApplication;
8 import org.springframework.boot.builder.SpringApplicationBuilder;
9 import org.springframework.context.annotation.Bean;
10
11 @MapperScan("cn.itcast.mybatisplus.mapper")
12 @SpringBootApplication
13 public class MyApplication {
14
15     /**
16      * 分页插件
17      */
18     @Bean
19     public PaginationInterceptor paginationInterceptor() {
20         return new PaginationInterceptor();
21     }
22
23     public static void main(String[] args) {
24         SpringApplication.run(MyApplication.class, args);
25     }
26 }
27
28
```

下面进行查询：

```
1 @Test
2 public void testSelectPage() {
3     Page<User> page = new Page<>(1, 2);
4     IPage<User> userIPage = this.userMapper.selectPage(page, null);
5     System.out.println("总条数 -----> " + userIPage.getTotal());
6     System.out.println("当前页数 -----> " + userIPage.getCurrent());
7     System.out.println("当前每页显示数 -----> " + userIPage.getSize());
8
9     List<User> records = userIPage.getRecords();
10    for (User user : records) {
11        System.out.println(user);
12    }
13 }
```

✓ Tests passed: 1 of 1 test - 337 ms

```
总条数 -----> 6
当前页数 -----> 2
当前每页显示数 -----> 2
User{id=3, name='Tom', age=28, email='test3@baomidou.com'}
User{id=4, name='Sandy', age=21, email='test4@baomidou.com'}
```

2.6、配置

虽然在MybatisPlus中可以实现零配置，但是有些时候需要我们自定义一些配置，就需要使用Mybatis原生的一些配置文件方式了。

application.properties :

```
1 # 指定全局配置文件
2 mybatis-plus.config-location = classpath:mybatis-config.xml
3 # 指定mapper.xml文件
4 mybatis-plus.mapper-locations = classpath*:mybatis/*.xml
```

更多配置：<http://mp.baomidou.com/guide/config.html#%E5%9F%BA%E6%9C%AC%E9%85%8D%E7%BD%AE>

2.7、Lombok

lombok 提供了简单的注解的形式来帮助我们简化消除一些必须有但显得很臃肿的 java 代码，尤其是针对pojo，在MybatisPlus中使用lombok。

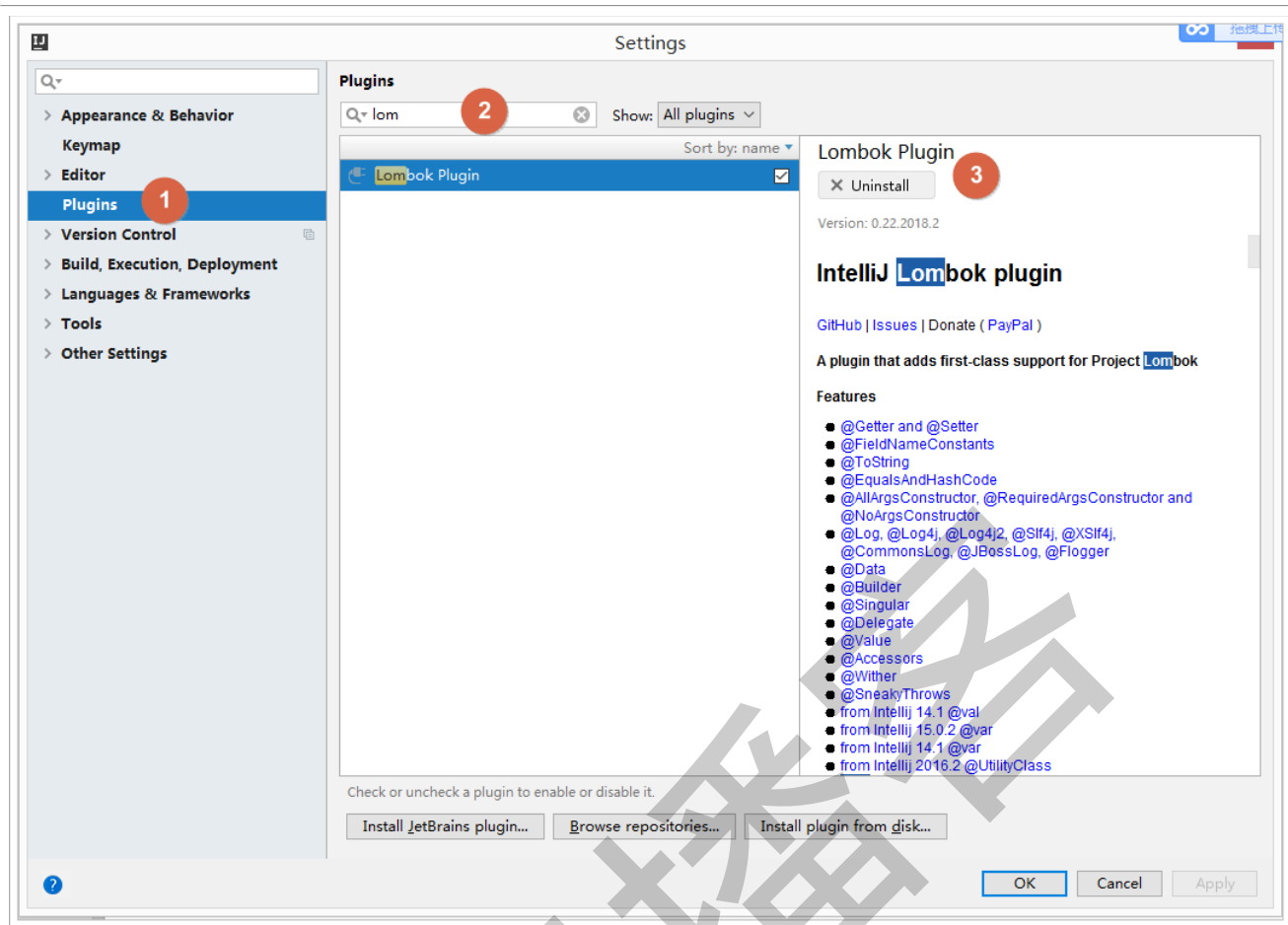
官网：<https://projectlombok.org/>

2.7.1、配置安装

导入依赖：

```
1 <!--简化代码的工具包-->
2 <dependency>
3   <groupId>org.projectlombok</groupId>
4   <artifactId>lombok</artifactId>
5   <optional>true</optional>
6   <version>1.18.4</version>
7 </dependency>
```

安装IDEA插件：

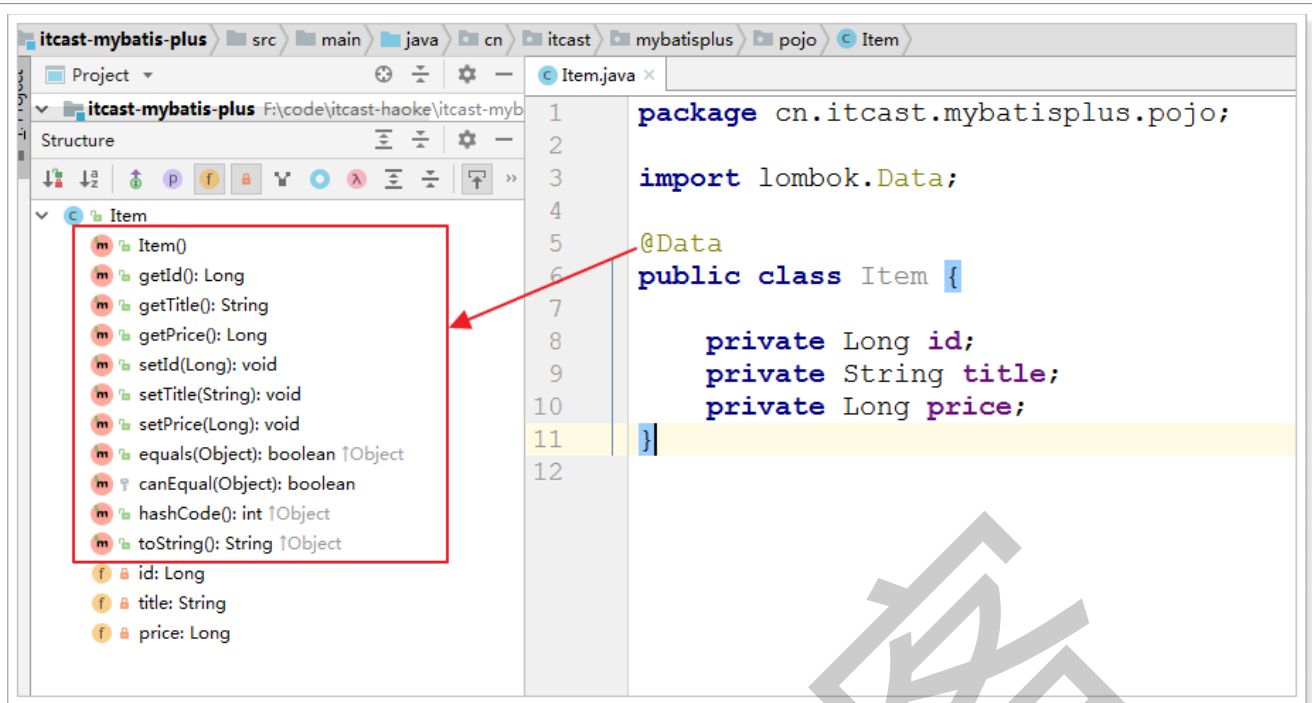


如果不安装插件，程序可以正常执行，但是看不到生成的一些代码，如：get、set方法。

2.7.2、常用注解

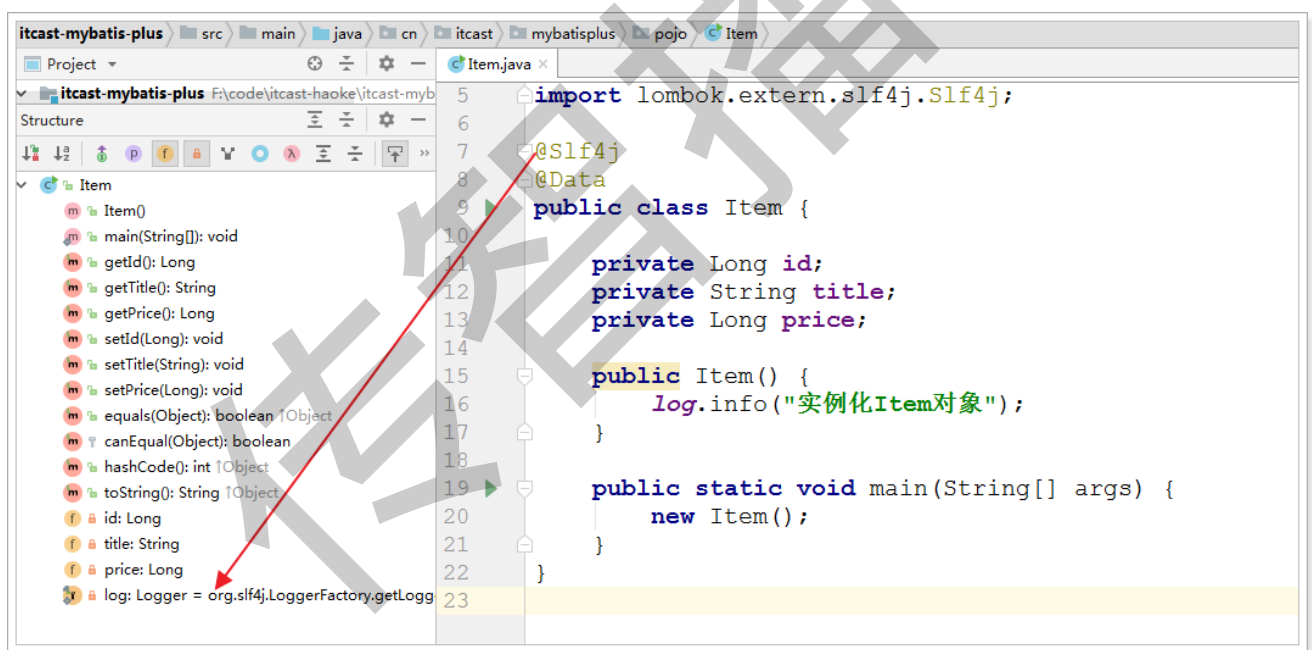
- @Data：注解在类上；提供类所有属性的 getting 和 setting 方法，此外还提供了 equals、hashCode、toString 方法
- @Setter：注解在属性上；为属性提供 setting 方法
- @Getter：注解在属性上；为属性提供 getting 方法
- @Slf4j：注解在类上；为类提供一个 属性名为 log 的 slf4j 日志对象
- @NoArgsConstructor：注解在类上；为类提供一个无参的构造方法
- @AllArgsConstructor：注解在类上；为类提供一个全参的构造方法
- @Builder：使用 Builder 模式构建对象

测试一：使用@Data注解

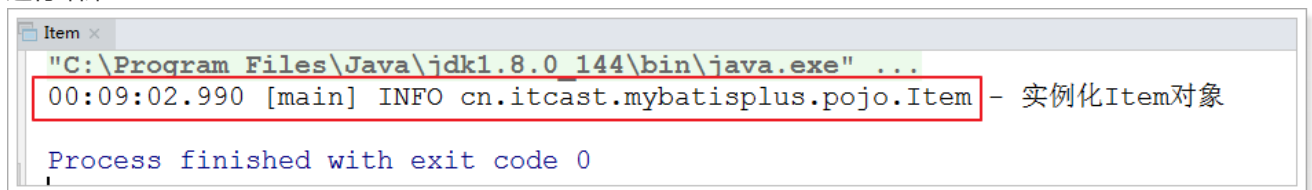


是不是很神奇？！

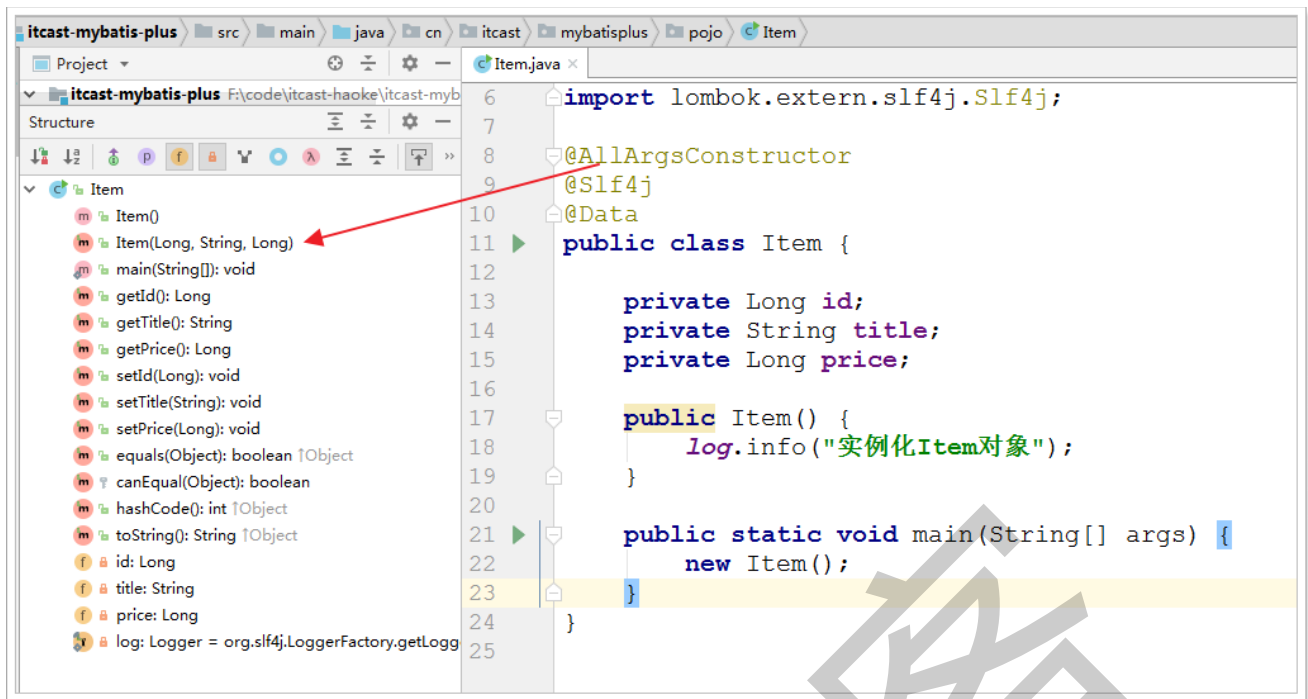
测试二：使用@Slf4j注解



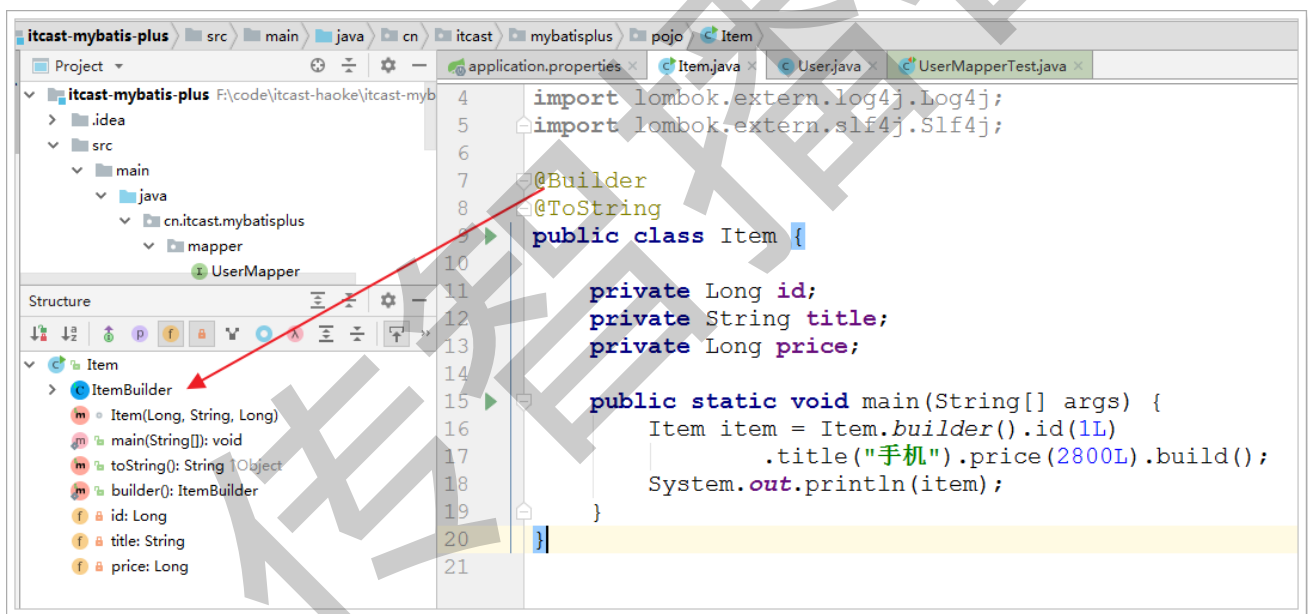
运行结果：



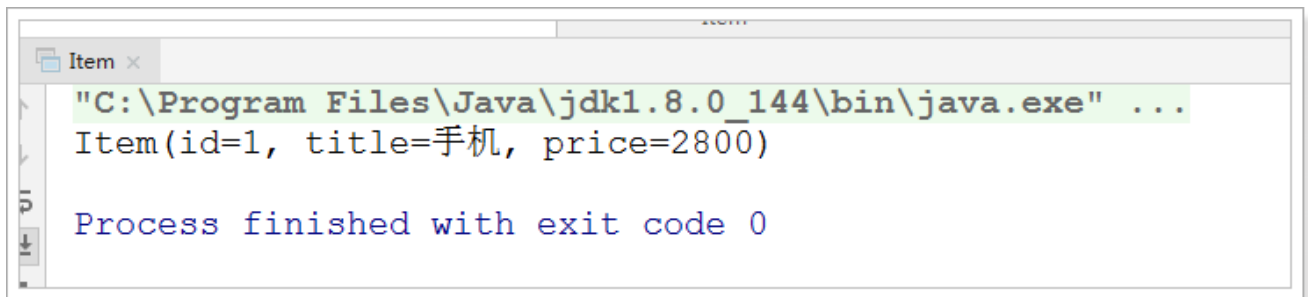
测试三：@AllArgsConstructor注解的使用



测试四：@Builder

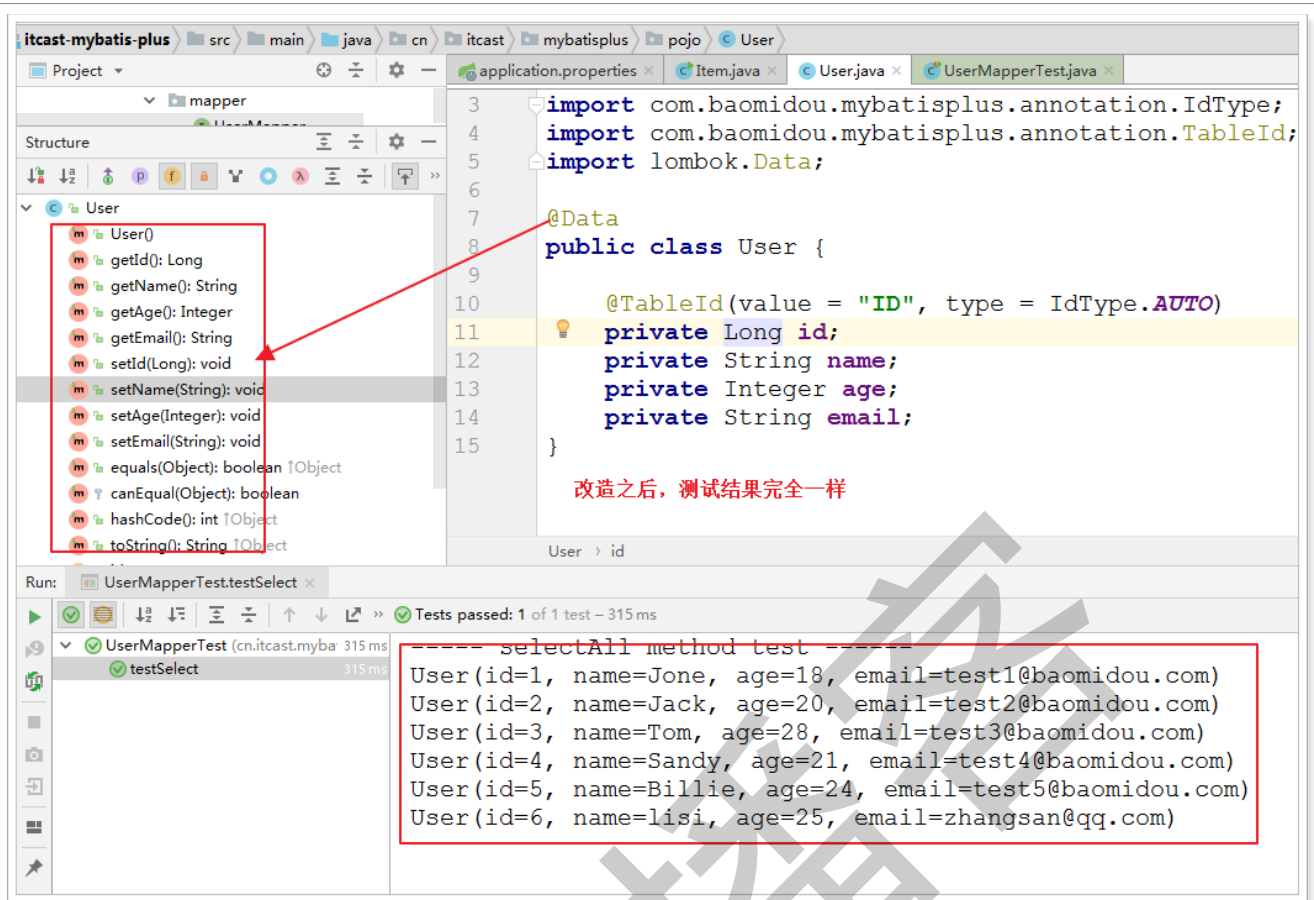


测试结果：



2.7.3、使用lombok改造User对象

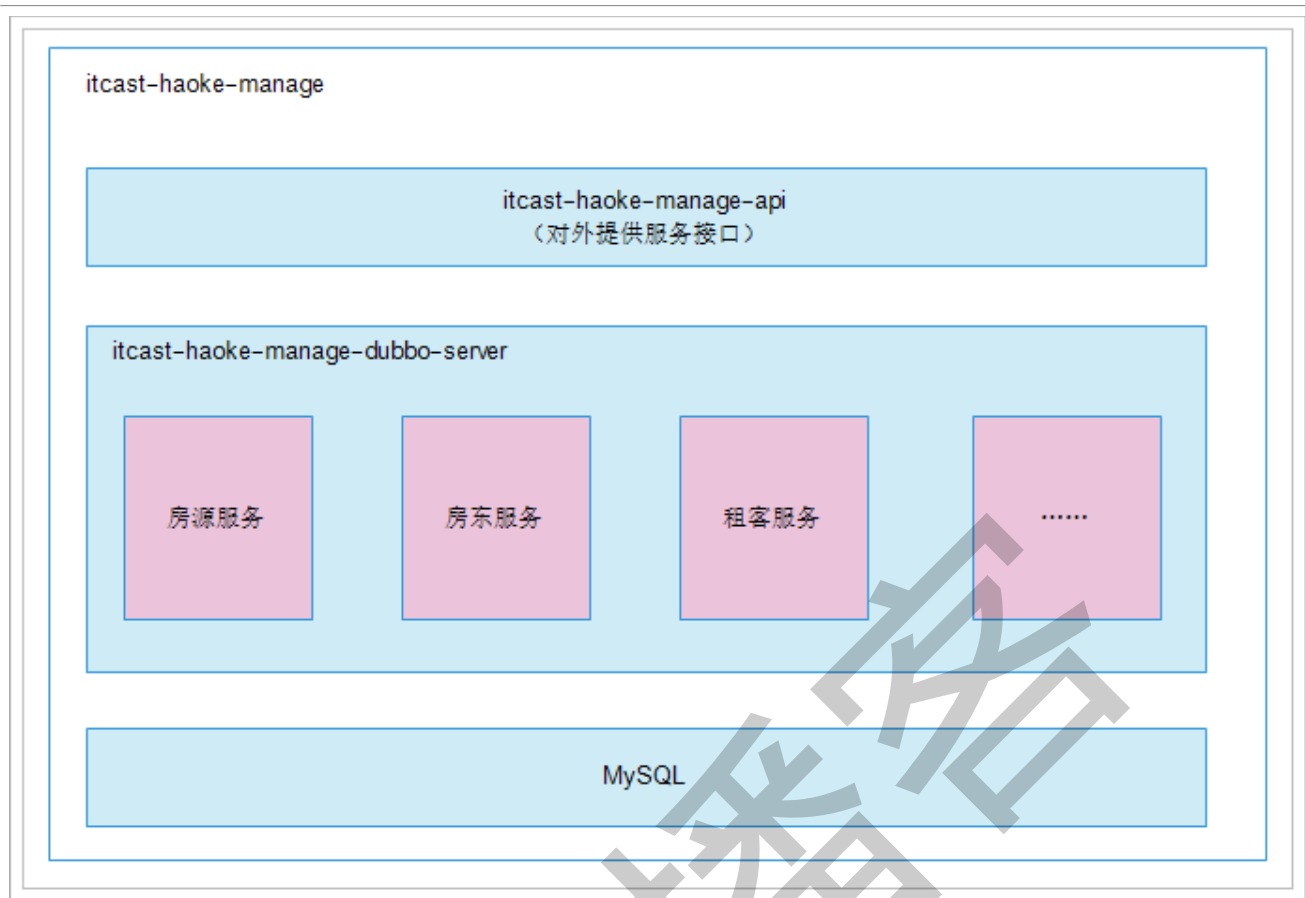
将前面测试的User对象进行改造。



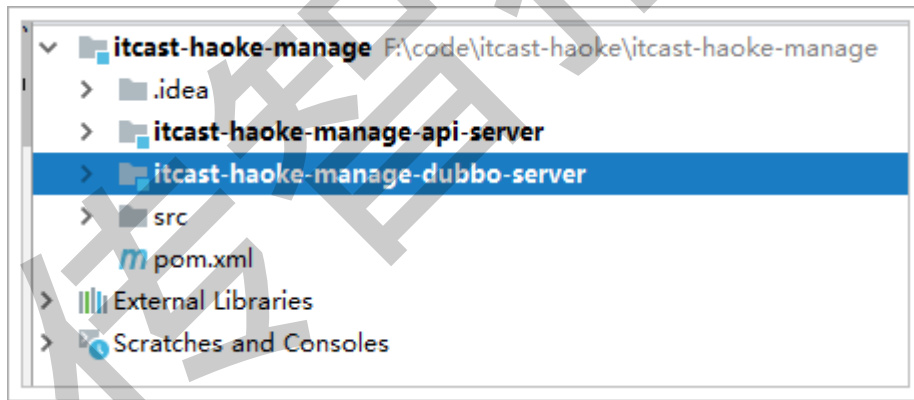
3、搭建后台服务系统

后台服务系统采用SOA的架构思想，使用dubbo作为服务治理框架进行搭建。

后台服务系统架构：



工程结构：



3.1、创建itcast-haoke-manage工程

pom.xml文件：

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5         http://maven.apache.org/xsd/maven-4.0.0.xsd">
6     <modelVersion>4.0.0</modelVersion>
7
8     <!--spring boot的支持-->
9     <parent>
10         <groupId>org.springframework.boot</groupId>
```




```
10     <artifactId>spring-boot-starter-parent</artifactId>
11     <version>2.1.0.RELEASE</version>
12 </parent>
13
14
15 <groupId>cn.itcast.haoke</groupId>
16 <artifactId>itcast-haoke-manage</artifactId>
17 <packaging>pom</packaging>
18 <version>1.0-SNAPSHOT</version>
19
20 <!--子模块-->
21 <modules>
22     <module>itcast-haoke-manage-api-server</module>
23     <module>itcast-haoke-manage-dubbo-server</module>
24 </modules>
25
26 <dependencies>
27     <!--springboot 测试支持-->
28     <dependency>
29         <groupId>org.springframework.boot</groupId>
30         <artifactId>spring-boot-starter-test</artifactId>
31         <scope>test</scope>
32     </dependency>
33
34     <!--dubbo的springboot支持-->
35     <dependency>
36         <groupId>com.alibaba.boot</groupId>
37         <artifactId>dubbo-spring-boot-starter</artifactId>
38         <version>0.2.0</version>
39     </dependency>
40     <!--dubbo框架-->
41     <dependency>
42         <groupId>com.alibaba</groupId>
43         <artifactId>dubbo</artifactId>
44         <version>2.6.4</version>
45     </dependency>
46     <dependency>
47         <groupId>org.apache.commons</groupId>
48         <artifactId>commons-lang3</artifactId>
49         <version>3.7</version>
50     </dependency>
51     <!--zk依赖-->
52     <dependency>
53         <groupId>org.apache.zookeeper</groupId>
54         <artifactId>zookeeper</artifactId>
55         <version>3.4.13</version>
56     </dependency>
57     <dependency>
58         <groupId>com.github.sgroschupf</groupId>
59         <artifactId>zookeeper-client</artifactId>
60         <version>0.1</version>
61     </dependency>
62 </dependencies>
```



```
63
64     <build>
65         <plugins>
66             <!--springboot的maven框架-->
67             <plugin>
68                 <groupId>org.springframework.boot</groupId>
69                 <artifactId>spring-boot-maven-plugin</artifactId>
70             </plugin>
71         </plugins>
72     </build>
73
74
75 </project>
```

3.2、创建itcast-haoke-manage-api-server工程

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
5     <parent>
6         <artifactId>itcast-haoke-manage</artifactId>
7         <groupId>cn.itcast.haoke</groupId>
8         <version>1.0-SNAPSHOT</version>
9     </parent>
10    <modelVersion>4.0.0</modelVersion>
11
12    <artifactId>itcast-haoke-manage-api-server</artifactId>
13
14    <dependencies>
15        <!--springboot的web支持-->
16        <dependency>
17            <groupId>org.springframework.boot</groupId>
18            <artifactId>spring-boot-starter-web</artifactId>
19        </dependency>
20    </dependencies>
21
22 </project>
```

3.3、创建itcast-haoke-manage-dubbo-server工程

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
5     <parent>
6         <artifactId>itcast-haoke-manage</artifactId>
7         <groupId>cn.itcast.haoke</groupId>
8         <version>1.0-SNAPSHOT</version>
```



```
9     </parent>
10     <modelVersion>4.0.0</modelVersion>
11
12     <artifactId>itcast-haoke-manage-dubbo-server</artifactId>
13     <packaging>pom</packaging>
14
15     <dependencies>
16         <dependency>
17             <groupId>org.springframework.boot</groupId>
18             <artifactId>spring-boot-starter</artifactId>
19         </dependency>
20     </dependencies>
21
22 </project>
```

4、新增房源服务

接下来，我们实现新增房源功能。

4.1、数据结构

tb_estate (楼盘表) :

```
1 CREATE TABLE `tb_estate` (
2   `id` bigint(20) NOT NULL AUTO_INCREMENT,
3   `name` varchar(100) DEFAULT NULL COMMENT '楼盘名称',
4   `province` varchar(10) DEFAULT NULL COMMENT '所在省',
5   `city` varchar(10) DEFAULT NULL COMMENT '所在市',
6   `area` varchar(10) DEFAULT NULL COMMENT '所在区',
7   `address` varchar(100) DEFAULT NULL COMMENT '具体地址',
8   `year` varchar(10) DEFAULT NULL COMMENT '建筑年代',
9   `type` varchar(10) DEFAULT NULL COMMENT '建筑类型',
10  `property_cost` varchar(10) DEFAULT NULL COMMENT '物业费',
11  `property_company` varchar(20) DEFAULT NULL COMMENT '物业公司',
12  `developers` varchar(20) DEFAULT NULL COMMENT '开发商',
13  `created` datetime DEFAULT NULL COMMENT '创建时间',
14  `updated` datetime DEFAULT NULL COMMENT '更新时间',
15  PRIMARY KEY (`id`)
16 ) ENGINE=InnoDB AUTO_INCREMENT=1006 DEFAULT CHARSET=utf8 COMMENT='楼盘表';
17
18 --初始数据
19 INSERT INTO `tb_estate` (`id`, `name`, `province`, `city`, `area`, `address`,
20   `year`, `type`, `property_cost`, `property_company`, `developers`, `created`,
21   `updated`) VALUES ('1001', '中远两湾城', '上海市', '上海市', '普陀区', '远景路97弄',
22   '2001', '塔楼/板楼', '1.5', '上海中远物业管理发展有限公司', '上海万业企业股份有限公司', '2018-
23   11-06 23:00:20', '2018-11-06 23:00:23');
20 INSERT INTO `tb_estate` (`id`, `name`, `province`, `city`, `area`, `address`,
21   `year`, `type`, `property_cost`, `property_company`, `developers`, `created`,
22   `updated`) VALUES ('1002', '上海康城', '上海市', '上海市', '闵行区', '莘松路958弄',
23   '2001', '塔楼/板楼', '1.5', '盛孚物业', '闵行房地产', '2018-11-06 23:02:30', '2018-11-27
24   23:02:33');
```

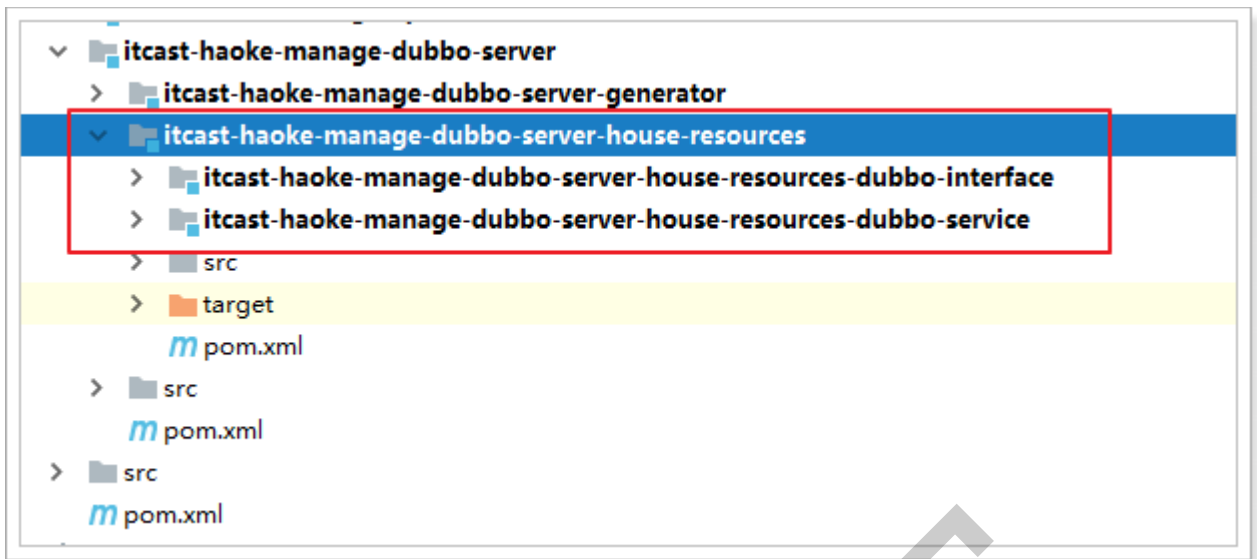


```
21 INSERT INTO `tb_estate` (`id`, `name`, `province`, `city`, `area`, `address`,  
    `year`, `type`, `property_cost`, `property_company`, `developers`, `created`,  
    `updated`) VALUES ('1003', '保利西子湾', '上海市', '上海市', '松江区', '广富林路1188弄',  
    '2008', '塔楼/板楼', '1.75', '上海保利物业管理', '上海城乾房地产开发有限公司', '2018-11-06  
    23:04:22', '2018-11-06 23:04:25');  
22 INSERT INTO `tb_estate` (`id`, `name`, `province`, `city`, `area`, `address`,  
    `year`, `type`, `property_cost`, `property_company`, `developers`, `created`,  
    `updated`) VALUES ('1004', '万科城市花园', '上海市', '上海市', '松江区', '广富林路1188弄',  
    '2002', '塔楼/板楼', '1.5', '上海保利物业管理', '上海城乾房地产开发有限公司', '2018-11-13  
    16:43:40', '2018-11-13 16:43:42');  
23 INSERT INTO `tb_estate` (`id`, `name`, `province`, `city`, `area`, `address`,  
    `year`, `type`, `property_cost`, `property_company`, `developers`, `created`,  
    `updated`) VALUES ('1005', '上海阳城', '上海市', '上海市', '闵行区', '罗锦路888弄',  
    '2002', '塔楼/板楼', '1.5', '上海莲阳物业管理有限公司', '上海莲城房地产开发有限公司', '2018-  
    11-06 23:23:52', '2018-11-06 23:23:55');  
24
```

tb_house_resources (房源表) :

```
1 CREATE TABLE `tb_house_resources` (  
2     `id` bigint(20) NOT NULL AUTO_INCREMENT,  
3     `title` varchar(100) DEFAULT NULL COMMENT '房源标题',  
4     `estate_id` bigint(20) DEFAULT NULL COMMENT '楼盘id',  
5     `building_num` varchar(5) DEFAULT NULL COMMENT '楼号(栋)',  
6     `building_unit` varchar(5) DEFAULT NULL COMMENT '单元号',  
7     `building_floor_num` varchar(5) DEFAULT NULL COMMENT '门牌号',  
8     `rent` int(10) DEFAULT NULL COMMENT '租金',  
9     `rent_method` tinyint(1) DEFAULT NULL COMMENT '租赁方式, 1-整租, 2-合租',  
10    `payment_method` tinyint(1) DEFAULT NULL COMMENT '支付方式, 1-付一押一, 2-付三押一, 3-  
    付六押一, 4-年付押一, 5-其它',  
11    `house_type` varchar(255) DEFAULT NULL COMMENT '户型, 如: 2室1厅1卫',  
12    `covered_area` varchar(10) DEFAULT NULL COMMENT '建筑面积',  
13    `use_area` varchar(10) DEFAULT NULL COMMENT '使用面积',  
14    `floor` varchar(10) DEFAULT NULL COMMENT '楼层, 如: 8/26',  
15    `orientation` varchar(2) DEFAULT NULL COMMENT '朝向: 东、南、西、北',  
16    `decoration` tinyint(1) DEFAULT NULL COMMENT '装修, 1-精装, 2-简装, 3-毛坯',  
17    `facilities` varchar(50) DEFAULT NULL COMMENT '配套设施, 如: 1,2,3',  
18    `pic` varchar(200) DEFAULT NULL COMMENT '图片, 最多5张',  
19    `house_desc` varchar(200) DEFAULT NULL COMMENT '描述',  
20    `contact` varchar(10) DEFAULT NULL COMMENT '联系人',  
21    `mobile` varchar(11) DEFAULT NULL COMMENT '手机号',  
22    `time` tinyint(1) DEFAULT NULL COMMENT '看房时间, 1-上午, 2-中午, 3-下午, 4-晚上, 5-全  
    天',  
23    `property_cost` varchar(10) DEFAULT NULL COMMENT '物业费',  
24    `created` datetime DEFAULT NULL,  
25    `updated` datetime DEFAULT NULL,  
26    PRIMARY KEY (`id`)  
27 ) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8 COMMENT='房源表';  
28
```

4.2、创建itcast-haoke-manage-dubbo-server-house-resources工程



- itcast-haoke-manage-dubbo-server-house-resources-dubbo-interface
 - 对外提供的sdk包
 - 只提供pojo实体以及接口，不提供实现类
- itcast-haoke-manage-dubbo-server-house-resources-dubbo-service
 - 具体实现

pom.xml文件：

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5         http://maven.apache.org/xsd/maven-4.0.0.xsd">
6     <parent>
7         <artifactId>itcast-haoke-manage-dubbo-server</artifactId>
8         <groupId>cn.itcast.haoke</groupId>
9         <version>1.0-SNAPSHOT</version>
10    </parent>
11    <modelVersion>4.0.0</modelVersion>
12    <artifactId>itcast-haoke-manage-dubbo-server-house-resources</artifactId>
13    <packaging>pom</packaging>
14    <modules>
15        <module>itcast-haoke-manage-dubbo-server-house-resources-dubbo-
16        interface</module>
17        <module>itcast-haoke-manage-dubbo-server-house-resources-dubbo-
18        service</module>
19    </modules>
20    <dependencies>
21        <dependency>
22            <groupId>org.projectlombok</groupId>
23            <artifactId>lombok</artifactId>
24            <!--需要注意：传递依赖中，如果需要使用，请显示引入-->
25            <optional>true</optional>
26        </dependency>
27    </dependencies>
28 </project>
```



```
26     <dependency>
27         <groupId>com.baomidou</groupId>
28         <artifactId>mybatis-plus-boot-starter</artifactId>
29         <version>3.0.5</version>
30         <optional>true</optional>
31     </dependency>
32     <dependency>
33         <groupId>mysql</groupId>
34         <artifactId>mysql-connector-java</artifactId>
35         <version>5.1.47</version>
36         <optional>true</optional>
37     </dependency>
38 </dependencies>
39
40
41 </project>
```

4.3、创建子工程

4.3.1、itcast-haoke-manage-dubbo-server-house-resources-dubbo-interface

pom.xml :

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
5     <parent>
6         <artifactId>itcast-haoke-manage-dubbo-server-house-resources</artifactId>
7         <groupId>cn.itcast.haoke</groupId>
8         <version>1.0-SNAPSHOT</version>
9     </parent>
10    <modelVersion>4.0.0</modelVersion>
11
12    <artifactId>itcast-haoke-manage-dubbo-server-house-resources-dubbo-
interface</artifactId>
13
14
15 </project>
```

4.3.2、itcast-haoke-manage-dubbo-server-house-resources-dubbo-service

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
5     <parent>
6         <artifactId>itcast-haoke-manage-dubbo-server-house-resources</artifactId>
7         <groupId>cn.itcast.haoke</groupId>
8         <version>1.0-SNAPSHOT</version>
```



```
9      </parent>
10      <modelVersion>4.0.0</modelVersion>
11
12      <artifactId>itcast-haoke-manage-dubbo-server-house-resources-dubbo-
service</artifactId>
13
14      <dependencies>
15          <dependency>
16              <groupId>org.springframework.boot</groupId>
17              <artifactId>spring-boot-starter-jdbc</artifactId>
18          </dependency>
19
20          <dependency>
21              <groupId>cn.itcast.haoke</groupId>
22              <artifactId>itcast-haoke-manage-dubbo-server-house-resources-dubbo-
interface</artifactId>
23              <version>1.0-SNAPSHOT</version>
24          </dependency>
25      </dependencies>
26
27  </project>
```

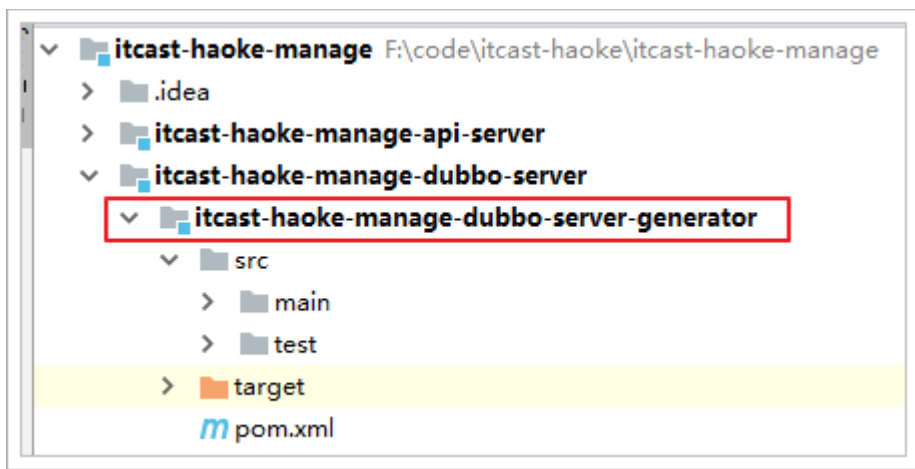
4.4、编写BasePojo文件

编写BasePojo，所有的pojo类都要继承该类，在该类中定义了created、updated字段，表明在每一个表中都需要有这2个字段。

```
1  package cn.itcast.haoke.dubbo.server.pojo;
2
3  import lombok.Data;
4
5  import java.util.Date;
6
7  @Data
8  public abstract class BasePojo implements java.io.Serializable{
9
10     private Date created;
11     private Date updated;
12
13 }
```

4.5、使用MybatisPlus的AutoGenerator插件生成代码文件

4.5.1、创建itcast-haoke-manage-dubbo-server-generator工程



pom.xml文件：

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5         http://maven.apache.org/xsd/maven-4.0.0.xsd">
6     <parent>
7         <artifactId>itcast-haoke-manage-dubbo-server</artifactId>
8         <groupId>cn.itcast.haoke</groupId>
9         <version>1.0-SNAPSHOT</version>
10    </parent>
11    <modelVersion>4.0.0</modelVersion>
12    <artifactId>itcast-haoke-manage-dubbo-server-generator</artifactId>
13
14    <dependencies>
15        <!-- freemarker 模板引擎 -->
16        <dependency>
17            <groupId>org.freemarker</groupId>
18            <artifactId>freemarker</artifactId>
19            <version>2.3.28</version>
20        </dependency>
21    </dependencies>
22
23 </project>
```

4.5.2、编写CodeGenerator

代码参考官方提供代码实现。

```
1 package cn.itcast.haoke.generator;
2
3 import com.alibaba.dubbo.common.utils.StringUtils;
4 import com.baomidou.mybatisplus.core.exceptions.MybatisPlusException;
5 import com.baomidou.mybatisplus.core.toolkit.StringPool;
6 import com.baomidou.mybatisplus.generator.AutoGenerator;
7 import com.baomidou.mybatisplus.generator.InjectionConfig;
8 import com.baomidou.mybatisplus.generator.config.*;
```



```
9  import com.baomidou.mybatisplus.generator.config.po.TableInfo;
10 import com.baomidou.mybatisplus.generator.config.rules.NamingStrategy;
11 import com.baomidou.mybatisplus.generator.engine.FreemarkerTemplateEngine;
12
13 import java.util.ArrayList;
14 import java.util.List;
15 import java.util.Scanner;
16
17 public class CodeGenerator {
18
19     /**
20      * <p>
21      * 读取控制台内容
22      * </p>
23      */
24     public static String scanner(String tip) {
25         Scanner scanner = new Scanner(System.in);
26         StringBuilder help = new StringBuilder();
27         help.append("请输入" + tip + ":");
28         System.out.println(help.toString());
29         if (scanner.hasNext()) {
30             String ipt = scanner.next();
31             if (StringUtils.isNotEmpty(ipt)) {
32                 return ipt;
33             }
34         }
35         throw new MybatisPlusException("请输入正确的" + tip + "！");
36     }
37
38     public static void main(String[] args) {
39         // 代码生成器
40         AutoGenerator mpg = new AutoGenerator();
41
42         // 全局配置
43         GlobalConfig gc = new GlobalConfig();
44         String projectPath = System.getProperty("user.dir");
45         gc.setOutputDir(projectPath + "/src/main/java");
46         gc.setAuthor("itcast");
47         gc.setOpen(false);
48         mpg.setGlobalConfig(gc);
49
50         // 数据源配置
51         DataSourceConfig dsc = new DataSourceConfig();
52         dsc.setUrl("jdbc:mysql://172.16.55.185:3306/haoke?
useUnicode=true&characterEncoding=utf8&autoReconnect=true&allowMultiQueries=true")
53         ;
54         // dsc.setSchemaName("public");
55         dsc.setDriverName("com.mysql.jdbc.Driver");
56         dsc.setUsername("root");
57         dsc.setPassword("root");
58         mpg.setDataSource(dsc);
59
60         // 包配置
```

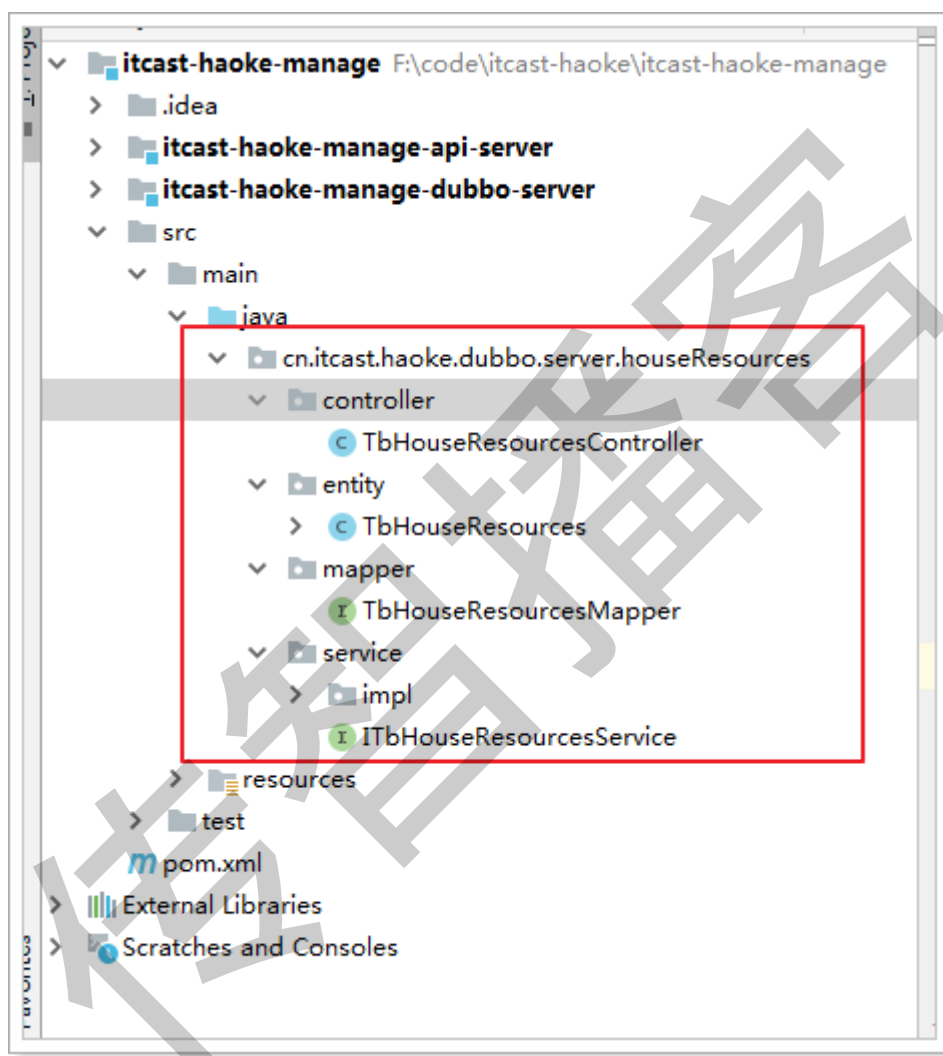


```
60     PackageConfig pc = new PackageConfig();
61     pc.setModuleName(scanner("模块名"));
62     pc.setParent("cn.itcast.haoke.dubbo.server");
63     mpg.setPackageInfo(pc);
64
65     // 自定义配置
66     InjectionConfig cfg = new InjectionConfig() {
67         @Override
68         public void initMap() {
69             // to do nothing
70         }
71     };
72     List<FileOutConfig> focList = new ArrayList<>();
73     focList.add(new FileOutConfig("/templates/mapper.xml.ftl") {
74         @Override
75         public String outputFile(TableInfo tableInfo) {
76             // 自定义输入文件名称
77             return projectPath + "/src/main/resources/mapper/" +
pc.getModuleName()
78                 + "/" + tableInfo.getEntityName() + "Mapper" +
StringPool.DOT_XML;
79         }
80     });
81     cfg.setFileOutConfigList(focList);
82     mpg.setCfg(cfg);
83     mpg.setTemplate(new TemplateConfig().setXml(null));
84
85     // 策略配置
86     StrategyConfig strategy = new StrategyConfig();
87     strategy.setNaming(NamingStrategy.underline_to_camel);
88     strategy.setColumnNaming(NamingStrategy.underline_to_camel);
89
90     strategy.setSuperEntityClass("cn.itcast.haoke.dubbo.server.pojo.BasePojo");
91     strategy.setEntityLombokModel(true);
92     strategy.setRestControllerStyle(true);
93
94     //
95     strategy.setSuperControllerClass("com.baomidou.ant.common.BaseController");
96     strategy.setInclude(scanner("表名"));
97     strategy.setSuperEntityColumns("id");
98     strategy.setControllerMappingHyphenStyle(true);
99     strategy.setTablePrefix(pc.getModuleName() + "_");
100     mpg.setStrategy(strategy);
101     mpg.setTemplateEngine(new FreemarkerTemplateEngine());
102     mpg.execute();
103 }
```

4.5.3、运行代码

```
"C:\Program Files\Java\jdk1.8.0_144\bin\java.exe" ...  
请输入模块名:  
houseResources  
20:29:26.977 [main] INFO com.alibaba.dubbo.common.logger.Log  
请输入表名:  
tb_house_resources
```

生成的结果如下：



可以看到，自动生成了controller、entity、mapper、service等内容。

其实，一般只需要entity就可以了，也就是pojo类。

生成的类是这样的：



```
import ...

/**
 * <p>
 * 房源表
 * </p>
 *
 * @author itcast
 */
@Data
@EqualsAndHashCode(callSuper = true)
@Accessors(chain = true)
public class TbHouseResources extends BasePojo {

    private static final long serialVersionUID = 1L;

    /**
     * 房源标题
     */
}
```

- @EqualsAndHashCode(callSuper = true)
 - 这个是自动生成equals和hashCode方法，我们一般不需要，所以将该注解去掉
- @Accessors(chain = true)
 - 这个表示，生成的set方法将采用链式编程方式，建议保留。

```
public class AccessorsExample {
    private int age = 10;

    public int age() {
        return this.age;
    }

    public AccessorsExample age(final int age) {
        this.age = age;
        return this;
    }
}

class PrefixExample {
    private String fName = "Hello, World!";

    public String getName() {
        return this.fName;
    }
}
```



4.5.4、将整理到pojo文件并且拷贝到工程

将整理的pojo文件拷贝到itcast-haoke-manage-dubbo-server-house-resources-dubbo-interface工程中。

```
1 package cn.itcast.haoke.dubbo.server.pojo;
2
3 import com.baomidou.mybatisplus.annotation.IdType;
4 import com.baomidou.mybatisplus.annotation.TableId;
5 import com.baomidou.mybatisplus.annotation.TableName;
6 import lombok.Data;
7 import lombok.experimental.Accessors;
8
9 /**
10  * <p>
11  * 房源表
12  * </p>
13  *
14  * @author itcast
15  */
16 @Data
17 @Accessors(chain = true)
18 @TableName("tb_house_resources")
19 public class HouseResources extends BasePojo {
20
21     private static final long serialVersionUID = 1690556353781308824L;
22
23     @TableId(value = "id", type = IdType.AUTO)
24     private Long id;
25
26     /**
27      * 房源标题
28      */
29     private String title;
30
31     /**
32      * 楼盘id
33      */
34     private Long estateId;
35
36     /**
37      * 楼号(栋)
38      */
39     private String buildingNum;
40
41     /**
42      * 单元号
43      */
44     private String buildingUnit;
45
46     /**
47      * 门牌号
48      */
49     private String buildingFloorNum;
```



```
50
51     /**
52      * 租金
53      */
54     private Integer rent;
55
56     /**
57      * 租赁方式，1-整租，2-合租
58      */
59     private Integer rentMethod;
60
61     /**
62      * 支付方式，1-付一押一，2-付三押一，3-付六押一，4-年付押一，5-其它
63      */
64     private Integer paymentMethod;
65
66     /**
67      * 户型，如：2室1厅1卫
68      */
69     private String houseType;
70
71     /**
72      * 建筑面积
73      */
74     private String coveredArea;
75
76     /**
77      * 使用面积
78      */
79     private String useArea;
80
81     /**
82      * 楼层，如：8/26
83      */
84     private String floor;
85
86     /**
87      * 朝向：东、南、西、北
88      */
89     private String orientation;
90
91     /**
92      * 装修，1-精装，2-简装，3-毛坯
93      */
94     private Integer decoration;
95
96     /**
97      * 配套设施，如：1,2,3
98      */
99     private String facilities;
100
101     /**
102      * 图片，最多5张
```




```
103     */
104     private String pic;
105
106     /**
107     * 描述
108     */
109     private String houseDesc;
110
111     /**
112     * 联系人
113     */
114     private String contact;
115
116     /**
117     * 手机号
118     */
119     private String mobile;
120
121     /**
122     * 看房时间, 1-上午, 2-中午, 3-下午, 4-晚上, 5-全天
123     */
124     private Integer time;
125
126     /**
127     * 物业费
128     */
129     private String propertyCost;
130
131
132 }
133
```

4.6、定义新增房源的dubbo服务

在itcast-haoke-manage-dubbo-server-house-resources-dubbo-interface工程。

```
1 package cn.itcast.haoke.dubbo.server.api;
2
3 import cn.itcast.haoke.dubbo.server.pojo.HouseResources;
4
5 public interface ApiHouseResourcesService {
6
7     /**
8     *
9     * @param houseResources
10    *
11    * @return -1:输入的参数不符合要求, 0:数据插入数据库失败, 1:成功
12    */
13    int saveHouseResources(HouseResources houseResources);
14
15 }
16
```



4.7、实现新增房源服务

4.7.1、编写application.properties配置文件

```
1 # Spring boot application
2 spring.application.name = itcast-haoke-manage-dubbo-server-house-resources
3
4 # 数据库
5 spring.datasource.driver-class-name=com.mysql.jdbc.Driver
6 spring.datasource.url=jdbc:mysql://172.16.55.185:3306/haoke?
  useUnicode=true&characterEncoding=utf8&autoReconnect=true&allowMultiQueries=true&us
  eSSL=false
7 spring.datasource.username=root
8 spring.datasource.password=root
9
10 # 服务的扫描包
11 dubbo.scan.basePackages = cn.itcast.haoke.dubbo.server.api
12
13 # 应用名称
14 dubbo.application.name = dubbo-provider-house-resources
15
16 # 协议以及端口
17 dubbo.protocol.name = dubbo
18 dubbo.protocol.port = 20881
19
20 # zk注册中心
21 dubbo.registry.address = zookeeper://172.16.55.185:2181
22 dubbo.registry.client = zkclient
23
```

4.7.2、编写HouseResourcesMapper接口

```
1 package cn.itcast.haoke.dubbo.server.mapper;
2
3 import cn.itcast.haoke.dubbo.server.pojo.HouseResources;
4 import com.baomidou.mybatisplus.core.mapper.BaseMapper;
5
6 public interface HouseResourcesMapper extends BaseMapper<HouseResources> {
7 }
8
```

编写MybatisPlus配置类：



```
1 package cn.itcast.haoke.dubbo.server.config;
2
3 import org.mybatis.spring.annotation.MapperScan;
4 import org.springframework.context.annotation.Configuration;
5
6 @MapperScan("cn.itcast.haoke.dubbo.server.mapper")
7 @Configuration
8 public class MybatisConfig {
9
10 }
11
```

4.7.3、编写service

这里编写的Service是Spring的service，不是dubbo服务，因为需要控制事务以及一些逻辑。

先定义接口：

```
1 package cn.itcast.haoke.dubbo.server.service;
2
3 import cn.itcast.haoke.dubbo.server.pojo.HouseResources;
4
5 public interface HouseResourcesService {
6
7     /**
8      *
9      * @param houseResources
10     *
11     * @return -1:输入的参数不符合要求, 0:数据插入数据库失败, 1:成功
12     */
13     int saveHouseResources(HouseResources houseResources);
14 }
15
```

编写BaseService实现：

```
1 package cn.itcast.haoke.dubbo.server.service.impl;
2
3 import cn.itcast.haoke.dubbo.server.pojo.BasePojo;
4 import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;
5 import com.baomidou.mybatisplus.core.mapper.BaseMapper;
6 import com.baomidou.mybatisplus.core.metadata.IPage;
7 import com.baomidou.mybatisplus.extension.plugins.pagination.Page;
8 import org.springframework.beans.factory.annotation.Autowired;
9
10 import java.util.Date;
11 import java.util.List;
12
13 public abstract class BaseServiceImpl<T extends BasePojo> {
14
15     @Autowired
16     private BaseMapper<T> mapper;
```



```
17
18     /**
19      * 根据id查询数据
20      *
21      * @param id
22      * @return
23      */
24     public T queryById(Long id) {
25         return this.mapper.selectById(id);
26     }
27
28     /**
29      * 查询所有数据
30      *
31      * @return
32      */
33     public List<T> queryAll() {
34         return this.mapper.selectList(null);
35     }
36
37     /**
38      * 根据条件查询一条数据
39      *
40      * @param record
41      * @return
42      */
43     public T queryOne(T record) {
44         return this.mapper.selectOne(new QueryWrapper<>(record));
45     }
46
47     /**
48      * 根据条件查询数据列表
49      *
50      * @param record
51      * @return
52      */
53     public List<T> queryListByWhere(T record) {
54         return this.mapper.selectList(new QueryWrapper<>(record));
55     }
56
57     /**
58      * 根据条件分页查询数据列表
59      *
60      * @param record
61      * @param page
62      * @param rows
63      * @return
64      */
65     public IPage<T> queryPageListByWhere(T record, Integer page, Integer rows) {
66         // 获取分页数据
67         return this.mapper.selectPage(new Page<T>(page, rows), new QueryWrapper<>
(record));
68     }
```



```
69
70     /**
71      * 保存数据
72      *
73      * @param record
74      * @return
75      */
76     public Integer save(T record) {
77         record.setCreated(new Date());
78         record.setUpdated(record.getCreated());
79         return this.mapper.insert(record);
80     }
81
82     /**
83      * 更新数据
84      *
85      * @param record
86      * @return
87      */
88     public Integer update(T record) {
89         record.setUpdated(new Date());
90         return this.mapper.updateById(record);
91     }
92
93     /**
94      * 根据id删除数据
95      *
96      * @param id
97      * @return
98      */
99     public Integer deleteById(Long id) {
100         return this.mapper.deleteById(id);
101     }
102
103     /**
104      * 根据ids批量删除数据
105      *
106      * @param ids
107      * @return
108      */
109     public Integer deleteByIds(List<Long> ids) {
110         return this.mapper.deleteBatchIds(ids);
111     }
112
113     /**
114      * 根据条件删除数据
115      *
116      * @param record
117      * @return
118      */
119     public Integer deleteByWhere(T record){
120         return this.mapper.delete(new QueryWrapper<>(record));
121     }
```



```
122  
123  
124 }  
125
```

具体实现类：

```
1 package cn.itcast.haoke.dubbo.server.service.impl;  
2  
3 import cn.itcast.haoke.dubbo.server.pojo.HouseResources;  
4 import cn.itcast.haoke.dubbo.server.service.HouseResourcesService;  
5 import org.apache.commons.lang3.StringUtils;  
6 import org.springframework.stereotype.Service;  
7 import org.springframework.transaction.annotation.Transactional;  
8  
9 @Service //这是一个Spring的服务  
10 @Transactional //开启事务  
11 public class HouseResourcesServiceImpl extends BaseServiceImpl<HouseResources>  
12     implements HouseResourcesService {  
13  
14     public int saveHouseResources(HouseResources houseResources) {  
15         // 编写校验逻辑，如果校验不通过，返回-1  
16         if (StringUtils.isBlank(houseResources.getTitle())) {  
17             return -1;  
18         }  
19         // 其他校验以及逻辑省略 .....  
20         return super.save(houseResources);  
21     }  
22  
23 }  
24
```

4.7.4、实现dubbo服务

```
1 package cn.itcast.haoke.dubbo.server.api;  
2  
3 import cn.itcast.haoke.dubbo.server.pojo.HouseResources;  
4 import cn.itcast.haoke.dubbo.server.service.HouseResourcesService;  
5 import com.alibaba.dubbo.config.annotation.Service;  
6 import org.springframework.beans.factory.annotation.Autowired;  
7  
8 @Service(version = "1.0.0") // 这是dubbo服务，对外进行暴露  
9 public class ApiHouseResourcesServiceImpl implements ApiHouseResourcesService {  
10  
11     @Autowired  
12     private HouseResourcesService houseResourcesService;  
13  
14     @Override  
15     public int saveHouseResources(HouseResources houseResources) {  
16         return this.houseResourcesService.saveHouseResources(houseResources);  
17     }  
18 }
```

```
18 }  
19
```

4.8、编写启动类

```
1 package cn.itcast.haoke.dubbo.server;  
2  
3 import org.springframework.boot.WebApplicationType;  
4 import org.springframework.boot.autoconfigure.SpringBootApplication;  
5 import org.springframework.boot.builder.SpringApplicationBuilder;  
6  
7 @SpringBootApplication  
8 public class DubboProvider {  
9  
10     public static void main(String[] args) {  
11         new SpringApplicationBuilder(DubboProvider.class)  
12             .web(WebApplicationType.NONE) // 非 web 应用  
13             .run(args);  
14     }  
15 }
```

4.9、进行测试

PROVIDERS			CONSUMERS
IP ↑	Port	Timeout(ms)	
192.168.40.1	20881		<div>dubbo://192.168.40.1:20881/cn.itcast.haoke.dubbo.server.api.ApiHouseResourcesService</div> <div>URL</div>
Rows per page: 5			1-1 of 1 < >

可以看到，已经有服务注册到了注册中心。

5、实现RESTful接口

itcast-haoke-manage-api-server工程是，为前端系统提供接口服务，是dubbo服务的消费方。

5.1、添加依赖

因为是dubbo的服务方，需要添加dubbo提供方提供的接口、pojo的依赖。

```
1 <dependency>  
2     <groupId>cn.itcast.haoke</groupId>  
3     <artifactId>itcast-haoke-manage-dubbo-server-house-resources-dubbo-  
4     interface</artifactId>  
5     <version>1.0-SNAPSHOT</version>  
6 </dependency>
```

5.2、编写service



HouseResourcesService用于调用dubbo服务。

```
1 package cn.itcast.haoke.dubbo.api.service;
2
3 import cn.itcast.haoke.dubbo.server.api.ApiHouseResourcesService;
4 import cn.itcast.haoke.dubbo.server.pojo.HouseResources;
5 import com.alibaba.dubbo.config.annotation.Reference;
6 import org.springframework.stereotype.Service;
7
8 @Service
9 public class HouseResourcesService {
10
11     @Reference(version = "1.0.0")
12     private ApiHouseResourcesService apiHouseResourcesService;
13
14     public boolean save(HouseResources houseResources){
15         int result =
16         this.apiHouseResourcesService.saveHouseResources(houseResources);
17         return result == 1;
18     }
19 }
20
```

5.3、编写controller

```
1 package cn.itcast.haoke.dubbo.api.controller;
2
3 import cn.itcast.haoke.dubbo.api.service.HouseResourcesService;
4 import cn.itcast.haoke.dubbo.server.pojo.HouseResources;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.http.HttpStatus;
7 import org.springframework.http.ResponseEntity;
8 import org.springframework.stereotype.Controller;
9 import org.springframework.web.bind.annotation.*;
10
11 @Controller
12 @RequestMapping("house/resources")
13 public class HouseResourcesController {
14
15     @Autowired
16     private HouseResourcesService houseResourcesService;
17
18     /**
19      * 新增房源
20      *
21      * @param houseResources json数据
22      * @return
23      */
24     @PostMapping
25     @ResponseBody
26     public ResponseEntity<Void> save(@RequestBody HouseResources houseResources){
27
28     }
```




```
27     try {
28         boolean bool = this.houseResourcesService.save(houseResources);
29         if(bool){
30             return ResponseEntity.status(HttpStatus.CREATED).build();
31         }
32     } catch (Exception e) {
33         e.printStackTrace();
34     }
35     return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
36 }
37
38
39 /**
40  * test
41  *
42  * @return
43  */
44 @GetMapping
45 @ResponseBody
46 public ResponseEntity<String> get(){
47     return ResponseEntity.ok("ok");
48 }
49
50 }
51
```

5.4、编写启动类

```
1 package cn.itcast.haoke.dubbo.api;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication
7 public class DubboApiApplication {
8
9     public static void main(String[] args) {
10         SpringApplication.run(DubboApiApplication.class, args);
11     }
12
13 }
14
```

5.5、编写application.properties配置文件



```
1 # spring boot application
2 spring.application.name = itcast-haoke-manage-api-server
3 server.port = 18080
4 #logging.level.root=DEBUG
5
6 # 应用名称
7 dubbo.application.name = dubbo-consumer-haoke-manage
8 # zk注册中心
9 dubbo.registry.address = zookeeper://172.16.55.185:2181
10 dubbo.registry.client = zkclient
11
```

5.6、运行启动服务

运行即可启动tomcat服务，进行测试：

http://127.0.0.1:18080/house/resources

GET POST PUT PATCH DELETE HEAD OPTIONS Other

Raw Form Headers

Raw Form Files (0) Payload

Encode payload Decode payload

[{"title": "东方曼哈顿 3室2厅 16000元", "buildingName": "2", "buildingUnit": "1", "buildingFloorNum": "1", "rent": "1111", "paymentMethod": "1", "rentMethod": "1", "coveredArea": "2", "useArea": "2", "orientation": "南", "decoration": "1", "facilities": "1,2,3,8,9", "houseDesc": "这个经纪人很懒，没写核心卖点", "contact": "张三", "mobile": "1111111111", "time": "1", "propertyCost": "11", "floor": "1/2", "houseType": "1室1厅1卫1厨1阳台", "estateId": "1005"}]

application/json Set "Content-Type" header to overwrite this value.

Clear Send

结果：

application/json Set "Content-Type" header to overwrite this value.

Status 201 Loading time: 5783 ms

Request headers Origin: chrome-extension://mhaabkcmplalpgkembnlplemmhndkgk
User-Agent: Mozilla/5.0 (Windows NT 6.3; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/70.0.3538.67 Safari/537.36
Content-Type: application/json
Accept: */*
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8,zh-TW;q=0.7
Cookie: _ga=GA1.4.548092130.1505735846

Response headers Content-Length: 0
Date: Wed, 14 Nov 2018 14:57:44 GMT

Raw Response

Word unwrap Copy to clipboard Save as file

Response does not contain any data.



测试数据：

```
1 url: http://127.0.0.1:18080/house/resources
2 数据：
3 {"title": "东方曼哈顿 3室2厅 16000
  元", "buildingNum": "2", "buildingUnit": "1", "buildingFloorNum": "1", "rent": "1111", "paymentMethod": "1", "rentMethod": "1", "coveredArea": "2", "useArea": "2", "orientation": "南", "decoration": "1", "facilities": "1,2,3,8,9", "houseDesc": "这个经纪人很懒，没写核心卖点", "contact": "张三", "mobile": "11111111111", "time": "1", "propertyCost": "11", "floor": "1/2", "houseType": "1室1厅1卫1厨1阳台", "estateId": "1005"}
4
```

下面，可以整合前端进行开发了。

6、整合前端系统

6.1、增加房源标题

之前的前端页面实现中，缺少了标题一项，现补上。

```
1 <FormItem {...formItemLayout} label="房源标题">
2   {getFieldDecorator('title', {rules: [{ required: true,
  message: "此项为必填项" }]})( <Input style={{ width: '100%' }} /> )
3 </FormItem>
```

效果：

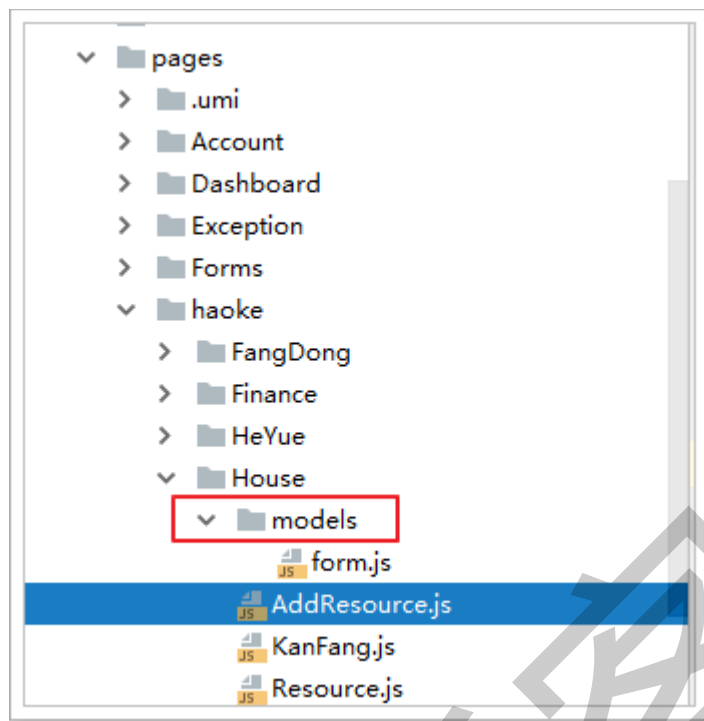
楼盘名称： 上海阳城

楼盘地址： 上海市,上海市,闵行区,罗锦路888弄

房源标题： 东方曼哈顿 3室2厅 16000元

楼栋： 2 栋 1 单元 1 门牌号

6.2、增加model



form.js文件：

```
1 import { routerRedux } from 'dva/router';
2 import { message } from 'antd';
3 import { addHouseResource } from '@services/haoke';
4
5 export default {
6   namespace: 'house',
7
8   state: {
9
10  },
11
12  effects: {
13    *submitHouseForm({ payload }, { call }) {
14      yield call(addHouseResource, payload);
15      message.success('提交成功');
16    }
17  },
18
19  reducers: {
20    saveStepFormData(state, { payload }) {
21      return {
22        ...state
23      };
24    },
25  },
26 };
27
```

6.3、增加services



haoke.js，用于请求服务并且处理业务逻辑。

```
1 import request from '@/utils/request';
2
3 export async function addHouseResource(params) {
4   return request('/haoke/house/resources', {
5     method: 'POST',
6     body: params
7   });
8 }
9
```

6.4、修改表单提交地址

AddResource.js：

```
1 dispatch({
2   type: 'house/submitHouseForm',
3   payload: values,
4 });
```

6.5、通过反向代理解决跨域问题

由于我们前端系统和后台服务系统的端口不同，会导致跨域问题，我们通过umi提供的反向代理功能解决这个问题。

配置地址：<https://umijs.org/zh/config/#proxy>

在config.js中进行配置proxy：

```
1 proxy: {
2   '/haoke/': {
3     target: 'http://127.0.0.1:18080',
4     changeOrigin: true,
5     pathRewrite: { '^/haoke/': '' }
6   }
7 },
```

配置代理后，以/haoke开头的请求，就会被代理。

代理效果是这样的：

请求：<http://localhost:8000/haoke/house/resources>

实际：<http://127.0.0.1:18080/house/resources>

6.6、测试



楼盘名称: 上海康城

楼盘地址: 上海市,上海市,闵行区,莘松路958弄

房源标题: 康城 2室 精装修

楼栋: 1 栋 2 单元 3 门牌号

租金: 2000 元/月

支付方式: 付一押一

租赁方式: 整租

户型: 2 室 1 厅 1 卫 1 厨 1 阳台

建筑面积: 80 平米

使用面积: 60 平米

楼层: 第 5 层 总 10 层



提交成功

联系人: 王五

手机号: 18888888888

看房时间: 上午

物业费: 1 元/平

提交

保存

Name

resources
/haoke/house

Headers Preview Response Cookies Timing

General

Request URL: http://localhost:8000/haoke/house/resources

Request Method: POST

Status Code: 201 Created

Remote Address: 127.0.0.1:8000

Referrer Policy: no-referrer-when-downgrade

Response Headers (4)

Request Headers (13)

Request Payload view source

```
{title: "康城 2室 精装修", buildingNum: "1", buildingUnit: "2",
  buildingFloorNum: "3",
  buildingNum: "1",
  buildingUnit: "2",
  contact: "王五",
  coveredArea: "80",
  decoration: "1",
  estateId: "1002",
  facilities: "1,2,3,4,6",
  floor: "5/10",
  houseDesc: "拎包入住, 好房不等人!",
  houseType: "2室1厅1卫1厨1阳台",
  mobile: "18888888888",
  orientation: "南",
  paymentMethod: "1",
  propertyCost: "1",
  rent: "2000",
  rentMethod: "1",
  time: "1",
  title: "康城 2室 精装修",
  useArea: "60"}
```

1 requests | 122 B transferred



id	title	estate_id	building_num	building_unit	building_floor_num	rent	re
1	繁华地段，品质小区，	1001	12	2	603	1900	
2	繁华地段，品质小区，	1001	12	2	603	1900	
3	东方曼哈顿 3室2厅 160	1005	2	1	1	1111	
4	东方曼哈顿 3室2厅 160	1005	2	1	1	1111	
5	东方曼哈顿 3室2厅 160	1005	2	1	1	1111	
6	康城 2室 精装修	1002	1	2	3	2000	

可以看到，数据以及插入到数据库，说明整个流程已经跑通了。