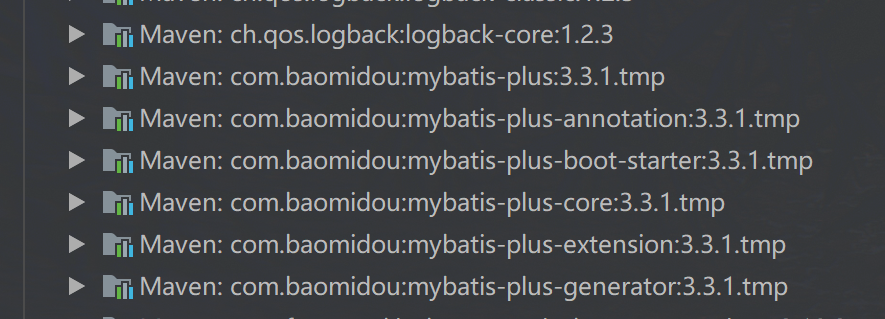
# 1 SpringBoot集成MyBatis-Plus

引入依赖：

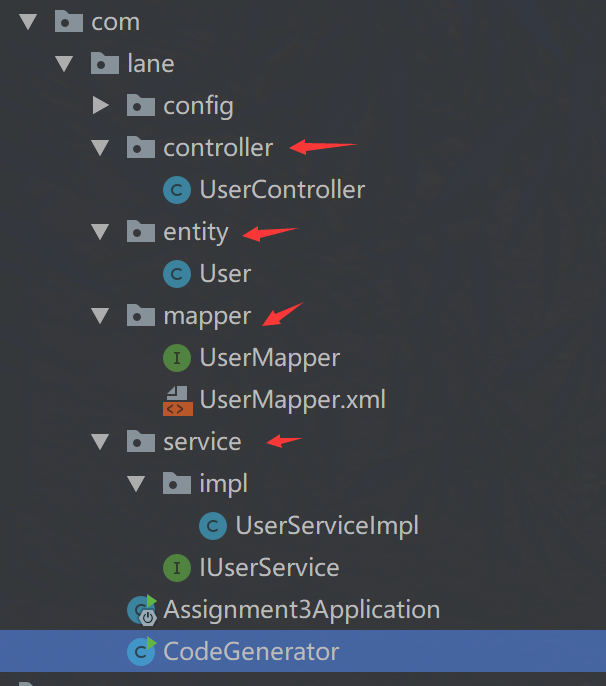
<dependency>  
 <groupId>com.baomidou</groupId>  
 <artifactId>mybatis-plus-generator</artifactId>  
 <version>3.3.1.tmp</version>  
</dependency>



# 2 根据数据表使用代码生成器完成各组件的自动生成

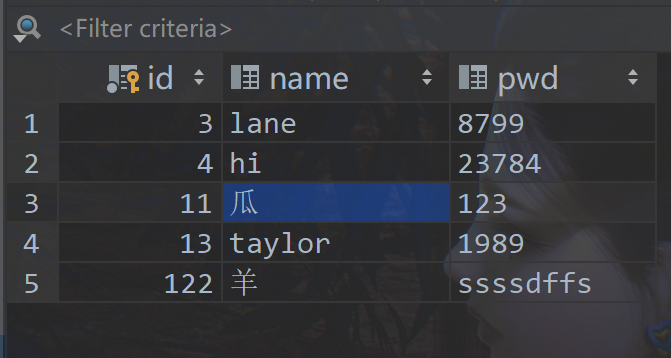
见 com.lane.CodeGenerator.java

运行后生成在com.lane下生成，entity、controller、mapper（包括\*Mapper.xml)，service包，包括相关文件的生成。



# 3 基于Service CRUD接口完成数据操作（含条件查询）

有一张user(id,name,pwd)表,当前状态如下。



对应的实体类为User（已经用代码生成器自动生成）。

增删查改操作均基于IService接口，IUserService继承了此接口。

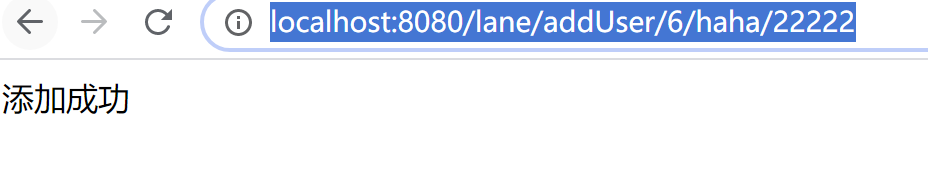
public interface IUserService extends IService<User> {  
}  
  
  
@Autowired  
private IUserService userService;

## 3.1 增

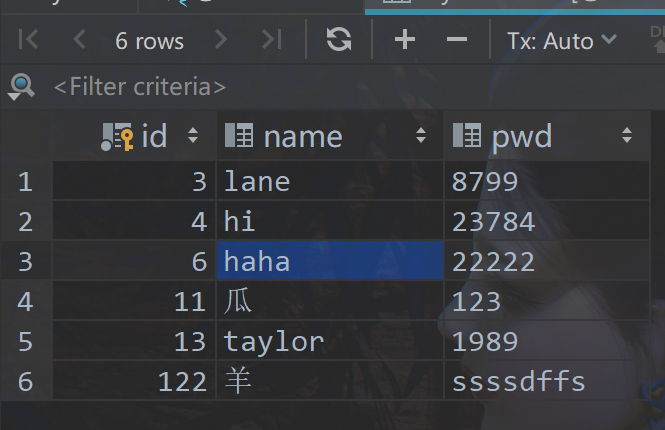
//增  
@RequestMapping("/addUser/{id}/{name}/{pwd}")  
public String addUser(@PathVariable("id") int id,@PathVariable("name") String name,@PathVariable("pwd") String pwd)  
{  
 User user=new User(id,name,pwd);  
 userService.save(user);  
 return "添加成功";  
}

测试：在地址栏输入：http://localhost:8080/lane/addUser/6/haha/22222 （其中路径上第一个参数为要增加的user的id，第二个参数为name，第三个参数为pwd）

结果：



观察数据库的user表，已经添加进了：



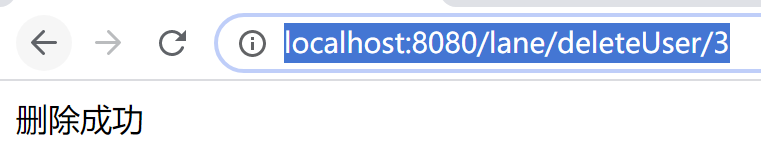
## 3.2 删

根据id删除user。

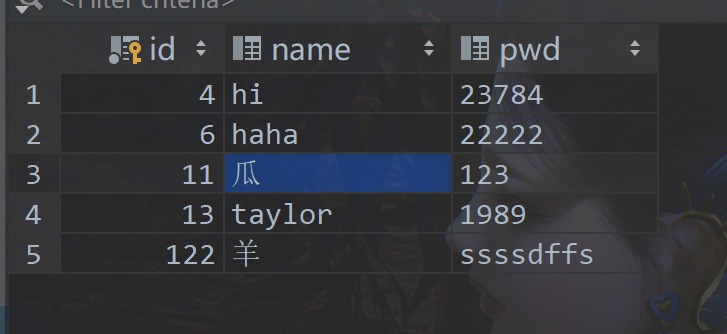
//删  
@RequestMapping("/deleteUser/{id}")  
public String deleteUser(@PathVariable("id") int id) {  
 userService.removeById(id);  
 return "删除成功";  
}

测试：在地址栏输入：http://localhost:8080/lane/deleteUser/3（其中路径上第一个参数为要删除的user的id）

结果：



观察数据库的user表，id为3的数据已经删除了。



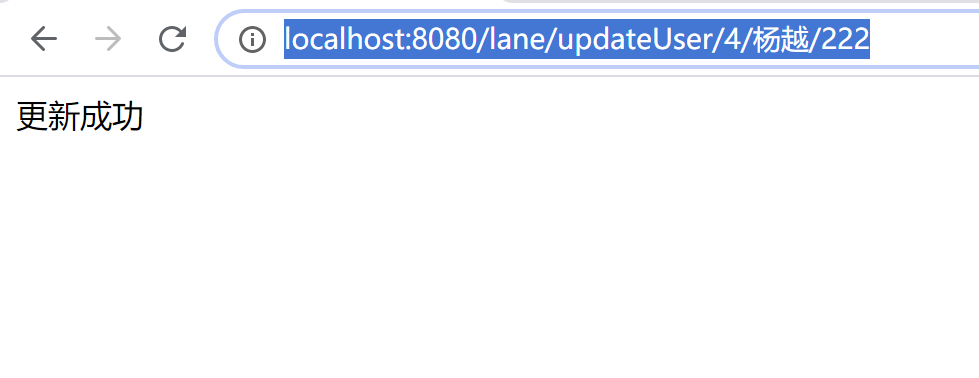
## 3.3 改

根据id更改user

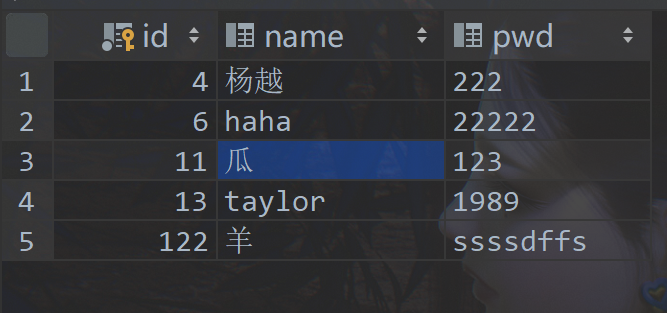
//改  
@RequestMapping("/updateUser/{id}/{name}/{pwd}")  
public String updateUser(@PathVariable("id") int id,@PathVariable("name") String name,@PathVariable("pwd") String pwd)  
{  
 User user=new User(id,name,pwd);  
 userService.updateById(user);  
 return "更新成功";  
}

测试：在地址栏输入：<http://localhost:8080/lane/updateUser/4/杨越/222> （其中路径上第一个参数为要更改的user的id，第二个参数为name，第三个参数为pwd）

结果：



观察数据库的user表，id为4的user已经被修改了：



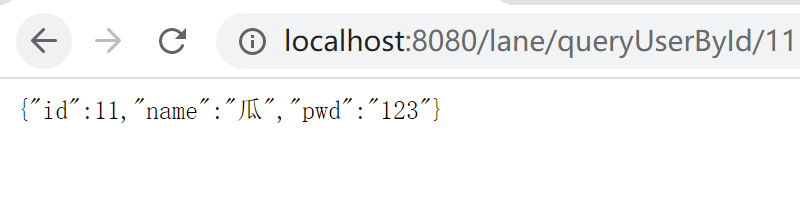
## 3.4 查

### 3.4.1 根据id查找数据

//查  
@RequestMapping("/queryUserById/{id}")  
public User getUser(@PathVariable("id") int id)  
{  
 User user=userService.getById(id);  
 return user;  
}

测试：在地址栏输入：http://localhost:8080/lane/queryUserById/11（其中路径上第一个参数为要查找的user的id）

结果：

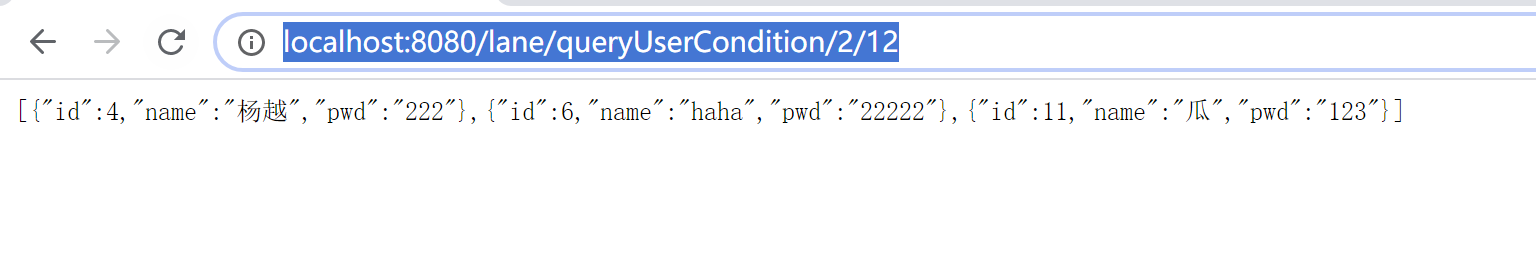


### 3.4.2 条件查询

//条件查询  
@RequestMapping("/queryUserCondition/{low}/{high}")  
public List<User> queryUserCondition(@PathVariable("low") int low,@PathVariable("high") int high) {  
 QueryWrapper<User> queryWrapper = new QueryWrapper<>();  
 queryWrapper.ge("id", low).le("id",high);  
 List<User> list=userService.list(queryWrapper);  
 return list;  
}

查找id大小在某个区间段的user。

测试：在地址栏输入http://localhost:8080/lane/queryUserCondition/2/12 （其中路径上第一个参数为要查询id的区间的最小值，第二个参数为最大值，也就是查找id值大小在2-12之间的user）

结果：

# 4 实现数据分页查询

## 4.1 分页插件配置

@Configuration  
public class MyBatisPlusConfig {  
 /\*\*  
 \* mybatis-plus分页插件  
 \*/  
 @Bean  
 public PaginationInterceptor paginationInterceptor() {  
 PaginationInterceptor paginationInterceptor = new PaginationInterceptor();  
 return paginationInterceptor;  
 }  
}

## 4.2 分页查询

//分页查询  
@RequestMapping("/page/{current}/{size}")  
public List<User> getPage(@PathVariable("current") int current,@PathVariable("size") int size)  
{  
  
 Map<String,Object> map = new HashMap<>();  
 QueryWrapper<User> queryWrapper = new QueryWrapper<>();  
 queryWrapper.orderByDesc("id");//按id降序  
 Page<User> page = new Page<>(current,size); // 查询第current页，每页大小为size  
 IPage<User> iPage = userMapper.selectPage(page,queryWrapper);  
 return iPage.getRecords();  
}

在地址栏输入：http://localhost:8080/lane/page/1/3 （路径上第一个参数表示当前页，第二个参数表示页的大小）

结果：