用户注册（持久层）

1 用户-创建数据表

用户注册是由一张表来进行维护的，这张表位用户表（t_user）

1.使用 use 命令先选中 store 数据库。

```
use store;
```

2.在 store 数据库中创建 t_user 用户数据表。

```
CREATE TABLE t_user (
    uid INT AUTO_INCREMENT COMMENT '用户 id',
    username VARCHAR(20) NOT NULL UNIQUE COMMENT '用户名',
    password CHAR(32) NOT NULL COMMENT '密码',
    salt CHAR(36) COMMENT '盐值',#密码加密
    phone VARCHAR(20) COMMENT '电话号码',
    email VARCHAR(30) COMMENT '电子邮箱',
    gender INT COMMENT '性别:0-女，1-男',
    avatar VARCHAR(50) COMMENT '头像',
    is_delete INT COMMENT '是否删除：0-未删除，1-已删除',
    created_user VARCHAR(20) COMMENT '日志-创建人',
    created_time DATETIME COMMENT '日志-创建时间',
    modified_user VARCHAR(20) COMMENT '日志-最后修改执行人',
    modified_time DATETIME COMMENT '日志-最后修改时间',
    PRIMARY KEY (uid)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

打开 navicat:

出现以下说明成功：

2 用户-创建实体类

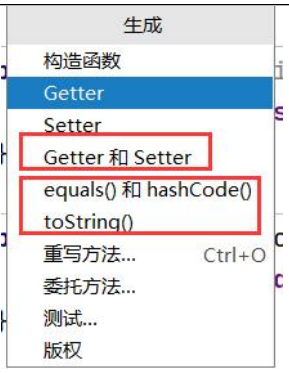
1.项目中许多实体类都会有日志相关的四个属性，所以在创建实体类之前，应先创建这些实体类的基类，将 4 个日志属性声明在基类中。在 com.fx.store.entity 包下创建 BaseEntity 类，作为实体类的基类。

```
package com.fx.store.entity;
import java.io.Serializable;
import java.util.Date;

//作为所有实体类的基类
public class BaseEntity implements Serializable {
    private String createdUser;
    private Date createdTime;
    private String modifiedUser;
    private String modifiedTime;
/**
 *    方法分为三部分：1、私有属性的 getter、setter 方法
```

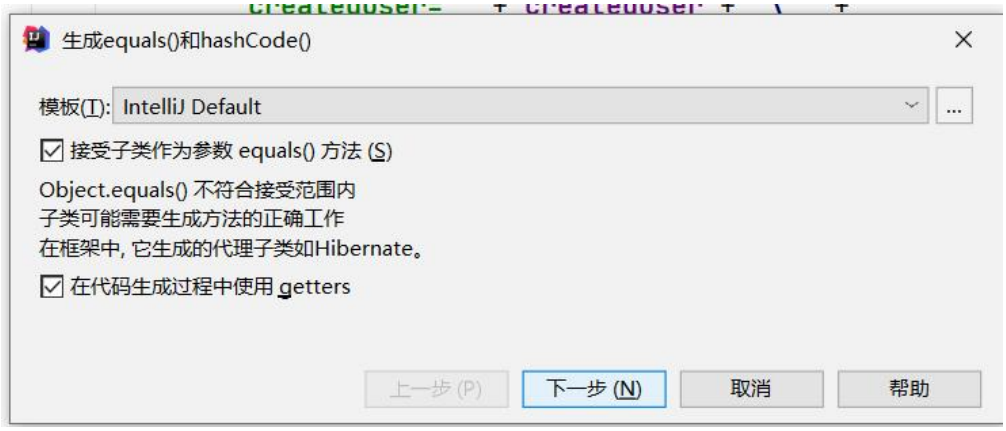
```
 *                      2、equals 和 hashcode 方法
 *                      3、toString 方法
 */

    public String getCreatedUser() {
        return createdUser;
    }

    public void setCreatedUser(String createdUser) {
        this.createdUser = createdUser;
    }

    public Date getCreatedTime() {
        return createdTime;
    }

    public void setCreatedTime(Date createdTime) {
        this.createdTime = createdTime;
    }

    public String getModifiedUser() {
        return modifiedUser;
    }

    public void setModifiedUser(String modifiedUser) {
        this.modifiedUser = modifiedUser;
    }

    public String getModifiedTime() {
        return modifiedTime;
    }

    public void setModifiedTime(String modifiedTime) {
        this.modifiedTime = modifiedTime;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof BaseEntity)) return false;

        BaseEntity that = (BaseEntity) o;

        if (getCreatedUser() != null ? !getCreatedUser().equals(that.getCreatedUser()) :
```

```
that.getCreatedUser() != null)
                return false;
            if (getCreatedTime() != null ? !getCreatedTime().equals(that.getCreatedTime()) :
that.getCreatedTime() != null)
                return false;
            if (getModifiedUser() !=
null ? !getModifiedUser().equals(that.getModifiedUser()) : that.getModifiedUser() != null)
                return false;
            return getModifiedTime() != null ?
getModifiedTime().equals(that.getModifiedTime()) : that.getModifiedTime() == null;
        }

        @Override
        public int hashCode() {
            int result = getCreatedUser() != null ? getCreatedUser().hashCode() : 0;
            result = 31 * result + (getCreatedTime() != null ? getCreatedTime().hashCode() :
0);
            result = 31 * result + (getModifiedUser() != null ?
getModifiedUser().hashCode() : 0);
            result = 31 * result + (getModifiedTime() != null ?
getModifiedTime().hashCode() : 0);
            return result;
        }

        @Override
        public String toString() {
            return "BaseEntity{" +
                    "createdUser='" + createdUser + '\'' +
                    ", createdTime=" + createdTime +
                    ", modifiedUser='" + modifiedUser + '\'' +
                    ", modifiedTime='" + modifiedTime + '\'' +
                    '}';
        }
}
```

鼠标右键找到生成--->



注意生成 equals 和 hashcode 方法时的模板。



2.创建 com.cy.store.entity.User 用户数据的实体类，继承自 BaseEntity 类，在类中声明与数据表中对应的属性。

```
package com.fx.store.entity;

import java.io.Serializable;

/**
 * 用户的实体类
 * 建议使用包装类定义，便于后期调用包装类相关的 api 做逻辑判断
 */
public class User extends BaseEntity implements Serializable {
    private Integer uid;    //'用户 id'
    private String username;    //'用户名'
    private String password; //'密码',
    private String salt;    //'盐值',
    private String phone;    //'电话号码',
    private String email;    //'电子邮箱',
    private Integer gender;    //'性别:0-女，1-男',
    private String avatar;    //'头像',
    private Integer isDelete;    //'是否删除：0-未删除，1-已删除',

/**任何实体类都要有以下方法：    1、私有属性的 getter、setter 方法
                              2、equals(比较两个对象是否相等，自定义比较规则)和
hashcode 方法(地址输出)
                              3、toString 方法（便于测试使用，输出对象）
 */
    public Integer getUid() {
        return uid;
    }

    public void setUid(Integer uid) {
        this.uid = uid;
```

```
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getSalt() {
        return salt;
    }

    public void setSalt(String salt) {
        this.salt = salt;
    }

    public String getPhone() {
        return phone;
    }

    public void setPhone(String phone) {
        this.phone = phone;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public Integer getGender() {
        return gender;
```

```
    }

    public void setGender(Integer gender) {
        this.gender = gender;
    }

    public String getAvatar() {
        return avatar;
    }

    public void setAvatar(String avatar) {
        this.avatar = avatar;
    }

    public Integer getIsDelete() {
        return isDelete;
    }

    public void setIsDelete(Integer isDelete) {
        this.isDelete = isDelete;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof User)) return false;
        if (!super.equals(o)) return false;

        User user = (User) o;

        if (getUid() != null ? !getUid().equals(user.getUid()) : user.getUid() != null) return
false;
        if (getUsername() != null ? !getUsername().equals(user.getUsername()) :
user.getUsername() != null)
            return false;
        if (getPassword() != null ? !getPassword().equals(user.getPassword()) :
user.getPassword() != null)
            return false;
        if (getSalt() != null ? !getSalt().equals(user.getSalt()) : user.getSalt() != null) return
false;
        if (getPhone() != null ? !getPhone().equals(user.getPhone()) : user.getPhone() !=
null) return false;
        if (getEmail() != null ? !getEmail().equals(user.getEmail()) : user.getEmail() != null)
return false;
```

```
        if (getGender() != null ? !getGender().equals(user.getGender()) : user.getGender() !=
null) return false;
        if (getAvatar() != null ? !getAvatar().equals(user.getAvatar()) : user.getAvatar() !=
null) return false;
        return getIsDelete() != null ? getIsDelete().equals(user.getIsDelete()) :
user.getIsDelete() == null;
    }

    @Override
    public int hashCode() {
        int result = super.hashCode();
        result = 31 * result + (getUid() != null ? getUid().hashCode() : 0);
        result = 31 * result + (getUsername() != null ? getUsername().hashCode() : 0);
        result = 31 * result + (getPassword() != null ? getPassword().hashCode() : 0);
        result = 31 * result + (getSalt() != null ? getSalt().hashCode() : 0);
        result = 31 * result + (getPhone() != null ? getPhone().hashCode() : 0);
        result = 31 * result + (getEmail() != null ? getEmail().hashCode() : 0);
        result = 31 * result + (getGender() != null ? getGender().hashCode() : 0);
        result = 31 * result + (getAvatar() != null ? getAvatar().hashCode() : 0);
        result = 31 * result + (getIsDelete() != null ? getIsDelete().hashCode() : 0);
        return result;
    }

    @Override
    public String toString() {
        return "User{" +
                "uid=" + uid +
                ", username='" + username + '\'' +
                ", password='" + password + '\'' +
                ", salt='" + salt + '\'' +
                ", phone='" + phone + '\'' +
                ", email='" + email + '\'' +
                ", gender=" + gender +
                ", avatar='" + avatar + '\'' +
                ", isDelete=" + isDelete +
                '}';
    }
}
```

3 用户-注册-持久层

通过 mybatis 来操作数据库，在做 mybatis 开发的流程。

3.1 规划需要执行的 SQL 语句

1.用户注册的本质是**向用户表中插入数据**，需要执行的 SQL 语句大致是

```
INSERT INTO t_user (usename，password 除了 uid 以外的字段列表) VALUES (匹配的值列表)
```

2.由于数据表中用户名字段被设计为 UNIQUE，在执行插入数据之前，还应该检查该用户名是否已经被注册，因此需要有“根据用户名查询用户数据”的功能。需要执行的 SQL 语句大致是：

```
SELECT * FROM t_user WHERE username=?
```

3.3 接口与抽象方法

1.定义 Mapper 接口，在项目的目录结构下首先创建一个 mapper 包，在这个包下再根据不同的功能来创建 mapper 接口。创建 com.cy.store.mapper.UserMapper 接口，并在接口中添加定义上述两个 SQL 语句的抽象方法。

```
package com.fx.store.mapper;

import com.fx.store.entity.User;

public interface UserMapper {
    /**
     * 插入用户的数据
     * @param user 用户的数据
     * @return 受影响的行数（增删改都有受影响的行数作为范围值，可以根据返回值判断是否执行成功）
     */
    Integer insert(User user);//通过插入后影响的行数来判断是否插入成功，因此返回值定义为 integer

    /**
     * 根据用户明来查询用户的数据
     * @param username 用户名
     * @return 如果找到对应的用户则返回这个用户的数据，如果没有找到则返回 null 值
     */
    User findByUserName(String username);
}
```

2.由于这是项目中第一次创建持久层接口，还应在 StoreApplication 启动类之前添加@MapperScan("com.cy.store.mapper")注解，以配置接口文件的位置。

MyBatis 与 Spring 整合后需要实现实体和数据表的映射关系。实现实体和数据表的映射关系可以在 Mapper 接口上添加@Mapper 注解。但建议以后直接在 SpringBoot 启动类中加@MapperScan("mapper包") 注解，这样会比较方便，不需要对每个 Mapper 都添加@Mapper 注解。

```
@SpringBootApplication
//MapperScan 作用：用于指定当前项目中 Mapper 接口路径的位置（括号中就是具体的位置）
//在项目启动时会自动加载所有的接口
```

```
@MapperScan("com.fx.store.mapper")
public class DemoSpringbootApplication {

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

3.4 配置 SQL 映射

1.定义 xml 映射文件，与对应的接口进行关联。在 src/main/resources 下创建 mapper 文件夹，并在该文件夹下创建 UserMapper.xml 映射文件，进行以上两个抽象方法的映射配置。
2.创建接口对应的映射文件，遵循和接口名称保持一致。
Mybatis 官网的示例

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper
  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="org.mybatis.example.BlogMapper">
  <select id="selectBlog" resultType="Blog">
    select * from Blog where id = #{id}
  </select>
</mapper>
```

3.选中 mapper 包，右键新建一个 file，取名为 UserMapper.xml，将上述示例赋值进去，并做以下修改（1、删除 select 语句，2、修改 namespace）。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper
        PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<!--namespace 属性：用于指定当前的映射文件和哪个接口进行映射，需要指定接口的文件路径，需要标注包的完整路径接口-->
<mapper namespace="com.fx.store.mapper.UserMapper">

</mapper>
```

4.配置接口中的方法对应到 SQL 语句上，需要借助标签来完成，insert\update\delete\select，对应 SQL 语句的增删改查操作。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper
        PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<!--namespace 属性：用于指定当前的映射文件和哪个接口进行映射，需要指定接口的文件路径，需要标注包的完整路径接口-->
<mapper namespace="com.fx.store.mapper.UserMapper">
```

```
    <!--自定义映射规则:resultMap 标签来完成映射规则的定义-->
    <!--
        id 属性：表示给这个映射负责分配唯一的 id 值，对应的就是 resultMap="id 属性的值"
属性的值
        type 属性：取值时一个类，表示的是数据库中的查询结果与 java 中的哪个实体类进行
结果集的映射
    -->
    <resultMap id="userEntityMap" type="com.fx.store.entity.User">
        <!--将表的字段和类的属性不一致的字段进行匹配指定，名称一致的字段可以省略不写
            配合完成名称不一致的映射
            column 属性：表示数据库表中的资源名称
            property：表示 java 定义类中的属性名称
        -->
        <!--在定义映射规则时主键时不可以省略的-->
        <id column="uid" property="uid"></id>
        <result column="is_delete" property="isDelete"></result>
        <result column="created_user" property="createdUser"></result>
        <result column="created_time" property="createdTime"></result>
        <result column="modified_user" property="modifiedUser"></result>
        <result column="modified_time" property="modifiedTime"></result>
    </resultMap>

    <!--id 属性：表示映射的接口中方法的名称，直接在标签内部来编写 SQL 语句-->
    <!--
        useGeneratedKeys 属性：表示开启某个字段的值递增（通常主键设置为递增）
        keyProperty 属性：表示将表中的哪个字段作为主键递增
    -->
    <insert id="insert" useGeneratedKeys="true" keyProperty="uid">
        INSERT INTO t_user
(username,password,salt,phone,email,gender,avatar,is_delete,created_user,created_time,modified_user,
modified_time)
        VALUES (#{username},#{password},#{salt}, #{phone}, #{email}, #{gender},#{avatar},
#{isDelete}, #{createdUser}, #{createdTime}, #{modifiedUser},#{modifiedTime})
    </insert>
    <!--select 语句在执行的时候，查询的结果时一个对象，多个对象-->
    <!--
        resultType：表示查询的结果集类型，只需要指定对应映射类的类型，并且包含完整的
包接口：resultType="com.fx.store.entity.user"
        resultMap:表示当表的资源和类对象属性字段不一致时，自定义查询集的映射规则
    -->
    <select id="findByUserName" resultMap="userEntityMap">
        SELECT * FROM t_user WHERE username = #{username}
    </select>
</mapper>
```

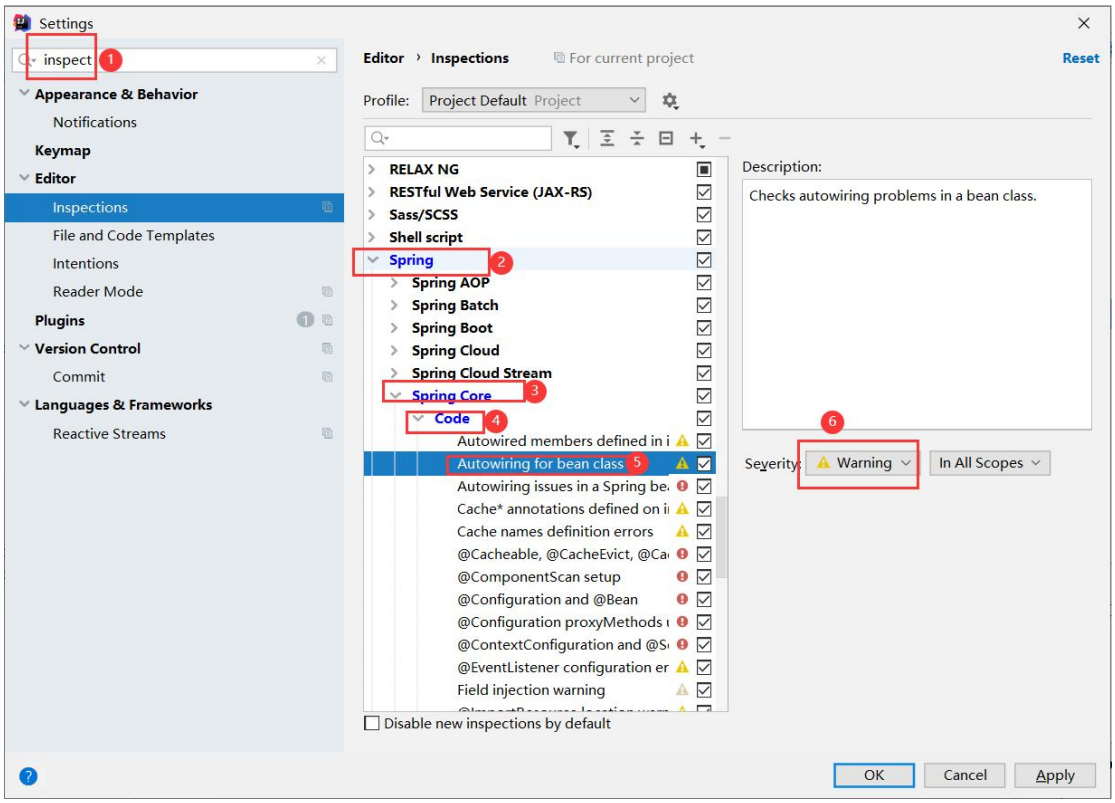
5.单元测试：每个独立的层编写完毕后需要编写单元测试方法，来测试当前的功能。完成后及时执行单元测试，检查以上开发的功能是否可正确运行。在 src/test/java 下创建 com.fx.store.mapper.UserMapperTests 单元测试类，在测试类的声明之前添加 @RunWith(SpringRunner.class)和@SpringBootTest 注解，并在测试类中声明持久层对象，通过自动装配来注入值。

```
package com.fx.store.mapper;

import com.fx.store.entity.User;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

/**
 * 自定义测试类需要声明注解：@SpringBootTest：表示当前的类是一个测试类，不会随项目一块打包
 */

@SpringBootTest

public class UserMapperTests {
// idea 有检测的功能，接口是不能直接创建 Bean
    @Autowired
    private UserMapper userMapper;

    /**
     * 单元测试方法：可以单独独立运行，而不需要启动整个项目，可以做单元测试，提升代码的测试效率
     * 1、必须被@Test 注解修饰
     * 2、返回值类型必须是 void 类型，否则会报错
     * 3、方法的参数泪飙不指定任何类型
     * 4、方法的访问修饰符必须是 public
     */
    @Test
    public void insert(){
        User user = new User();
        user.setUsername("tom");
        user.setPassword("123");
        Integer rows = userMapper.insert(user);
        System.out.println(rows);
    }
    @Test
    public void findByUserName(){
```

```
            User user = userMapper.findByUserName("tom");
            System.out.println(user);
        }
}
```

```
public class UserMapperTests {
// idea有检测的功能，接口是不能直接创建Bean
    @Autowired
    private UserMapper userMapper;
```

如果 userMapper 报红，则打开 setting，通过下图方式修改注解的权限为 warning，或者将注解改为 Resource。



如果出现 org.apache.ibatis.binding.BindingException: Invalid bound statement (not found)异常可能原因：

1.在 resources 文件加下创建的 mapper 文件夹类型没有正确选择（eclipse 选择 Folder，idea 选择 Directory）。

2.映射文件的 mapper 标签的 namespace 属性没有正确映射到 dao 层接口，或者 application.properties 中的属性 mybatis.mapper-locations 没有正确配置 xml 映射文件。

```
#下面这些内容是为了让MyBatis映射
#指定Mybatis的Mapper文件
mybatis.mapper-locations=classpath:mapper/*.xml
#指定Mybatis的实体目录
mybatis.type-aliases-package=com.fx.store.mybatis.entity
```