用户-注册-业务层

4.1 业务的定位

1.业务：一套完整的数据处理过程，通常表现为用户认为的一个功能，但是在开发时对应多项数据操作。在项目中，通过业务控制每个“功能”（例如注册、登录等）的处理流程和相关逻辑。
2.流程：先做什么，再做什么。例如：注册时，需要先判断用户名是否被占用，再决定是否完成注册。
3.逻辑：能干什么，不能干什么。例如：注册时，如果用户名被占用，则不允许注册；反之，则允许注册。
4.业务的主要作用是保障数据安全和数据的完整性、有效性。

4.2 规划异常

关于异常

1.请列举你认识的不少于十种异常

RuntimeException 异常，用户操作过程中所产生的任何异常都可以作为这个异常的子类，再去定义具体的异常类型来继承这个异常。（解决异常笼统，统一管理，建立异常机制）

```
Throwable
    Error
        OutOfMemoryError(OOM)
    Exception
        SQLException
        IOException
            FileNotFoundException
        RuntimeException（运行过程中出现的异常（密码错误、用户名占用））
            NullPointerException
            ArithmeticException
            ClassCastException
            IndexOutOfBoundsException
                ArrayIndexOutOfBoundsException
                StringIndexOutOfBoundsException
```

2.异常的处理方式和处理原则：

异常的处理方式有：捕获处理(try...catch...finally)，声明抛出(throw/throws)。如果当前方法适合处理，则捕获处理；如果当前方法不适合处理，则声明抛出。

异常规划

1.为了便于统一管理自定义异常，做以下包创建：

在 ex 包下应先创建 com.fx.store.service.ex.ServiceException 自定义异常的基类异常，继承自 RuntimeException 类，并从父类生成子类的五个构造方法。

```
package com.fx.store.service.ex;
/**
 * 业务层异常的基类
 */
public class ServiceException extends RuntimeException{
    //重写父接口的 5 个构造方法（ctrl+o）
    public ServiceException() {//无参构造
        super();
    }

    public ServiceException(String message) {//抛出信息
        super(message);
    }

    public ServiceException(String message, Throwable cause) {//Throwable 是
RuntimeException 的父类
        super(message, cause);
    }

    public ServiceException(Throwable cause) {
        super(cause);
    }

    protected ServiceException(String message, Throwable cause, boolean
enableSuppression, boolean writableStackTrace) {
        super(message, cause, enableSuppression, writableStackTrace);
    }
}
```

根据不同的功能来定义具体的异常的类型，统一的去继承 ServiceException。

2.当用户进行注册时，可能会因为用户名被占用而导致无法正常注册，此时需要抛出用户名被占用的异常，因此可以设计一个用户名重复的 com.fx.store.service.ex.UsernameDuplicateException 异常类，继承自 ServiceException 类，并从父类生成子类的五个构造方法。

```
package com.fx.store.service.ex;

/**
 * 用户名被占用异常
 */
public class UsernameDuplicateException extends ServiceException{
    //alt+insert ----override methods

    public UsernameDuplicateException() {
        super();
    }

    public UsernameDuplicateException(String message) {
        super(message);
    }

    public UsernameDuplicateException(String message, Throwable cause) {
        super(message, cause);
    }

    public UsernameDuplicateException(Throwable cause) {
        super(cause);
    }

    protected UsernameDuplicateException(String message, Throwable cause, boolean
enableSuppression, boolean writableStackTrace) {
        super(message, cause, enableSuppression, writableStackTrace);
    }
}
```

3.在用户进行注册时，会执行数据库的 INSERT 操作，该操作也是有可能失败的。则创建 cn.tedu.store.service.ex.InsertException`异常类，继承自 ServiceException 类，并从父类生成子类的 5 个构造方法。

```
package com.fx.store.service.ex;
/** 插入数据的异常 */
public class InsertException extends ServiceException{
    public InsertException() {
        super();
    }

    public InsertException(String message) {
```

```
        super(message);
    }

    public InsertException(String message, Throwable cause) {
        super(message, cause);
    }

    public InsertException(Throwable cause) {
        super(cause);
    }

    protected InsertException(String message, Throwable cause, boolean enableSuppression,
boolean writableStackTrace) {
        super(message, cause, enableSuppression, writableStackTrace);
    }
}
```

4.所有的自定义异常，都应是 RuntimeException 的子孙类异常。项目中目前异常的继承结构是见下。

```
RuntimeException
    -- ServiceException
        -- UsernameDuplicateException
        -- InsertException
```

4.3 接口与抽象方法

接口命名建议采用大写的 I 开头，后面再跟上业务名称。

1.在 service 包下先创建 com.fx.store.service.IUserService 业务层接口，并在接口中添加抽象方法。

```
package com.fx.store.service;
import com.fx.store.entity.User;
/** 处理用户数据的业务层接口 */
public interface IUserService {
/**
* 用户注册
* @param user 用户数据
*/
   void reg(User user);
}
```

2.创建业务层接口目的是为了解耦。关于业务层的抽象方法的设计原则。

```
1.仅以操作成功为前提来设计返回值类型，不考虑操作失败的情况；
2.方法名称可以自定义，通常与用户操作的功能相关；
3.方法的参数列表根据执行的具体业务功能来确定，需要哪些数据就设计哪些数据。通常情况下，参数需要足以调用持久层对应的相关功能；同时还要满足参数是客户端可以传递给控制器的；
4.方法中使用抛出异常的方式来表示操作失败。
```

4.4 实现抽象方法

1.创建 com.fx.store.service.impl.UserServiceImpl 业务层实现类，并实现 IUserService 接口。在类之前添加@Service 注解，并在类中添加持久层 UserMapper 对象。

```
package com.fx.store.service.impl;

import com.fx.store.entity.User;
import com.fx.store.mapper.UserMapper;
import com.fx.store.service.IUserService;
import com.fx.store.service.ex.InsertException;
import com.fx.store.service.ex.UsernameDuplicateException;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.Date;

/**
 * 用户模块业务层的实现类
 */
@Service//Service 注解，将当前类的对象交给 spring 来管理，自动创建类的对象以及对象的维护
public class IUserServiceImpl implements IUserService {
    @Autowired
    private UserMapper userMapper;
    @Override
    public void reg(User user) {
        /**
         * 1、根据参数 user 对象获取注册的用户名
         * 2、调用持久层的 User findByUsername(String username)方法，根据用户名查询用户数据
         * 3、判断查询结果是否不为 null
         * 4、是：表示用户名已被占用，则抛出 UsernameDuplicateException 异常
         */
        //通过 user 参数来获取传递过来的 username
        String username = user.getUsername();
        //调用 findByUsername（username）判断用户是否被注册过
        User result = userMapper.findByUserName(username);
        if (result != null){
            //抛出异常
            throw new UsernameDuplicateException("用户名被占用");
        }

        // 补全数据：isDelete(0)
        user.setIsDelete(0);
```

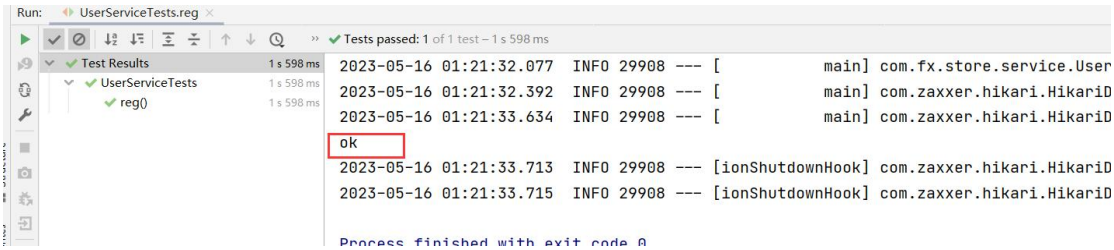
```
            // 补全数据：4 项日志属性
            user.setCreatedUser(user.getUsername());
            user.setModifiedUser(user.getUsername());
            Date date = new Date();
            user.setCreatedTime(date);
            user.setModifiedTime(date);

            /**
             * 1、表示用户名没有被占用，则允许注册
             * 2、调用持久层 Integer insert(User user)方法，执行注册并获取返回值(受影响的行数)
             * 3、判断受影响的行数是否不为 1
             * 4、是：插入数据时出现某种错误，则抛出 InsertException 异常
             */
            //执行注册业务功能的实现（rows==1）
            Integer rows = userMapper.insert(user);
            if (rows != 1){
                throw new InsertException("在用户注册过程中产生了未知的异常");
            }
        }
}
```

2.完成后在 src/test/java 下创建 com.fx.store.service.UserServiceTests 测试类，编写并执行用户注册业务层的单元测试。

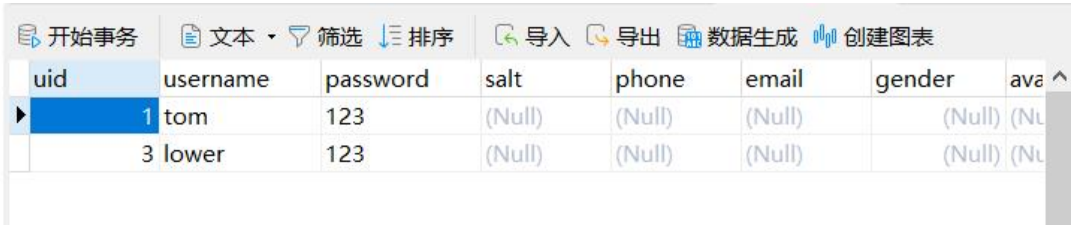
```
package com.fx.store.service;

import com.fx.store.entity.User;
import com.fx.store.mapper.UserMapper;
import com.fx.store.service.ex.ServiceException;
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 UserServiceTests {
// idea 有检测的功能，接口是不能直接创建 Bean
    @Autowired
    private IUserService iUserService;
```

```
    /**
     * 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 reg(){
        try {
            User user = new User();
            user.setUsername("lower");
            user.setPassword("123");
            iUserService.reg(user);
            System.out.println("ok");
        }catch (ServiceException e){
            //获取类的对象，再获取类的名称
            System.out.println("注册失败！"+e.getClass().getSimpleName());
            //获取类的具体描述信息
            System.out.println(e.getMessage());
        }

    }

}
```

运行单元测试方法，出现以下：



并再数据库中看一下插入记录：



如果这样插入数据，则会出现密码明文。

4.5 密码加密介绍

密码加密可以有效的防止数据泄密后带来的账号安全问题。通常，程序员不需要考虑加密过程中使用的算法，因为已经存在非常多成熟的加密算法可以直接使用。但是所有的加密算法都不适用于对密码进行加密，因为加密算法都是可以进行逆向运算的。即：如果能够获取加密过程中所有的参数，就可以根据密文得到原文。

对密码进行加密时，需使用消息摘要算法。消息摘要算法的特点是：

1.原文相同时，使用相同的摘要算法得到的摘要数据一定相同；

2.使用相同的摘要算法进行运算，无论原文的长度是多少，得到的摘要数据长度是固定的；

3.如果摘要数据相同，则原文几乎相同，但也可能不同，可能性极低。

不同的原文，在一定的概率上能够得到相同的摘要数据，发生这种现象时称为碰撞。

以 MD5 算法为例，运算得到的结果是 128 位的二进制数。在密码的应用领域中，通常会限制密码长度的最小值和最大值，可是密码的种类是有限的，发生碰撞在概率上可以认为是不存在的。

常见的摘要算法有 SHA(Secure Hash Argorithm)家族和 MD(Message Digest)系列的算法。

关于 MD5 算法的破解主要来自两方面。一个是王小云教授的破解，学术上的破解其实是研究消息摘要算法的碰撞，也就是更快的找到两个不同的原文却对应相同的摘要，并不是假想中的“根据密文逆向运算得到原文”。另一个是所谓的“在线破解”，是使用数据库记录大量的原文与摘要的对应关系，当尝试“破解”时本质上是查询这个数据库，根据摘要查询原文。

为进一步保障密码安全，需满足以下加密规则：

1.要求用户使用安全强度更高的原始密码；

2.加盐；

3.多重加密；

4.综合以上所有应用方式。

在 IUserServiceImpl 实现类的 reg 方法中补全加密后的密码和盐值数据，并创建一个**执行密码加密**的方法 getMd5Password（String password,String salt）。

```
package com.fx.store.service.impl;

import com.fx.store.entity.User;
import com.fx.store.mapper.UserMapper;
import com.fx.store.service.IUserService;
import com.fx.store.service.ex.InsertException;
import com.fx.store.service.ex.UsernameDuplicateException;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.util.DigestUtils;

import java.util.Date;
import java.util.UUID;

/**
 * 用户模块业务层的实现类
 */
@Service//Service 注解，将当前类的对象交给 spring 来管理，自动创建类的对象以及对象的维护
public class IUserServiceImpl implements IUserService {
    @Autowired
```

```
    private UserMapper userMapper;
    @Override
    public void reg(User user) {
        /**
         * 1、根据参数 user 对象获取注册的用户名
         * 2、调用持久层的 User findByUsername(String username)方法，根据用户名查询用户数据
         * 3、判断查询结果是否不为 null
         * 4、是：表示用户名已被占用，则抛出 UsernameDuplicateException 异常
         */
        //通过 user 参数来获取传递过来的 username
        String username = user.getUsername();
        //调用 findByUsername（username）判断用户是否被注册过
        User result = userMapper.findByUserName(username);
        if (result != null){
            //抛出异常
            throw new UsernameDuplicateException("用户名被占用");
        }
        //补全数据：加密后的密码
        /**
         * 密码加密处理实现：md5 加密形式
         * （串 + password + 串）----md5 算法进行加密，连续加载三次，这里的串就是盐值
         * （盐值 + password + 盐值）----盐值就是一个随机的字符串
         */
        String userPassword = user.getPassword();
        //获取盐值（随机生成一个盐值）
        String salt = UUID.randomUUID().toString().toUpperCase();
        //将密码和盐值作为一个整体进行加密处理
        String md5Password = getMd5Password(userPassword, salt);
        //补全数据：加密后的密码
        user.setPassword(md5Password);
        //补全数据：盐值
        user.setSalt(salt);
        // 补全数据：isDelete(0)
        user.setIsDelete(0);
        // 补全数据：4 项日志属性
        user.setCreatedUser(user.getUsername());
        user.setModifiedUser(user.getUsername());
        Date date = new Date();
        user.setCreatedTime(date);
        user.setModifiedTime(date);

        /**
```

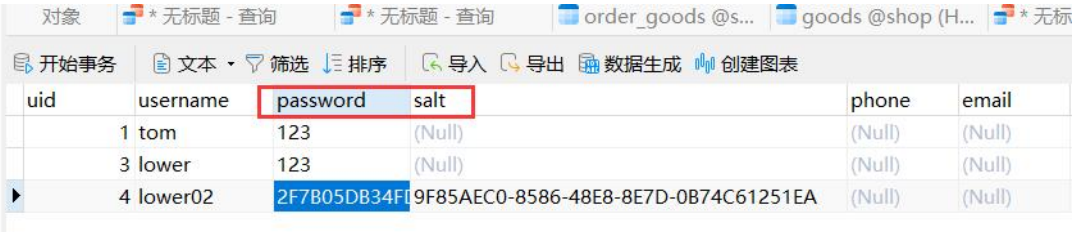
```
            * 1、表示用户名没有被占用，则允许注册
            * 2、调用持久层 Integer insert(User user)方法，执行注册并获取返回值(受影响
的行数)
            * 3、判断受影响的行数是否不为 1
            * 4、是：插入数据时出现某种错误，则抛出 InsertException 异常
            */
           //执行注册业务功能的实现（rows==1）
           Integer rows = userMapper.insert(user);
           if (rows != 1){
               throw new InsertException("在用户注册过程中产生了未知的异常");
           }
       }
      /**
     * 执行密码加密
     * @param password 原始密码
     * @param salt 盐值
     * @return 加密后的密文
     */
       private String getMd5Password(String password,String salt){
           for (int i=0;i<3;i++){
               //md5 加密方法的调用（进行三次调用）
               password =
DigestUtils.md5DigestAsHex((salt+password+salt).getBytes()).toUpperCase();
           }
           //返回加密后的密码
           return password;
       }
}
```

重新运行测试类 UserServiceTests 中的测试方法 reg（）。

注意：根据用户名的唯一性一定要换一个用户名，否则会报错。

```
user.setUsername("lower02");
```

运行结束后查询数据库结果如下：



uid	username	password	salt	phone	email
1	tom	123	(Null)	(Null)	(Null)
3	lower	123	(Null)	(Null)	(Null)
4	lower02	2F7B05DB34F[	9F85AEC0-8586-48E8-8E7D-0B74C61251EA	(Null)	(Null)