5. Eclipse与异常处理

5.1. 异常处理

- try与catch关键字:

```
// 在程序中出现异常，就必须进行处理，处理格式如下:
try (...) {// JDK1.7以后直接释放资源
    // 可能发生异常的代码段
} catch (异常类型 对象) {
    // 异常的处理操作
} catch (异常类型 对象) {
    // 异常的处理操作
}...
finally {
    // 异常的统一出口
}
```

- 异常处理过程分析:
  - 一旦产生异常，则系统会自动产生一个异常类的实例化对象
  - 此时如果存在了try语句，则会自动找到匹配的catch语句执行，如果没有异常处理，则程序退出，并由系统报告错误
  - 所有的catch根据方法的参数匹配异常类的实例化对象，如果匹配成功，则表示由此catch进行处理
- finally关键字:
  - 在进行异常的处理之后，在异常的处理格式中还有一个finally语句，那么此语句将作为异常的统一出口，不管是否产生了异常，最终都要执行此段代码
- 什么是异常?
  - 异常是阻止当前方法或作用域继续执行的问题，在程序中导致程序中断运行的一些指令
- throw和throws关键字:
  - throws关键字主要在方法的声明上使用，表示方法中不处理异常，而交给调用处理。实际上对于Java程序来讲，如果没有加入任何的异常处理，默认由JVM进行异常的处理操作

    - throws关键字表示在程序中手动抛出一个异常，因此从异常处理机制来看，所有的异常一旦产生之后，实际上抛出的就是一个异常类的实例化对象，那么此对象也可以由throw直接抛出
- 异常处理的语法法则：
    - try语句不能单独存在，可以和catch、finally组成try..catch.finally、try...catch、try...finally三种结构，catch语句可以有一个或多个，finally语句最多一个，try、catch、finally这三个关键字均不能单独使用
    - try、catch、finally三个代码块中变量的作用域分别独立而不能相互访问
    - 多个catch块时候，Java虚拟机会匹配其中一个异常类或子类，就执行这个catch块，而不会执行别的catch块

5.2. 自定义异常

- 自定义异常类：CustomException

```
/**
 * 自定义异常通常都是通过继承一个异常类来实现的
 * 1.Throwable
 * 2.Exception
 * 3.RuntimeException
 * 自定义异常通常的实现是重写父类的构造方法
 * 异常对象本身是没有实际功能，只是一个有意义的标识
 */
public class CustomException extends Exception {
    public CustomException() {
        super();
    }

    public CustomException(String message) {
        super(message);
    }
}
```

- 用户类：User

```
/**
 * @description User类
 */
public class User {
    private String userName;
    private String password;
    private int age;
    private String sex;

    public User() {
    }

    public User(String userName, String password, int age, String sex)
{
```

```
        super();
        this.userName = userName;
        this.password = password;
        this.age = age;
        this.sex = sex;
    }

    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 int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public String getSex() {
        return sex;
    }

    public void setSex(String sex) {
        this.sex = sex;
    }

    @Override
    public String toString() {
        return "User [userName=" + userName + ", password=" + password
+ ", age=" + age + ", sex=" + sex + "]";
    }

}
```

- 用户服务类：UserService

```
/**
 * @description UserService类
 */
public class UserService {
    public User login(String userName, String password) throws
CustomException {
        if (!"admin".equals(userName)) {
            throw new CustomException("用户名错误");
```

```
        }
        if (!"123456".equals(password)) {
            throw new CustomException("密码错误");
        }

        User user = new User("admin", "123456", 18, "男");
        return user;
    }
}
```

- 登录测试类（主函数）：LoginDemo

```
import java.util.Scanner;

/**
 * @description LoginDemo类
 */
public class LoginDemo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("请输入用户名: ");
        String name = scanner.nextLine();
        System.out.println("请输入密码: ");
        String password = scanner.nextLine();

        UserService userService = new UserService();
        try {
            User user = userService.login(name, password);
            System.out.println("登录成功");
            System.out.println(user);
        } catch (CustomException e) {
            e.printStackTrace();
        }

        scanner.close();
    }
}
```

5.3. 受检与非受检异常

- 受检异常：Exception
  - 定义方法时必须声明所有可能会抛出的exception；在调用这个方法时，必须捕获它的checked exception，不然就得把它的exception传递下去；exception是从java.lang.Exception类衍生出来的。例如：IOException，SQLException
- 非受检异常：RuntimeException
  - 在定义方法时不需要声明会抛出的runtime exception；在调用这个方法时不需要捕获这个runtime exception；runtime exception是从java.lang.RuntimeException或java.lang.Error类衍生出来的。例如：NullPointerException，IndexOutOfBoundsException

- 注意：assert关键字：表示断言
  - 当程序执行到某个固定的位置时，程序中的某个变量的取值肯定是预期的结果，那么这种操作可以使用断言完成

```
// 格式
public class AssertDemo {
    public static void main(String[] args) {
        int result = add(1, 10);
        assert result == 10 : "结果不正确";
    }

    private static int add(int a, int b) {
        return a + b;
    }
}

// 在eclipse中运行时需要对运行进行简单配置，配置Arguments中的VM arguments，配置为-ea
```

5.4. 注意：

- Throwable是异常的基类，分为Error和Exception，在编程中我们要关注Exception
- Exception分为编译期异常（受检）和运行期异常（非受检）
- 异常会导致程序中断，无法继续执行
- 在开发中，我们需要把可能出现异常的代码使用try语句块包裹起来
- 处理异常可以让程序保持运行状态
- catch可以有多个，顺序为从子类到父类，大的放后面，小的放前面