异 常

> Author：zhangzhang
>
> Version：1.0.0

一、异常

1.1 概念

> 异常：程序在运行过程中出现的特殊情况。

1.2 异常的必要性

> 任何程序都可能存在大量的未知问题、错误。
>
> 如果不对这些问题进行正确处理，则可能导致程序的中断，造成不必要的损失。

二、异常分类

> Throwable：可抛出的，一切错误或异常的父类，位于java.lang包中。

2.1 错误

> - Error： JVM、硬件、执行逻辑错误，不能手动处理。
> - 常见错误： StackOverflowError、 OutOfMemoryError等。

2.2 异常

> - Exception：程序在运行和配置中产生的问题，可处理。
>   - RuntimeException：运行时异常，可处理，可不处理。
>   - CheckedException：检查时异常，必须处理。

> 常见运行时异常：

异常	描述
NullPointerException	空指针异常
ArrayIndexOutOfBoundsException	数组越界异常
ClassCastException	类型转换异常
NumberFormatException	数字格式化异常
ArithmeticException	算术异常

> 案例演示：

```
public class Demo1 {
  public static void main(String[] args) {
    //常见运行时异常
    //1NullPointerException
    String name=null;
    System.out.println(name.equals("zhangsan"));
    //2ArrayIndexOutOfBoundsException
    int[] arr= {10,30,50};
    System.out.println(arr[3]);
    //3ClassCastException
      Object str="hello";
    Integer i=(Integer)str;
    //4NumberFormatException
    int n=Integer.parseInt("100a");
    System.out.println(n);
    //5ArithmeticExceptioin
    int n=10/0;
    System.out.println(n);


    try {
      FileInputStream fis=new FileInputStream("d:\\hell.txt");
    } catch (FileNotFoundException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
  }
}
```

三、异常产生和传递

3.1 异常产生

> - 自动抛出异常：当程序在运行时遇到不符合规范的代码或结果时，会产生异常。
> - 手动抛出异常：语法：throw new 异常类型(“实际参数”)。
> - 产生异常结果：相当于遇到 return语句，导致程序因异常而终止。

3.2 异常传递

> - 异常的传递：
>   - 按照方法的调用链反向传递，如始终没有处理异常，最终会由JVM进行默认异常处理（打印堆栈跟踪信息）。
> - 受查异常：throws 声明异常，修饰在方法参数列表后端。
> - 运行时异常：因可处理可不处理，无需声明异常。

> 案例演示：异常的产生、传递。

```
/**
 * 演示异常的产生和传递
 * 要求：输入两个数字实现两个数字相除
 */
public class TestException1 {
  public static void main(String[] args) {
    operation();
  }
  public static void operation() {
    System.out.println("---opration-----");
    divide();
  }
  public static void divide() {
    Scanner input=new Scanner(System.in);
    System.out.println("请输入第一个数字");
    int num1=input.nextInt();//出现异常，没有处理，程序中断
    System.out.println("请输入第二个数字");
    int num2=input.nextInt();
    int result=num1/num2;//出现异常没有处理，所以程序中断
    System.out.println("结果:"+result);
    System.out.println("程序执行完毕了...");
  }
}
```

四、异常处理【 重点 】

> Java的异常处理是通过5个关键字来实现的：
>
> - try：执行可能产生异常的代码 。
> - catch：捕获异常 ，并处理。
> - finally：无论是否发生异常，代码总能执行。
> - throw： 手动抛出异常 。

- throws：声明方法可能要抛出的各种异常。

4.1 try...catch...

语法：

try {

  //可能出现异常的代码

} catch(Exception e) {

  //异常处理的相关代码，如：getMessage()、printStackTrace()

}

```
public class TestException2 {
  public static void main(String[] args) {
    Scanner input=new Scanner(System.in);
    int result=0;
    try {
      System.out.println("请输入第一个数字");
      int num1=input.nextInt();//InputMismatchException
      System.out.println("请输入第二个数字");
      int num2=input.nextInt();
      result=num1/num2;//发生异常// ArethmicException
    }catch (Exception e) {//捕获 Exception：是所有异常的父类
      //处理
      //e.printStackTrace();
      System.out.println(e.getMessage());
    }
    System.out.println("结果是:"+result);
    System.out.println("程序结束了...");

  }
}
```

注：1、正常请求 2、出现异常并处理 3、异常类型不匹配。

4.2 try...catch...finally...

语法：

try {
  //可能出现异常的代码
} catch(Exception e) {

  //异常处理的相关代码，如：getMessage()、printStackTrace()

} finally{
  //是否发生异常都会执行，可以释放资源等。
}

```
public class TestException3 {
  public static void main(String[] args) {
    Scanner input=new Scanner(System.in);
    int result=0;
    try {
      System.out.println("请输入第一个数字");
      int num1=input.nextInt();//InputMismatchException
      System.out.println("请输入第二个数字");
      int num2=input.nextInt();
      result=num1/num2;//发生异常// ArethmicException
      //手动退出JVM
      //System.exit(0);
    }catch (Exception e) {//捕获 Exception：是所有异常的父类
      //处理
      //e.printStackTrace();
      System.out.println(e.getMessage());
    }finally {
      System.out.println("释放资源...");
    }
    System.out.println("结果是:"+result);
    System.out.println("程序结束了...");
  }
}
```

注：1、finally块是否发生异常都执行，释放资源等 2、finally块不执行的唯一情况,退出java虚拟机。

4.3 多重catch

语法：

try{

  //可能出现异常的代码。

}catch(异常类型1){

```
  //满足异常类型1执行的相关代码。
}catch(异常类型2){
  //满足异常类型2执行的相关代码。
}catch(异常类型3){
  //满足异常类型3执行的相关代码
}
```

```java
public class TestException4 {
  public static void main(String[] args) {
    Scanner input=new Scanner(System.in);
    int result=0;
    try {
//      String string=null;
//      System.out.println(string.equals("hello"));
      System.out.println("请输入第一个数字");
      int num1=input.nextInt();//InputMismatchException
      System.out.println("请输入第二个数字");
      int num2=input.nextInt();
      result=num1/num2;//发生异常// ArethmicException
    }catch (ArithmeticException e) {//捕获 Exception：是所有异常的父类
      System.out.println("算术异常");
    }catch (InputMismatchException e) {
      System.out.println("输入不匹配异常");
    }catch (Exception e) {
      System.out.println("未知异常");
    }
    System.out.println("结果是:"+result);
    System.out.println("程序结束了...");
  }
}
```

注意：

- **子类异常在前，父类异常在后。**
- **发生异常时按顺序逐个匹配。**
- **只执行第一个与异常类型匹配的catch语句。**
- **finally根据需要可写或不写。**

4.4 try...finally...

- try...finally...不能捕获异常，仅仅用来当发生异常时，用来释放资源。
- 一般用在底层代码，只释放资源不做异常处理，把异常向上抛出。

```
语法：

try{
  //可能出现异常的代码
}finally{
  //是否发生异常都会执行，可以释放资源等
}
```

```java
public class TestException5 {
  public static void main(String[] args) {//JVM
    try {
      divide();
    }catch (Exception e) {
      System.out.println("出现异常:"+e.getMessage());
    }
  }
  public static void divide() {
    Scanner input=new Scanner(System.in);
    int result=0;
    try {
      System.out.println("请输入第一个数字");
      int num1=input.nextInt();//InputMismatchException
      System.out.println("请输入第二个数字");
      int num2=input.nextInt();
      result=num1/num2;//发生异常// ArethmicException
    }finally {
      System.out.println("释放资源");
    }
    System.out.println("结果是:"+result);
    System.out.println("程序结束了...");
  }
}
```

4.5 小结

- try{ } catch{ }
- try{ } catch{ } catch{ }

- try{ } catch{ } finally{ }
- try{ } catch{ } catch{ } finally{ }
- try{ } finally{ }

注：多重catch，遵循从子(小)到父(大)的顺序，父类异常在最后。

五、声明、抛出异常

5.1 声明异常

> 如果在一个方法体中抛出了异常，如何通知调用者?
>
> - throws关键字：声明异常

```
public class TestException6 {
  public static void main(String[] args){//JVM
    try {
      divide();
    } catch (Exception e) {
      // TODO Auto-generated catch block
      //e.printStackTrace();
      System.out.println(e.getMessage());
    }

  }
  public static void divide() throws Exception {
    Scanner input=new Scanner(System.in);
    System.out.println("请输入第一个数字");
    int num1=input.nextInt();
    System.out.println("请输入第二个数字");
    int num2=input.nextInt();
    int result=num1/num2;
    System.out.println("结果:"+result);
  }
}
```

5.2 抛出异常

> 除了系统自动抛出异常外，有些问题需要程序员自行抛出异常。
>
> - throw关键字：抛出异常

```
public class Person {
  private String name;
  private String sex;
  private int age;
  public Person() {
    // TODO Auto-generated constructor stub
  }
  public Person(String name, String sex, int age) {
    super();
    this.name = name;
    this.sex = sex;
    this.age = age;
  }
  public String getName() {
    return name;
  }
  public void setName(String name) {
    this.name = name;
  }
  public String getSex() {
    return sex;
  }
  public void setSex(String sex) {
    if(sex.equals("男")||sex.equals("女")) {
      this.sex = sex;
    }else {
      throw new RuntimeException("性别不符合要求");
    }
  }
  public int getAge() {
    return age;
  }
  public void setAge(int age) {
    if(age>0&&age<=120) {
      this.age = age;
    }else {
      //抛出异常
      throw new RuntimeException("年龄不符合要求");
    }
  }
  @Override
  public String toString() {
```

```
    return "Person [name=" + name + ", sex=" + sex + ", age=" + age + "]";
  }

}
```

六、自定义异常

6.1 编写自定义异常

- 需继承Exception或Exception的子类，代表特定问题。
- 异常类型名称望文生义，可在发生特定问题时抛出对应的异常。

常用构造方法：

- 无参数构造方法。
- String message参数的构造方法。

```
public class AgeException extends RuntimeException{

  public AgeException() {
    super();
    // TODO Auto-generated constructor stub
  }

  public AgeException(String message, Throwable cause, boolean enableSuppression, boolean writableStackTrace) {
    super(message, cause, enableSuppression, writableStackTrace);
    // TODO Auto-generated constructor stub
  }

  public AgeException(String message, Throwable cause) {
    super(message, cause);
    // TODO Auto-generated constructor stub
  }

  public AgeException(String message) {
    super(message);
    // TODO Auto-generated constructor stub
  }

  public AgeException(Throwable cause) {
    super(cause);
    // TODO Auto-generated constructor stub
  }

}
```

6.2 异常中方法覆盖

带有异常声明的方法重写：

- 方法名、参数列表、返回值类型必须和父类相同。
- 子类的访问修饰符合父类相同或是比父类更宽。
- 子类中的方法，不能抛出比父类更多、更宽的检查时异常。

```
public class Animal {
  public void eat(){
    System.out.println("父类吃方法.........");
  }
}
```

```
public class Dog extends Animal{
  @Override
  public void eat() throw Exception{
        //出现错误，父类没有声明异常，子类不能声明异常
    System.out.println("子类的吃的方法.........");
  }
}
```