**Java中的Exception**

# **1异常相关概述**

## 什么是异常

* 异常就是Java程序在运行过程中出现的错误。
* 前面接触过的**空指针，数组越界，类型转换错误异常**等

## Throwable

* Throwable 类是 Java 语言中所有错误或异常的超类。
* 只有当对象是此类（或其子类之一）的实例时，才能通过 JVM 或者 throw 语句抛出。

## 1.3异常的继承体系

|  |  |
| --- | --- |
| -Throwable  -Error  -Exception  -RuntimeException |  |

## JVM默认是如何处理异常的?

* jvm有一个默认的异常处理机制,就将该异常的名称、异常的信息、异常出现的位置打印在了控制台上,同时程序停止运行。

|  |
| --- |
|  |
|  |

## Java处理异常的两种方式

* Java虚拟机处理
* 自己处理

## 为什么会有异常

因为你不知道未来会怎么样，需要做个准备（写代码也是，你需要对一些未知的东西做下处理），比如你去旅游，不知道会下雨还是出大太阳，你需要准备雨伞和帽子.如果下雨，我有雨伞可用，如果出大太阳，可以用帽子，也有可能天气刚刚好，雨伞和帽子都用不着，这叫白带。

## 回顾几个常见异常

|  |
| --- |
|  |

# 2 try-catch

自己处理异常的两种方试

* try…catch…finally
* throws

## 2.1 try-catch异常处理方式

|  |
| --- |
|  |
|  |

## 2.2 try-catch-catch 多个catch处理方式

|  |
| --- |
| try {  int[] arr = {1,2,3};  System.out.println(arr[4]);//ArrayIndexOutOfBoundsException数组越界异常  int a = 10 / 0;//ArithmeticException：算术异常  System.out.println(a);    int[] arr1 = null;  System.out.println(arr1[0]);//NullPointerException空指针  } catch (ArrayIndexOutOfBoundsException e) {  System.out.println("数组越界异常...");  } catch (ArithmeticException e){  System.out.println("算术异常...");  } catch(NullPointerException e){  System.out.println("空指针异常...");  } |

## 2.3 try-catch-catch 多个异常-另一种写法

* 把多个异常放在一个catch中，用|或符号连接

|  |
| --- |
| try {  int[] arr = {1,2,3};  System.out.println(arr[4]);//ArrayIndexOutOfBoundsException数组越界异常  int a = 10 / 0;//ArithmeticException：算术异常  System.out.println(a);    //catch (ArrayIndexOutOfBoundsException | ArithmeticException | NullPointerException e)  } catch (ArrayIndexOutOfBoundsException | ArithmeticException e) {  System.out.println(e.getClass());  System.out.println("数组越界异常或者算法异常...");  } |

## 2.4 多个异常处理的一个注意事项

* catch中父类的Exception只能放在最后面
* catch一个原则，先由子类处理异常，子类不能处理异常，再由父类处理

|  |
| --- |
| try {  //1.有可能出现数组越界异常  int[] arr = {1,2,3};  System.out.println(arr[1]);    //2.算术异常  int a = 10 / 2;    //3.空指针异常  int[] arr1 = null;  //NullPointerException np;  System.out.println(arr1[0]);  }catch (ArrayIndexOutOfBoundsException e) {  System.out.println("数组越界异常");  } catch (ArithmeticException e) {  System.out.println("算术异常");  **} catch (Exception e) {**  **System.out.println("其它异常");**  **}** |

## 2.5 "编译时异常"和"运行时异常

* Java中的异常被分为两大类："编译时异常"和"运行时异常"。
* **编译时异常**和**运行时异常**的区别

>所有的RuntimeException类及其子类被称为运行时异常

>其他的异常就是编译时异常

### 编译时异常

* Java程序必须显示处理，否则程序就会发生错误，无法通过编译

|  |
| --- |
|  |
|  |

### 运行时异常

* 无需显示处理，也可以像编译时异常一样处理
* 比如算术异常，数组越界,空指针都是运行时异常

|  |
| --- |
|  |

## 2.6 Throwable有几个常用的方法

* Exception是一个继承Throwable的子类
* getMessage():获取异常信息，返回字符串。
* toString():获取异常类名和异常信息，返回字符串。
* printStackTrace():打印异常类名和异常信息，以及异常出现在程序中的位置

|  |
| --- |
|  |

# 3 throws

## 3.1 throws方式处理异常

* 实现方式：

1. 定义方法时，在方法名后面添加throws 异常类名，相当于问题暴露出来让调用者去处理
2. 方法内部要写个throw 异常对象，来抛出一个异常对象

|  |
| --- |
| Person.java |
| class Person {  private int age;  public void setAge(int age)**throws Exception** {  //年龄要1~150岁内  if(age >= 1 && age <=150){  this.age = age;  }else{  //System.out.println("你是来自火星");  **throw new Exception("你是来自火星");//告诉外界异常的类型**  }  }  public void say(){  System.out.println("我今年"+ age);  }  } |
| main方法 |
|  |

## 3.2 throws和throw的区别

### throws

用在方法声明后面，跟的是异常类名

可以跟多个异常类名，用逗号隔开

它表示抛出异常，由该方法的调用者来处理

### throw

用在方法体内，跟的是异常对象名

只能抛出一个异常对象名，表示抛出异常

# 4 finally

## 4.1 try-catch-finally

### finally的特点

* finally的语句体一般情况下一定会执行
* 特殊情况：在执行到finally之前jvm退出了(比如System.exit(0))

### finally的作用

* 用于释放资源，在IO流操作和数据库操作中会见到

|  |
| --- |
|  |

## 4.2 面试题

### final,finally和finalize的区别

* final可以修饰类,不能被继承;修饰方法,不能被重写;修饰变量,只能赋值一次

* finally是try语句中的一个语句体,不能单独使用,用来释放资源

* finalize是一个方法,当垃圾回收器确定不存在对该对象的更多引用时，由对象的垃圾回收器调用此方法。【当对象从内存中消失会调用这个方法】

### catch中关于return的面试题

如果catch里面有return语句，请问finally的代码还会执行吗?如果会，请问是在return前还是return后?

答：会执行，finally的代码在return之前执行

|  |
| --- |
|  |

# 5 自定义异常

## 5.1案例：自定义年龄异常

* 自定义异常的步骤

|  |
| --- |
| /\*\*  \* 自定义异常的步骤：  \* 1.写一个类(AgeException)继承Exception，这种是编译时异常  \* 2.在子类异常AgeException提供一个带字符串参数的构造方法  \* public AgeException(String msg){  super(msg);  }  3.自定义异常时，也可以继承RuntimeException,这种是运行时异常  \*/ |

* 代码：

|  |
| --- |
| AgeException.class |
|  |
| Person.class |
|  |

## 5.2自定义异常注意事项

|  |
| --- |
| /\*  1.子类重写父类方法时，子类的方法必须抛出相同的父类异常  2.如果被重写的方法没有异常抛出,那么子类的方法最好不要抛出异常  3.如果子类方法内有异常发生,那么子类只能try,不能throws\*/ |
|  |

## 5.3工作中如何使用异常处理

1.原则:如果自己能处理的问题，就用try-catch, 如果自己不能解决的问题，就throws

2.try-catch和throws的区别:

后续程序需要继续运行就用try

后续程序不需要继续运行就throws

3.如果JDK没有提供对应的异常，需要自定义异常。

|  |
| --- |
|  |
|  |

## 5.4 案例：自定义一个登录异常

* 注意事项：什么时候方法后面不需要写throws？

方法内throw的异常对象如果是一个运行时异常，方法后面就不会写throws

|  |
| --- |
|  |
|  |

## 5.5 案例：自定义一个非法字符异常

|  |
| --- |
|  |
|  |