# 尚马教育 JAVA 基础课程

# 异常

文档编号：A10

创建日期： 2017-04-12

最后修改日期：2019-09-23

版 本 号：V3.0

电子版文件名：尚马教育-第一阶段-10.异常专题课程.docx

**文档修改记录：**

|  |  |  |  |
| --- | --- | --- | --- |
| 更新日期 | 更新作者 | 更新说明 | 版本号 |
| 2017-07-30 | 张元林 | 初始版本 | V1.0 |
| 2018-08-01 | 王绍成 | Java基础版本更新 | V2.0 |
| 2019-08-09 | 徐丽莎 | Java基础版本更新 | V3.0 |

**主讲人：**

**徐丽莎**

目录

[尚马教育 JAVA 基础课程 1](#_Toc23647)

[异常 1](#_Toc1374)

[1. 异常 2](#_Toc19375)

[1.1. 概念 2](#_Toc11837)

[1.1.1. 异常（Exception） 2](#_Toc14785)

[1.1.2. 错误（Error） 3](#_Toc17056)

[2. Exception 4](#_Toc5875)

[2.1. 异常体系 4](#_Toc14743)

[2.2. 常见异常 5](#_Toc28030)

[2.2.1. NullPointerException-空指针异常 6](#_Toc20031)

[2.2.2. ArithmeticException-数学异常 6](#_Toc1738)

[3. 异常处理 6](#_Toc7218)

[3.1. 标准异常处理流程 6](#_Toc28816)

[3.2. Try...catch...finally 7](#_Toc163)

[3.3. throw与throws 12](#_Toc2065)

[4. 自定义异常 14](#_Toc27973)

[5. Try...with...resources 16](#_Toc9181)

[6. 作业 17](#_Toc871)

## 异常

* Java语言的健壮性:
  + GC
  + 异常的处理

### 概念

* 说到异常，总有人想到错误。在异常机制中，异常和错误是两个概念，异常能够被处理，错误不能。本节我们先简单了解两者的区别。

#### 异常（Exception）

* 异常指的是程序运行时发生的不正常事件；异常能够被程序处理，保证程序继续运行下去；例如除数为0、文件没有找到、输入的数字格式不对……
* 下面程序因为除数为0，所以发生了异常；

|  |
| --- |
| **public** **static** **void** main(String[] args) {  System.***out***.println("演示异常发生情况：");  // 除数为0，运行的时候程序会抛出一个异常  System.***out***.println(100 / 0);  System.***out***.println("如果我被打印出来，就是异常被处理了。");  } |
| 后续学习异常处理方法后，就可以处理该异常 |

#### 错误（Error）

* 错误程序没法处理，例如内存泄漏。发生错误后，一般虚拟机会选择终止程序运行，程序员需要修改代码才能解决相关错误；
* 下面程序因为数组长度过长，内存溢出

|  |
| --- |
| **public** **static** **void** main(String[] args) {  **int**[] array = **new** **int**[1024 \* 1024 \* 1024];  } |
| 错误没法处理，只能乖乖地把长度改小吧~~~ |

|  |
| --- |
|  |

## Exception

### 异常体系

* API中标准异常的顶级父类是Throwable类；
* Throwable类有两个子类：Exception和Error；
* 所有异常都是Exception类的直接或间接子类；所有错误都是Error的直接或间接子类；

|  |
| --- |
|  |

* Exception有很多子类；这些子类又可以分为两大类；
* 即**运行时异常和非运行时异常(检测到异常)**;
* RuntimeException的子类都是运行时异常，其他的都是非运行时异常；

|  |
| --- |
|  |

* 运行时异常：也称为非检测异常（unchecked Exception）， 这些异常在编译期不检测，程序中可以选择处理，也可以不处理。如果不处理运行时会中断，但是编译没问题；运行时异常经常在编程时发生，了解每种异常的概念有助于高效调试程序；
* 非运行时异常：也称为检测异常（checked Exception）, 是必须进行处理的异常，如果不处理，将发生编译期错误；

|  |
| --- |
|  |

### 常见异常

|  |
| --- |
|  |

#### NullPointerException-空指针异常

* 发生前提：当对一个空对象，即没有初始化，依然为null的对象调用属性或方法时；

|  |
| --- |
|  |

#### ArithmeticException-数学异常

* 发生前提：整数除以0时发生

|  |
| --- |
|  |

## 异常处理

* 上述内容中我们提到，异常可以被处理，处理后程序可以继续运行下去。
* 本节就开始学习Java中异常处理的流程和语句。

### 标准异常处理流程

* Java语言中异常处理主要使用到try/catch/finally三种语句，

|  |
| --- |
|  |

### Try...catch...finally

#### try块

* 一般用来监视有可能产生异常的代码部分；
* 把所有可能抛出异常的，或者肯定抛出异常的代码都写到try代码块中；

#### catch块

* catch语句紧随try语句后，用来捕获异常并进行处理
* 当try块中中代码抛出了异常对象后，异常处理机制就将这个对象的类型与try后的catch语句中的异常类型进行匹配，如果类型相同，或者抛出的是捕获的子类，就称为匹配成功，那么异常就被捕获，就运行catch块中的语句；否则，称为异常没有被捕获，程序将中断；

#### 语法

|  |
| --- |
| **try**{  //可能抛出异常的代码块;  }**catch**(异常类型 变量名){  // 处理异常的代码;  } |

#### 案例

* 抛出异常并处理成功；

|  |
| --- |
| **public** **static** **void** main(String[] args) {  **try** {  **int** x = 100;  **int** y = 0;  //1.抛出ArithmeticException异常，try中之后代码不运行  System.***out***.println("x/y=" + x / y);  System.***out***.println("x/y计算结束");  } **catch** (ArithmeticException e) {//2.异常处理机制将ArithmeticException与catch语句的异常类型匹配  System.***out***.println("发生了数学异常，注意除数不能为0.");//3.匹配成功， 运行catch代码块，异常被处理；  }  System.***out***.println("main方法运行结束");//4.程序继续运行。  } |

* 抛出异常，但是没有被处理

|  |
| --- |
| **public** **static** **void** main(String[] args) {  **try** {  **int** x = 100;  **int** y = 0;  System.***out***.println("x/y=" + x / y);  //1.抛出ArithmeticException异常，try中之后代码不运行  System.***out***.println("x/y计算结束");  } **catch** (NullPointerException e) {//2.异常处理机制将ArithmeticException与catch语句的异常类型匹配  System.***out***.println("发生了异常");//3.匹配失败， 不运行catch代码块，异常没有被处理；  }  System.***out***.println("main方法运行结束");//4.程序中断运行，不会打印“main方法运行结束”  } |

* 没有抛出异常

|  |
| --- |
| **public** **static** **void** main(String[] args) {  **try** {  //没有抛出ArithmeticException异常  //运行try块所有的逻辑  **int** x = 100;  **int** y = 10;  System.***out***.println("x/y=" + x / y);  System.***out***.println("x/y计算结束");  } **catch** (ArithmeticException e) {  //跳过catch代码块  System.***out***.println("发生了数学异常，注意除数不能为0.");  }  System.***out***.println("main方法运行结束");// 运行结束  } |

#### Catch块逻辑

* 捕获异常是为了处理它，不要捕获了却什么都不处理而抛弃之，如果不想处理它， 请将该异常抛给它的调用者。最外层的业务使用者，必须处理异常，将其转化为用户可以理解的内容。
* catch语句里都写什么代码？
  + 可以写任意需要对异常进行处理的代码；
  + 可以调用异常对象的方法，例如printStackTrace，查看异常发生的栈轨迹
* 开发中常用的处理方式:
  + 异常通常与日志进行结合，将异常信息写入到不同级别的日志文件中，方便查看
  + 异常信息的传递

#### 多个catch

* 如果try块中有多行代码，有可能抛出多种类型异常，那么可以使用多个catch语句；
* 注意：catch语句的异常类型必须从子类到父类的顺序，否则编译错误；(也就是说级别越高，就放最下面的catch)

|  |
| --- |
| **public** **static** **void** main(String[] args) {  // 当y=0时，发生数学异常，运行catch(ArithmeticException e)代码块；  // 当y不等于0时，发生空指针异常，运行catch(NullPointerException e)代码块  //即使再有多个catch ，也是有且只允许一个  **try** {  **int** x = 100;  **int** y = 0;  String s = **null**;  System.***out***.println("x/y=" + x / y);  System.***out***.println("x/y计算结束");  System.***out***.println("字符串长度" + s.length());  } **catch** (ArithmeticException e) {  System.***out***.println("发生了数学异常，注意除数不能为0.");  } **catch** (NullPointerException e) {  System.***out***.println("发生了空指针异常");  } **catch** (Exception e) {  //从来不会运行这个catch块，因为没有其他类型异常；  //可见异常只要被成功捕获一次，就被处理了，不会再继续抛出了。  System.***out***.println("发生了其他异常");  }  System.***out***.println("main方法运行结束");  } |

* 也可以改成:

|  |
| --- |
| **public** **static** **void** main(String[] args) {  // 当y=0时，发生数学异常，运行catch(ArithmeticException e)代码块；  // 当y不等于0时，发生空指针异常，运行catch(NullPointerException e)代码块  //即使再有多个catch ，也是有且只允许一个  **try** {  **int** x = 100;  **int** y = 0;  String s = **null**;  System.***out***.println("x/y=" + x / y);  System.***out***.println("x/y计算结束");  System.***out***.println("字符串长度" + s.length());  } **catch** (ArithmeticException | NullPointerException e) {  System.***out***.println("发生了数学异常，注意除数不能为0.");  } **catch** (Exception e) {  //从来不会运行这个catch块，因为没有其他类型异常；  //可见异常只要被成功捕获一次，就被处理了，不会再继续抛出了。  System.***out***.println("发生了其他异常");  }  System.***out***.println("main方法运行结束");  } |

#### finally块

* 我们可以发现程序中有这样几种情况：
  + 没抛出异常；
  + 抛出异常并被处理了；
  + 抛出异常没有被处理；
* 如果希望不管什么情况，有一些代码都必须被执行，那么就可以把这些代码写到finally块中；
* 因此，在程序中，不管是否有异常，finally代码块的逻辑一定会被执行。
* Finally一般都是用来处理一个资源释放或者关闭的问题，

|  |
| --- |
| **finally**{  //不管什么情况，一定被执行的代码块；  } |

#### 注意

* 前面我们学习了try/catch以及try/catch/finally组合；
* 必须有try，catch可以有1个或多个，finally最多1个，可以没有，不能有多个
* finally块前有return语句，finally依然被执行；
* finally块前有System.exit(0)语句，finally不被执行；

### throw与throws

#### Throw

* 我们一直说：抛出异常，抛出异常，到底异常怎么抛出的呢？
* 抛出异常其实就是创建了一个异常对象，然后用throw关键字交给异常处理机制去处理；
* throw关键字在方法体中使用，用法如下：

|  |
| --- |
| 语法: **throw** 异常对象; |
| 例如：  **throw** **new** Exception();  或者  **catch**(Exception e){  **throw** e;  } |

* 运行时异常是JVM自动抛出，非运行时异常需要程序员用throw关键字抛出；

|  |
| --- |
| **public** **class** Calculator {  **public** **void** div(**int** x, **int** y) {  // 当除数为0时，抛出异常  **if** (y == 0) {  **throw** **new** Exception();// 编译错误  }  System.***out***.println("x/y=" + x / y);  }  } |

* 上述代码发生编译错误：由于抛出了Exception，是非运行时异常，所以编译期检测，要求必须处理，处理的方式有两种：
  + 使用try/catch/finally进行处理；
  + 不处理，用throws声明异常；
* 案例：

|  |
| --- |
| **public** **class** ThrowDemo {  **public** **static** **void** main(String[] args) {  **try** {  **int** age = Integer.*parseInt*(args[0]);  **if** (age < 0 || age > 100) {  // 创建一个异常实例，并将其手工抛出  **throw** (**new** Exception("您输入的年龄无效。"));  }  System.***out***.println("您的年龄是：" + age + "岁。");  } **catch** (Exception e) {// 捕捉异常  // 打印出异常信息  System.***out***.println(e.getMessage());  }  }  } |

#### Throws

* throws用在方法声明处，声明该方法可能发生的异常类型；
* 一个方法如果使用了throws，那么调用该方法时，编译期会提醒必须处理这些异常，否则编译错误；

|  |
| --- |
| **public** **class** Calculator {  **public** **void** div(**int** x, **int** y) **throws** Exception {  // 当除数为0时，抛出异常  **if** (y == 0) {  **throw** **new** Exception();// 编译错误  }  System.***out***.println("x/y=" + x / y);  }  } |

* throws后可以声明多种类型，用逗号隔开即可；
* 抽象方法也可以使用throws声明该方法可能抛出的异常类型；
* 一个方法如果使用了throws，那么调用该方法时，编译期会提醒必须处理这些异常，否则编译错误；
* 依然可以用两种方法处理，可以try/catch，可以继续throws，一般选择

|  |
| --- |
| **public** **static** **void** main(String[] args) {  **try** {  **new** Calculator().div(100, 10);  } **catch** (Exception e) {  e.printStackTrace();  }  } |

## 自定义异常

* 在实际开发中，常常要对一些不正常的事件进行统一处理，如果用API中的标准异常，会和API中的一些方法抛出异常混淆，因此需要自定义业务异常；

|  |
| --- |
|  |

* 自定义异常类非常简单，只要继承API中任意一个标准异常类即可；
* 多数情况下，继承Exception类；也可以选择继承其他类型异常；
* 一般自定义异常类中不写其他方法，只重载必要的构造方法；

|  |
| --- |
| **public** **class** DataValueException **extends** Exception {  **public** DataValueException() {  **super**();  }  **public** DataValueException(String message) {  **super**(message);  }  **public** DataValueException(Throwable cause) {  **super**(cause);  }  **public** DataValueException(String message, Throwable cause) {  }  } |
| **public** **class** Employee {  **private** **double** salary;  **public** **double** getSalary() {  **return** salary;  }  **public** **void** setSalary(**double** salary) **throws** DataValueException {  **if** (salary <= 2500) {  **throw** **new** DataValueException("薪资不能低于2500元");  } **else** {  **this**.salary = salary;  }  }  } |
| **public** **static** **void** main(String[] args) {  Employee e = **new** Employee("张三", 3000);  **try** {  e.setSalary(2400);  } **catch** (DataValueException e1) {  e1.printStackTrace();  }  } |

## Try...with...resources

* Jdk1.7+,对异常提供的新的编写方式，主要适用于对IO的关闭操作，不再编写finally代码块，一种更加优雅的方式来关闭流.
* 案例:

|  |
| --- |
| **public** **static** **void** main(String[] args) {  InputStream inputStream = **null**;  **try** {  inputStream = **new** FileInputStream(**new** File("a.txt"));  inputStream.read();  } **catch** (FileNotFoundException e) {  e.printStackTrace();  } **catch** (IOException e) {  e.printStackTrace();  } **finally** {  **try** {  **if** (inputStream != **null**)  inputStream.close();  } **catch** (IOException e) {  e.printStackTrace();  }  }  } |
| **public** **static** **void** main(String[] args) {  **try** (InputStream inputStream = **new** FileInputStream(**new** File("a.txt"));) {  inputStream.read();  } **catch** (FileNotFoundException e) {  e.printStackTrace();  } **catch** (IOException e) {  e.printStackTrace();  }  } |

## 作业

* 需通过控制台接收用户输入的两个整数，然后做除法。要求用异常处理输入非数字的异常，和除数为0的异常。
* 模拟实现用户购买商品的功能，使用数组模拟商品列表，当购买的商品不存在或者商品库存为0时，抛出自定义异常。用户购买某一个商品时，对异常进行处理，并对库存进行改变。