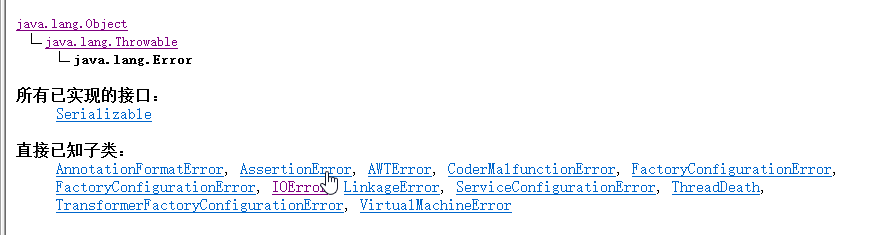
# 异常

在我们写程序的时候难免会出现错误，java中的异常机制为了提高我们程序的健壮性和容错性而存在。



# 错误

Error是throwable的子类，他是程序出现了严重的问题，这种问题程序解决不了。



# 异常

异常是程序在运行过程中可能产生问题，我们可以用程序来解决这种问题。



## 运行时异常

程序在运行的过程中可能发生的异常，在编译的时候不会报错。



RuntimeException下的所有的子类都是运行时产生的异常。我们的jvm是无法预料。

### 异常在不处理的情况下

如果一个方法发生异常，那么这个方法如果不处理这个异常，后面的代码无法执行，并且调用这个方法的方法后面的代码也不能执行。

### 运行时异常是怎么跑出来

|  |
| --- |
| **public class** ExceptionTest {  **public static void** main(String[] args) {  **int** a = 10;  **int** b = 0;  **int** result = *divide*(a, b);  System.***out***.println(**"除法的结果是："**+result);  }  **public static int** divide(**int** a, **int** b){  **int** c = 0;  *//jvm会用规则来判断这行代码是不是正确，jvm会自动的创建一个ArithmeticException对象，自动的抛向调用端* c = a/b;  System.***out***.println(**"除法正在进行"**);  **return** c;  } } |

### 如何解决异常呢

#### 自己解决问题

|  |
| --- |
| **public static int** divide(**int** a, **int** b){  **int** c = 0;   **try** {  *//可能产生问题的代码，要用try catch* c = a/b;  } **catch** (Exception e) {  System.***out***.println(**"我们采用b计划"**);  e.printStackTrace();  *//e.printStackTrace();* }  System.***out***.println(**"除法正在进行"**);  **return** c; } |

如果通过try….catch自己处理问题，那么不会终止自己的方法，同时也不会耽误调用端的方法执行。

#### 向上抛出上层处理

|  |
| --- |
| **public class** ExceptionTest1 {    **public static void** main(String[] args) {   **int** a = 10;  **int** b = 0;  **int** result = 0;  **try** {  *//被调用端向上抛出了问题* result = *divide*(a, b);  } **catch** (Exception e) {  *//e.printStackTrace();* System.***out***.println(**"b的参数非法"**);  result = *divide*(a, 2);  }  System.***out***.println(**"除法的结果是："**+result);  }    **public static int** divide(**int** a, **int** b){  **int** c = 0;  c = a/b;  System.***out***.println(**"除法正在进行"**);  **return** c;  } } |

### Finally的用法

Try…..finally

如果程序被异常终止了，可能会发生无法关闭一些占用系统资源的对象。

|  |
| --- |
| **public static int** divide(**int** a, **int** b){   **int** c = 0;  **try** {  c = a/b;  } **finally** {  System.***out***.println(**"执行程序的收尾工作"**);  }  **return** c; } |

Try….catch….finally

|  |
| --- |
| **public static int** divide(**int** a, **int** b){   **int** c = 0;   **try** {  c = a/b;  } **catch** (Exception e) {  e.printStackTrace();  } **finally** {  System.***out***.println(**"做收尾的工作"**);  }   **return** c; } |

Finally的特点：

返回值是4

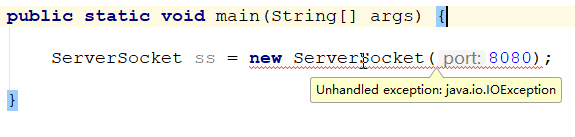
|  |
| --- |
| **public static int** method(){  **int** c = 0;   **try** {  c = c/0;  **return** c;  }**catch** (Exception e){  c = 4;  *//执行了return方法的返回值就形成了，后续的finally不管怎么修改这个变量都无济于事* **return** c;  }**finally** {  *//finally是为一个一种方式能在return后面能执行的代码块* c = 5;  } } |

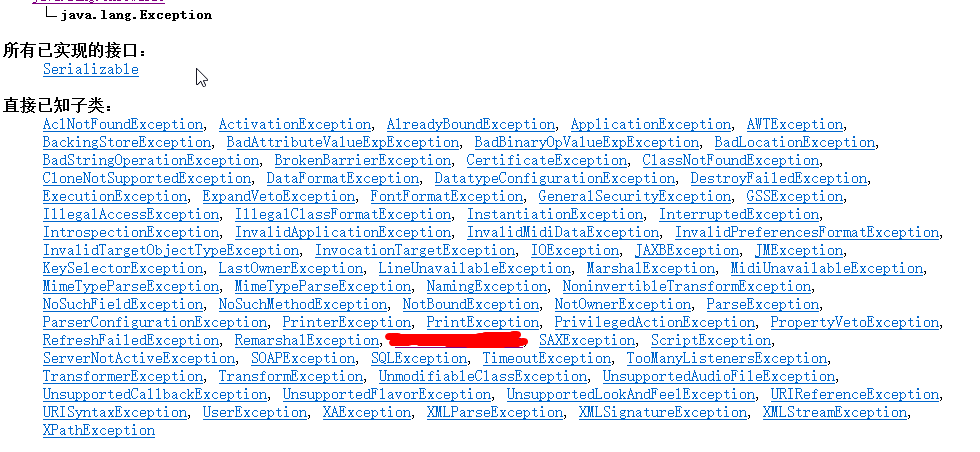
返回值是5

|  |
| --- |
| **public static int** method(){  **int** c = 0;   **try** {  c = c/0;  **return** c;  }**catch** (Exception e){  c = 4;  *//执行了return方法的返回值就形成了，后续的finally不管怎么修改这个变量都无济于事* **return** c;  }**finally** {  *//finally是为一个一种方式能在return后面能执行的代码块* c = 5;  *//finally内部的return可以修改方法的返回值* **return** c;  } } |

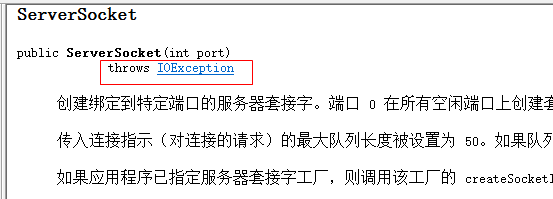
## 编译期异常

在编译的时候就会提示你处理这个异常，否则编译不会通过。





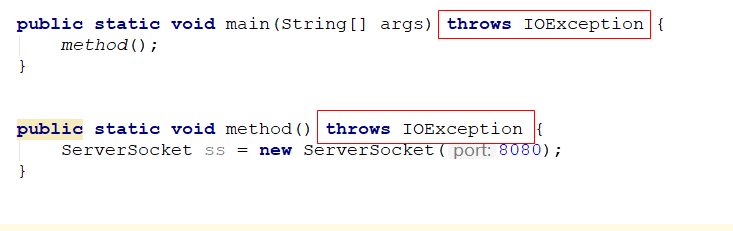
除了RuntimeException之外的都是编译期的异常。



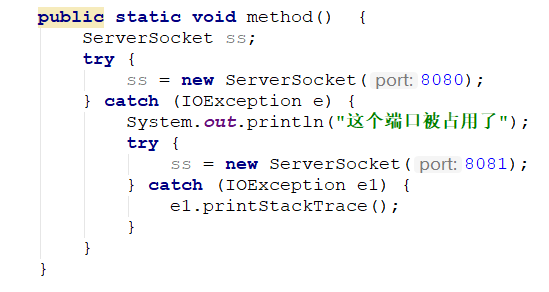
## 处理方式

## 1接着往上抛

运行时的异常自动向上抛，但是编译期的异常必须要手动的向上抛。



### 自己处理



运行时的异常一般都是向上抛出，编译期的异常都是自己处理。

# 自定义的异常

由于我们实际的项目中很多业务需要跟项目相关的异常，所以我们要自己来定义异常。

我们定义的都是运行时的异常。

库存不足：

自定义一个库存不足的异常

|  |
| --- |
| **public class** StockException **extends** RuntimeException {    **public** StockException() {  **super**();  }   **public** StockException(String message) {  **super**(message);  }   **public** StockException(String message, Throwable cause) {  **super**(message, cause);  }   **public** StockException(Throwable cause) {  **super**(cause);  }   **protected** StockException(String message, Throwable cause, **boolean** enableSuppression, **boolean** writableStackTrace) {  **super**(message, cause, enableSuppression, writableStackTrace);  } } |

自定义异常的用法

|  |
| --- |
| **public class** Test {   **public static void** main(String[] args) {  **try** {  *buy*(7);  } **catch** (Exception e) {  String message = e.getMessage();  System.***out***.println(message);  }  }   **public static void** buy(**int** count){  **if**(count > 5){  *//向上抛出异常的代码* **throw new** StockException(**"库存不足"**);  }  System.***out***.println(**"购买成功"**);  } } |