异常的捕获及处理机制：

·异常

·异常的捕获

·异常的处理

·常见的异常

异常的分类：

·错误error：编译时就会报的错误

·exception：程序运行中出现错误，导致程序中断执行

范例：简单异常的发生

|  |
| --- |
| **package** com.wanczy.exceptionDemo;  **public** **class** ExceptionDemo01 {  **public** **static** **void** main(String[] args) {  System.*out*.println(1/0);  }  } |

异常的捕获：在java中使用try{}catch（）{}捕获异常

|  |
| --- |
| try{  肯能出现异常的代码  }catch(异常的对象){  出现异常之后执行的代码  } |

|  |
| --- |
| **package** com.wanczy.exceptionDemo;  **public** **class** ExceptionDemo01 {  **public** **static** **void** main(String[] args) {  **try** {  System.*out*.println("异常出现之前");  System.*out*.println(1/0);  System.*out*.println("异常出现之后");  } **catch** (ArithmeticException e) {  System.*out*.println("捕获到异常");  }  }  } |

通过程序可以发现异常出现之后try中的代码不再执行，而是执行catch中的代码。

|  |
| --- |
| **package** com.wanczy.exceptionDemo;  **public** **class** ExceptionDemo01 {  **public** **static** **void** main(String[] args) {  **try** {  System.*out*.println("异常出现之前");  System.*out*.println(1/1);  System.*out*.println("异常出现之后");  } **catch** (ArithmeticException e) {  System.*out*.println("捕获到异常");  }  }  } |

继续观察发现如果try中没有出现异常，则会将try中代码执行完全，而不会去执行catch中的代码。

现在我们看看API：通过观察ApI发现Exception其实是一个类，继承自ThrowAble，此类有非常多的子类，这些子类都是异常，而且是非常详尽的异常。

|  |
| --- |
| **package** com.wanczy.exceptionDemo;  **public** **class** ExceptionDemo01 {  **public** **static** **void** main(String[] args) {  **try** {  System.*out*.println("异常出现之前");  // System.out.println(1/0);  **int** i[] = {1,3,4};  System.*out*.println(i[10]);  System.*out*.println("异常出现之后");  } **catch** (ArithmeticException e) {  e.printStackTrace();  System.*out*.println("捕获到异常");  }  }  } |

此程序中出现了数组下标越界异常，但是我们没有执行catch中的代码，这是因为catch只会去捕获算数异常。如果说要捕获数组下标越界异常的话，就必须使用数组下标越界异常的对象放在catch中进行捕获。

|  |
| --- |
| **package** com.wanczy.exceptionDemo;  **public** **class** ExceptionDemo01 {  **public** **static** **void** main(String[] args) {  **try** {  System.*out*.println("异常出现之前");  // System.out.println(1/0);  **int** i[] = {1,3,4};  System.*out*.println(i[10]);  System.*out*.println("异常出现之后");  } **catch** (ArithmeticException e) {//捕获算数异常  e.printStackTrace();  System.*out*.println("捕获到异常");  }**catch** (IndexOutOfBoundsException e) {//捕获数组下标越界异常  e.printStackTrace();  System.*out*.println("捕获到异常");  }  }  } |

现在可以告诉大家，Exception类是所有异常的最高父类，通过面向对象的多态性，是不是说所有Exception子类的对象都可以作为Exception对象，那么是不是可以使用Exception对象来捕获所有的异常呢？答案是肯定的。

|  |
| --- |
| **package** com.wanczy.exceptionDemo;  **public** **class** ExceptionDemo01 {  **public** **static** **void** main(String[] args) {  **try** {  System.*out*.println("异常出现之前");  System.*out*.println(1/0);  **int** i[] = {1,3,4};  System.*out*.println(i[10]);  System.*out*.println("异常出现之后");  } **catch** (Exception e) {//捕获算数异常  e.printStackTrace();  System.*out*.println("捕获到异常");  }  }  } |

现在有人在想，一段程序可能会出现多个异常，我先使用catch去捕获一些更细的异常，如果捕获不到再使用Exception去捕获。这个是可以的，但是有一点要说明：在捕获最粗的异常应该放在捕获最细的异常的后面。

|  |
| --- |
| **package** com.wanczy.exceptionDemo;  **public** **class** ExceptionDemo01 {  **public** **static** **void** main(String[] args) {  **try** {  System.*out*.println("异常出现之前");  // System.out.println(1/0);  **int** i[] = {1,3,4};  System.*out*.println(i[10]);  System.*out*.println("异常出现之后");  } **catch** (ArithmeticException e) {//捕获算数异常  e.printStackTrace();  System.*out*.println("捕获到异常");  }**catch** (IndexOutOfBoundsException e) {//捕获数组下标越界异常  e.printStackTrace();  System.*out*.println("捕获到异常");  }**catch**(Exception e){  e.printStackTrace();  System.*out*.println("Exception捕获到异常");  }  }  } |

但是大家可以发现，如果说细的异常都有相应异常的catch进行捕获了，则不会去执行最粗的异常的捕获。

|  |
| --- |
| try {  可能出现异常的代码  }catch (异常对象) {  出现异常之后执行的代码  }finally{  不管是否出现异常都会执行的代码  } |

这种程序结构会经常使用，尤其是在关闭数据库连接和关闭流的时候。

|  |
| --- |
| **package** com.wanczy.exceptionDemo;  **public** **class** ExceptionDemo01 {  **public** **static** **void** main(String[] args) {  **try** {  System.*out*.println("异常出现之前");  System.*out*.println(1/0);  **int** i[] = {1,3,4};  System.*out*.println(i[10]);  System.*out*.println("异常出现之后");  } **catch** (ArithmeticException e) {//捕获算数异常  e.printStackTrace();  System.*out*.println("捕获到异常");  }**catch** (IndexOutOfBoundsException e) {//捕获数组下标越界异常  e.printStackTrace();  System.*out*.println("捕获到异常");  }**catch**(Exception e){  e.printStackTrace();  System.*out*.println("Exception捕获到异常");  }**finally**{  System.*out*.println("不管有没有出现异常都会执行的代码，尤其实在关闭数据库连接和关闭流的操作");  }  }  } |

异常的处理机制：

·throws：写在方法的声明出，表示此方法不处理异常，而是在调用此方法的地方去处理异常，如果mian方法使用throws向上抛出异常，则表示将异常丢给JVM处理。

|  |
| --- |
| **package** com.wanczy.exceptionDemo;  **public** **class** ExceptionDemo01 {  **public** **static** **void** main(String[] args) {  **try** {  *funnction*();  } **catch** (Exception e) {  System.*out*.println("出现了异常");  e.printStackTrace();  }  }  **public** **static** **void** funnction() **throws** Exception{  System.*out*.println("异常出现之前");  System.*out*.println(1 / 0);  **int** i[] = { 1, 3, 4 };  System.*out*.println(i[10]);  System.*out*.println("异常出现之后");  }  } |

以上表示的是将异常丢个main处理

下面的代码表示将异常丢给JVM处理：

|  |
| --- |
| **package** com.wanczy.exceptionDemo;  **public** **class** ExceptionDemo01 {  **public** **static** **void** main(String[] args) **throws** Exception{  *funnction*();  }  **public** **static** **void** funnction() **throws** Exception {  System.*out*.println("异常出现之前");  System.*out*.println(1 / 0);  **int** i[] = { 1, 3, 4 };  System.*out*.println(i[10]);  System.*out*.println("异常出现之后");  }  } |

·throw：自定义异常，在一般的开发中肯定使用不到，但是如果今后技术已经到了登峰造极的程度，肯定会使用到，因为自定义异常一般在开发框架或者自定义组件。

|  |
| --- |
| **package** com.wanczy.exceptionDemo;  **public** **class** ExceptionDemo02 {  **public** **static** **void** main(String[] args) **throws** Exception{    *funnction*();  }  **public** **static** **void** funnction() {  //自定义异常其实就是实例化一个Exception的对象，自定义异常的名称  **try** {  **throw** **new** Exception("中午不睡，下午奔溃");  } **catch** (Exception e) {  e.printStackTrace();  }      }  } |