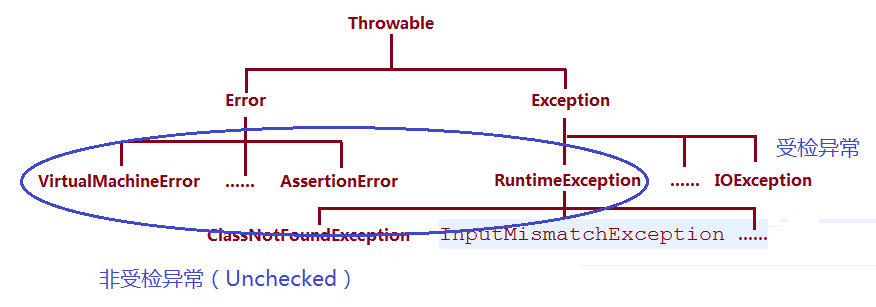
1. 异常
2. 概述

异常是不可预知的错误。在编程语言中，所谓异常是指在程序运行过程中所出现的不可预知的错误。异常将导致程序的执行中断(异常语句后面的代码不会被执行)。

Java中的异常是以对象的形式存在的，然后以堆栈的形式抛出。

1. 结构
2. 异常类



1. 描述

Throwalbe ：所有异常类的基类

|----- Error ：表示底层异常，Error异常是不需要进行捕| 获的误。如：StackOverflowError堆栈溢出错误等

|-----Exception ：表示程序代码中可以捕获的异常。Exception

| 异常是需要我们进行处理的异常

| -------非受检异常(Unchecked) ：非受检异常是指在

| 程序编译阶段不会出现，但是在运行阶段可能

| 会出现的异常。

|--------受检异常(Checked) ：受检异常是指在代

码编写(编译)阶段编译器就会进行提示的异常。

一般需要进行资源释放的代码几乎都使用了受

检异常。如：数据操作、Socket通信、IO操作

等。

1. 异常处理语句
2. 语法

|  |
| --- |
| Try{  可能出现异常的代码；  }  Catch(异常类型 对象名){  捕获并处理异常  }  ……  Catch(异常类型 对象名){  捕获并处理异常  }  Finally{  资源释放  } |

抓：捕获异常。获取异常出现的位置及异常对象

抛：将异常对象返回给调用者，由调用者进行处理(如果不处理则

程序中断)

1. 说明

* Try语句

Try语句相当于一个安全区，将可能会出现异常的代码都放在该语句块中。

* Catch语句

Catch语句对try语句块中出现的异常进行捕获。Catch语句块捕获到异常后可以进行处理(try语句块没有异常产生时catch语句块是不会被执行的)。一个try语句可以有多个catch语句块。

* Finally

Finally语句块用于对前面所用到的资源进行释放。Finally语句块是一定会被执行的代码块。

1. 使用形式

* Try-catch
* Try-finally
* Try-catch-finally

1. 示例

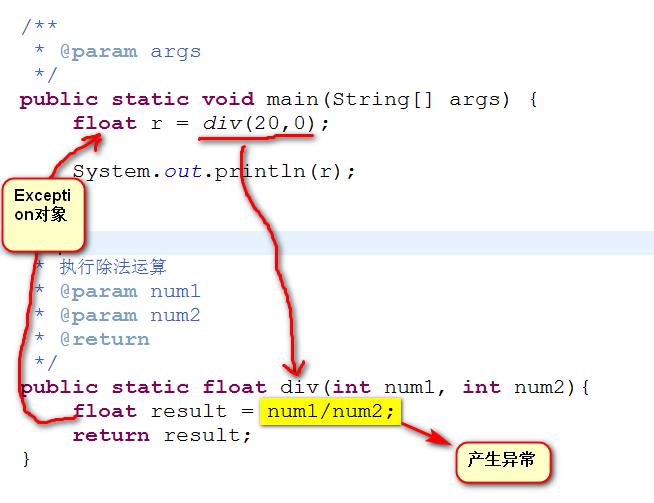
|  |
| --- |
| //非受检异常  Scanner input = **new** Scanner(System.in);  System.out.println("1、添加学生信息");  System.out.println("2、查询学生信息");  System.out.print("请选择功能项(1~2)：");    //接收用户的输入  //输入非数字时将产生异常。在程序运行过程中产生的  //在后面进行输出没有实际意义  //int item =0;  **try**{  **int** item = input.nextInt();  //如果出现错误该行代码不会被执行  System.out.println(item);  }  //catch(Exception ex){  **catch**(InputMismatchException ex){  System.out.println("您输入的数据类型有误，操作失败");  } |
| FileInputStream fs= **null**;  **byte**[] b = **new** **byte**[1024];    **try** {  fs = **new** FileInputStream("d:\1.txt");    //将文件内容读取到byte[]中  **while**(fs.read(b) !=-1){}  } **catch** (FileNotFoundException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  } **catch** (IOException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  **finally**{  //关闭流对象  **try** {  fs.close();  } **catch** (IOException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  } |

1. Try-catch执行过程

* 代码

|  |
| --- |
| /\*\*  \* **@param** args  \*/  **public** **static** **void** main(String[] args) {  **float** r = div(20,0);    System.out.println(r);  }    /\*\*  \* 执行除法运算  \* **@param** num1  \* **@param** num2  \* **@return**  \*/  **public** **static** **float** div(**int** num1, **int** num2){  **float** result = num1/num2;  **return** result;  } |

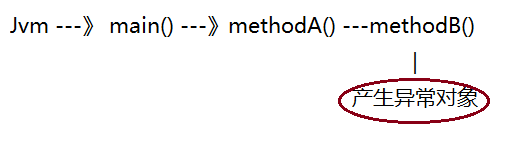
* 执行流程



* 流程描述
* Main()方法中调用了div()方法
* Div()方法执行过程中出现异常，从异常位置开始jvm会创建一个异常对象。然后将异常对象向上进行返回(返回给div()方法的调用者---main())。异常语句后面的所有代码都会被终止执行。
* Main()方法中接收到异常后继续进行返回。将异常返回给Jvm。

1. 执行原理

Java中的异常是以堆栈的方式进行处理的。例如有如下方法调用过程：



首先Jvm调用了main()方法，main()方法优调用了method A ()，而methodA()有调用了methodB()。在methodB执行过程产生了异常。

Jvm会实例化一个异常对象，然后将异常对象抛给methodA()方法，methodA()方法又将异常对象抛给了main()。然后main()继续向上抛除，将异常抛给了Jvm。

Jvm无法直接进行处理，所以通过printStackTrace()方法打印异常信息。

1. Finally
2. 作用

用于进行资源释放。

1. 执行

* 没有异常产生时，catch块不会被执行。Finally执行了
* 有异常产生时，catch和finally都会被执行

语句执行过程中首先从try语句块开始进行执行，出现异常时首先执行对应的catch语句块。当遇到return等跳转语句时，跳转语句将被忽略，转而执行finally语句块。

执行finally语句块时，如果finally中有return语句时则直接跳出方法，否则将从finally返回到catch中并执行catch语句中的return语句。

1. [面试题]：简述final和finally的区别？

* Final用于修饰类及类的成员，final修饰类表示该类不可继承；修饰方法表示方法不可被重写，修饰字段字段称为常量，初始化后其值初始化后不能在被修改；
* Finally是异常处理语句的一部分，用于进行资源释放。

1. Throw与throws
2. 异常的处理方式
   * 直接在输出窗口中打印堆栈中的异常信息(代码调试时这种方式居多)
   * 不进行输出打印吗，而将错误信息存储到文件或数据库中(日志，发布的项目多用这种方式)
   * 不进行错误处理，将错误向上进行抛出，由上一级调用者进行处理(一般在通用类库中应用较多)
3. Throw

* 作用
* 将现有异常对象抛给调用者
* 实例化一个新的对象并抛给调用者
* 示例

|  |
| --- |
| /\*\*  \* 异常处理方式  \*/  **public** **static** **int** div(){  **try**{  **return** 20/0;  }  **catch**(ArithmeticException ex){  //1、直接在输出窗口中打印堆栈中的异常信息(代码调试时这种方式居多)  //ex.printStackTrace();    //2、不进行输出打印吗，而将错误信息存储到文件或数据库中(日志)  //3、不进行错误处理，将错误向上进行抛出，由上一级调用者进行处理  **throw** ex;  }  //return -1;  } |
| **public** **void** setAge(**int** age) {  **if**(age >=18 && age<=25)  **this**.age = age;  **else**  //实例化一个异常对象并向上抛出  **throw** **new** RuntimeException("您的年龄不符征兵要求");  } |

1. Throws

* 作用
* Throws用于声明可以抛出的异常类型
* 有异常产生时异常对象将被向上进行抛出
* 示例

|  |
| --- |
| /\*\*  \* 方法中不希望对异常进行处理  \* 将异常抛给调用者  \*/  **public** **static** **void** getText2() **throws** FileNotFoundException,IOException{  FileInputStream fs = **new** FileInputStream("d:\1.txt");  **byte**[] b = **new** **byte**[1024];    //将文件内容读取到byte[]中  **while**(fs.read(b) !=-1){}  //关闭流对象  fs.close();  } |

1. [面试题]：简述throw和throws的区别。

* Throw用在方法内部；throws用于在方法声明部分。
* Throw用于向上抛出一个异常对象；throws用于声明可以抛出的异常类型；

1. 自定义异常
2. 步骤

* 定义类并继承于相应的异常类型，一般我们定义的类继承于RuntimeException
* 定义构造函数
* 加入自定义功能

1. 示例

|  |
| --- |
| /\*\*  \* 自定义异常类  \* **@author** X230  \*  \*/  **public** **class** MyException **extends** RuntimeException{  /\*\*  \*  \*/  **private** **static** **final** **long** *serialVersionUID* = 10090L;    **public** MyException(String message) {  **super**(message);    //将异常信息写入到数据库  System.*out*.println("异常信息：" + message+"已写入到数据库");  }  } |
| **public** **class** Student {  **private** String name;  **private** **int** age;  **public** String getName() {  **return** name;  }  **public** **void** setName(String name) {  **this**.name = name;  }  **public** **int** getAge() {  **return** age;  }  **public** **void** setAge(**int** age) {  **if**(age>=18 && age <=25)  **this**.age = age;  **else**  //使用了自定义异常  **throw** **new** MyException("您不符合征兵要求");  }      } |
| **public** **class** Tester {  /\*\*  \* **@param** args  \*/  **public** **static** **void** main(String[] args) {  Student stu = **new** Student();  stu.setName("Mike");  **try**{  stu.setAge(28);  }  **catch**(Exception ex){  System.*out*.println(ex.getMessage());  }  }  } |

1. Exception类
2. 成员

* getMessage() ：获取异常消息
* getStackTrace()：获取异常的堆栈信息