

异常机制

在编写一段代码时, 我们肯定都会想程序按照我的思想去运行, 但是如果代码实现可能会不够完美, 可能会有没考虑到的地方, 如果还可以正常运行还好, 如果出现错误或者异常将程序终止了那该怎么办?

在正常情况下, 在程序不能正常运行就会提示报错或者异常

异常类型

异常有很多个类型比如, 越界异常, 空指针异常, 算术异常单独; 每一个异常它都是一个类, 他们都继承自 `Exception` 类, 异常的本质也是需要依赖对象, 但是异常类型支持在程序运行出现问题时抛出, 也可以提前声明, 告知使用者需要处理可能出现的异常

异常的类型也有很多有运行时异常和编译时异常

1 运行时异常是指, 只有在程序运行的时候才知道会不会出错, 这样的异常称为运行时异常, **所有的运行时异常都是继承自 `RuntimeException`**

```
1 public class Main {
2
3     public static void main(String[] args) {
4         test(10,0); //被除数为0会报错
5
6     }
7
8     public static int test(int a, int b){
9         return a/b; //除法运算
10    }
11
12 }
```

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at com.company.zy01.Main.test(Main.java:12)
    at com.company.zy01.Main.main(Main.java:7)
```

这里就报错了 `ArithmeticException` 运算错误, 可以点进去看一下

```
1 //可以看到它也是一个类并且继承自RuntimeException
2 public class ArithmeticException extends RuntimeException {
3
4     private static final long serialVersionUID = 2256477558314496007L;
5
6     /**
7      * Constructs an {@code ArithmeticException} with no detail
8      * message.
9      */
10    public ArithmeticException() {
11        super();
12    }
13 }
```

```

12     }
13
14     /**
15      * Constructs an {@code ArithmeticException} with the specified
16      * detail message.
17      *
18      * @param s the detail message.
19      */
20     public ArithmeticException(String s) {
21         super(s);
22     }
23 }
24

```

2 还有一种编译时异常，编译时异常明确指出可能会出现的异常，在编译阶段就需要进行处理，必须要考虑到出现异常的情况，如果不进行处理就无法进行编译，默认继承自 `Exception` 类的异常都是编译时异常，在 `Object` 类中定义了一个 `Clone` 方法，就明确指出了可能会出现异常使用 `throws` 关键字

```

1 | protected native Object clone() throws CloneNotSupportedException;

```

它是继承自 `Exception`

```

1 | public
2 | class CloneNotSupportedException extends Exception {
3 |     private static final long serialVersionUID = 5195511250079656443L;
4 |
5 |     /**
6 |      * Constructs a <code>CloneNotSupportedException</code> with no
7 |      * detail message.
8 |      */
9 |     public CloneNotSupportedException() {
10 |         super();
11 |     }
12 |
13 |     /**
14 |      * Constructs a <code>CloneNotSupportedException</code> with the
15 |      * specified detail message.
16 |      *
17 |      * @param s the detail message.
18 |      */
19 |     public CloneNotSupportedException(String s) {
20 |         super(s);
21 |     }
22 | }
23 |

```

3 还有一种就是错误，错误比异常更加严重，异常不一定会导致致命的问题，而错误就是致命的问题，一般出现错误可能 `JVM` 就出无法进行正常运行了，看下面这个例子

```
1 public class Main {  
2  
3     public static void main(String[] args) {  
4         test();  
5  
6     }  
7  
8     public static void test(){  
9         test();  
10    }  
11  
12 }  
13
```

这样无限循环会出现就会出现内存溢出的问题，这里已经时错误了，错误就会影响到整个程序的运行了

```
public class Main {

    public static void main(String[] args) {
        test();
    }

    2 个用法
    public static void test(){
        test();
    }

}

Main x
D:\software\Java\jdk1.8.0_131\bin\java.exe ...
Exception in thread "main" java.lang.StackOverflowError Create breakpoint
    at com.company.zy01.Main.test(Main.java:12)
    at com.company.zy01.Main.test(Main.java:12)
    at com.company.zy01.Main.test(Main.java:12)
    at com.company.zy01.Main.test(Main.java:12)
    at com.company.zy01.Main.test(Main.java:12)
    at com.company.zy01.Main.test(Main.java:12)
    at com.company.zy01.Main.test(Main.java:12)
    at com.company.zy01.Main.test(Main.java:12)
    at com.company.zy01.Main.test(Main.java:12)
    at com.company.zy01.Main.test(Main.java:12)
    at com.company.zy01.Main.test(Main.java:12)
    at com.company.zy01.Main.test(Main.java:12)
```

还有一种情况就是内存溢出，JVM它是有一个内存限制的，如果超出了它的内存限制，就会导致内存溢出

```

1 public class Main {
2
3     public static void main(String[] args) {
4         Object[] objects = new Object[Integer.MAX_VALUE];
5
6     }
7
8     public static void test(){
9         test();
10    }
11
12 }

```

```

public class Main {
    public static void main(String[] args) {
        Object[] objects = new Object[Integer.MAX_VALUE];
    }

    1 个用法
    public static void test(){
        test();
    }
}

```

Main x

D:\software\Java\jdk1.8.0_131\bin\java.exe ...

Exception in thread "main" java.lang.OutOfMemoryError: Create breakpoint : Requested array size exceeds VM limit
at com.company.zy01.Main.main(Main.java:7)

抛出异常

1 调用方法时，如果传入了错误的参数而导致程序无法进行正常运行，这个时候就可以手动抛出一个异常来终止程序，同时告诉上一级方法执行出现了问题，需要使用到 `throw` 关键字

```

1 public class Main {
2
3     public static void main(String[] args) {
4         test(10,0);
5
6     }
7
8     public static int test(int a, int b){
9         if(b==0){
10             throw new RuntimeException("被除数不能为0! ");

```

```

11     }
12     return a/b;
13 }
14
15 }

```

异常的抛出同样需要船舰一个异常类出来，我们抛出的异常实际上就是这个异常对象的抛出，异常对象还携带我们抛出异常的一些信息，我们在这这里可以明确指定声明原因导致异常

```

Exception in thread "main" java.lang.OutOfMemoryError: Create breakpoint : Requested array size exceeds VM limit
at com.company.zy01.Main.main(Main.java:7)

```

2 在抛出异常时，程序会终止，还会打印栈追踪信息，也就是会打印在哪里出现了错误

3 如果抛出的异常时非运行时异常，那么就必须要告知函数的调用方法我们会抛出某个异常，函数的调用方法必须对这个异常进行处理才可以比如（并不是非运行时异常才可以，运行时异常也可以不过没什么意义）

```

1 public class Main {
2
3     public static void main(String[] args) throws Exception { //当然调用一方也可以
        无视，直接进行向上抛，这样抛就是抛给JVM，那就是程序直接结束了
4         test();
5
6     }
7
8     public static void test() throws Exception { //使用throw告诉调用需要处理好这个可
        能会的出现的异常
9         throw new Exception("我是编译时异常");
10    }
11
12 }

```

4 如果不同的分支条件会出现不同的异常，那么所有在方法中可能会抛出的异常都需要注明，也就是可能会抛出多个非运行时异常，他们都需要使用 `throw` 告知

```

1     public static void main(String[] args) throws Exception {
2         test(1);
3
4     }
5
6     public static void test(int a) throws FileNotFoundException,
        ClassNotFoundException {
7         if(a==1)
8             throw new FileNotFoundException();
9         else
10            throw new ClassNotFoundException();
11    }
12

```

异常的处理

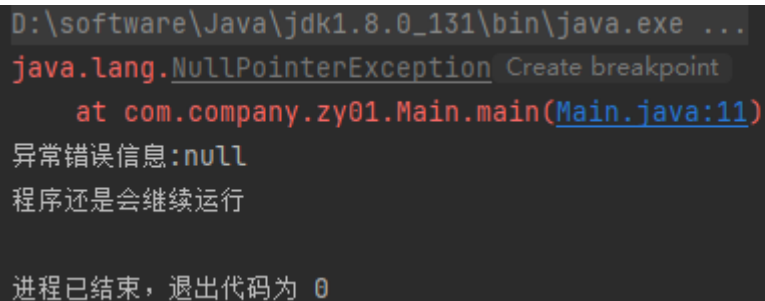
1 当程序没有按照我们理想的样子运行时就会出现异常，出现异常就会使得程序停止，现在可以自己处理出现的问题让程序进行运行，使用 `try catch`

```
1 public static void main(String[] args) {
2     try{ //相当于try内地代码被监控了
3         Object object = null;
4         object.toString();
5     }catch (NullPointerException e){//这里就可以捕获出现的异常
6
7     }
8     System.out.println("程序还是会继续运行");
9 }
```

⚠ `catch` 捕获的只能说 `Throwable` 的子类，要么是异常要么是错误，不能是其他任何类型

在 `catch` 中可以直接对捕获到的异常进行处理

```
1 public static void main(String[] args) {
2     try{
3         Object object = null;
4         object.toString();
5     }catch (NullPointerException e){
6         e.printStackTrace(); //打印栈追踪信息,当然不止这一个方法还有很多
7         System.out.println("异常错误信息:"+ e.getMessage()); //获取异常错误信息
8     }
9     System.out.println("程序还是会继续运行");
10 }
```



The screenshot shows a Java IDE window with the following content:

```
D:\software\Java\jdk1.8.0_131\bin\java.exe ...
java.lang.NullPointerException Create breakpoint
    at com.company.zy01.Main.main(Main.java:11)
异常错误信息:null
程序还是会继续运行

进程已结束，退出代码为 0
```

2 之前说过调用的方法如果告知了会抛出非运行时异常，调用方可以选择继续向上抛或者选择 `try-catch` 语句进行异常的捕获，不然就无法通过编译

```

1      public static void main(String[] args) { //当然调用一方也可以无视，直接进行向上
      抛，这样抛就是抛给JVM，那就是程序直接结束了
2          try {
3              test();
4          } catch (Exception e) {
5              throw new RuntimeException(e);
6          }
7      }
8
9
10     public static void test() throws Exception { //使用throw告诉调用需要处理好这个可
      能会出现的异常
11         throw new Exception("我是编译时异常");
12     }

```

3 如果我们要捕获的异常是某个异常的父类，那么发生这个异常时，也可以被捕获

```

1  public static void main(String[] args){
2      try {
3          int[] arr = new int[1];
4          arr[1] = 100; //这里发生的是数组越界异常，它是运行时异常的子类
5      } catch (RuntimeException e){ //使用运行时异常同样可以捕获到
6          System.out.println("捕获到异常");
7      }
8  }

```

4 当代码出现多种类型的异常时，可以使用多个catch来处理不同的异常，不过要注意捕获异常的顺序，如果第一个捕获的异常是第二个捕获异常的父类，发生了第二个异常，会直接捕获第一个异常，也就会直接使用它的父类来捕获

```

1  try {
2      //....
3  } catch (RuntimeException e){ //父类型在前，会将子类的也捕获
4
5  } catch (NullPointerException e) { //永远都不会被捕获
6
7  } catch (IndexOutOfBoundsException e){ //永远都不会被捕获
8
9  }

```

可以简写为

```

1  try {
2      //....
3  } catch (NullPointerException | IndexOutOfBoundsException e) { //用|隔开每种类型即可
4
5  }

```


5 如果希望无论出现或者不出现异常都要执行这个任务可以使用 `finally`

```
1 try {  
2     //....  
3 }catch (Exception e){  
4  
5 }finally {  
6     System.out.println("lbwnb");    //无论是否出现异常，都会在最后执行  
7 }
```

⚠ `try`至少要搭配 `catch` 或者 `finally` 中的一个

断言表达式

我们可以使用断言表达式来对某些东西进行判断，如果判断失误就会抛出错误，只不过默认情况下没有开启，需要手动开启

(我觉得这个没啥用我直接过了)