《面向对象》之异常、包

笔记本: JAVA
创建时间: 2018/8/8 12:09 **更新时间:** 2018/8/16 14:45
作者: Debao Xu

异常

异常是指Java程序在 **运行** 时期发生的不正常情况（问题）。Java就按照面向对象的思想对不正常情况进行描述和对象的封装。

问题分为两种：
1、Error：由**系统底层发生**的，再告诉jvm，接着再告诉使用者。对于这种问题，不做针对性处理，直接修改代码。
2、Exception：**由jvm发生**，并告诉使用者。对于这种问题，可以进行针对性的处理

```
public class Demo
{
	public static void main(String[] args)
	{
		int[] arr = new int[1];

		//这里的异常是“角标越界”，那么这个异常是如何发生的呢？输出语句发生问题时，jvm就将这个已知的问题封装成了对象，throw new ArrayIndexOutOfBoundsException: 1,将问题抛给了调用者main函数，而main函数没有针对性的处理方式，main就继续往外抛给调用者jvm，jvm就使用了默认的处理方式。即将“问题的名称+信息+位置”在控制台上显示出来，让调用者看到并结束程序。
		System.out.println(arr[1]);
	}
}
```

运行结果：

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 1
	at Demo.main(Demo.java:7)
```

异常的处理

1、遇到问题不进行具体的处理，而是继续抛给调用者，其实就是在函数上通过关键字throws来声明异常，告诉调用者处理在编写功能时，编写者知道该功能有可能发生问题，而这个问题很容易来自于调用者传递的参数，从而导致功能无法运行。这时发生的问题就应该让调用者知道，并最好让调用者有预先的处理方式。所以在定义功能时，需要在功能上对有可能发生的问题进行声明。声明问题需要使用关键字 throws（异常类），声明的目的：将是让调用者可以进行处理。

2、针对性的处理方式：捕获！

```
try
{
	//有可能发生异常的代码
}
catch(异常类 变量)
{
	//这是真正的捕获，处理异常的代码
}
finally
{
	//一定会被执行的代码
}
```

```
class exception
{
	int div(int a,int b) throws Exception //这里的Exception是一个父类，它有很多的子类
	{
		return a/b;
	}
}
```

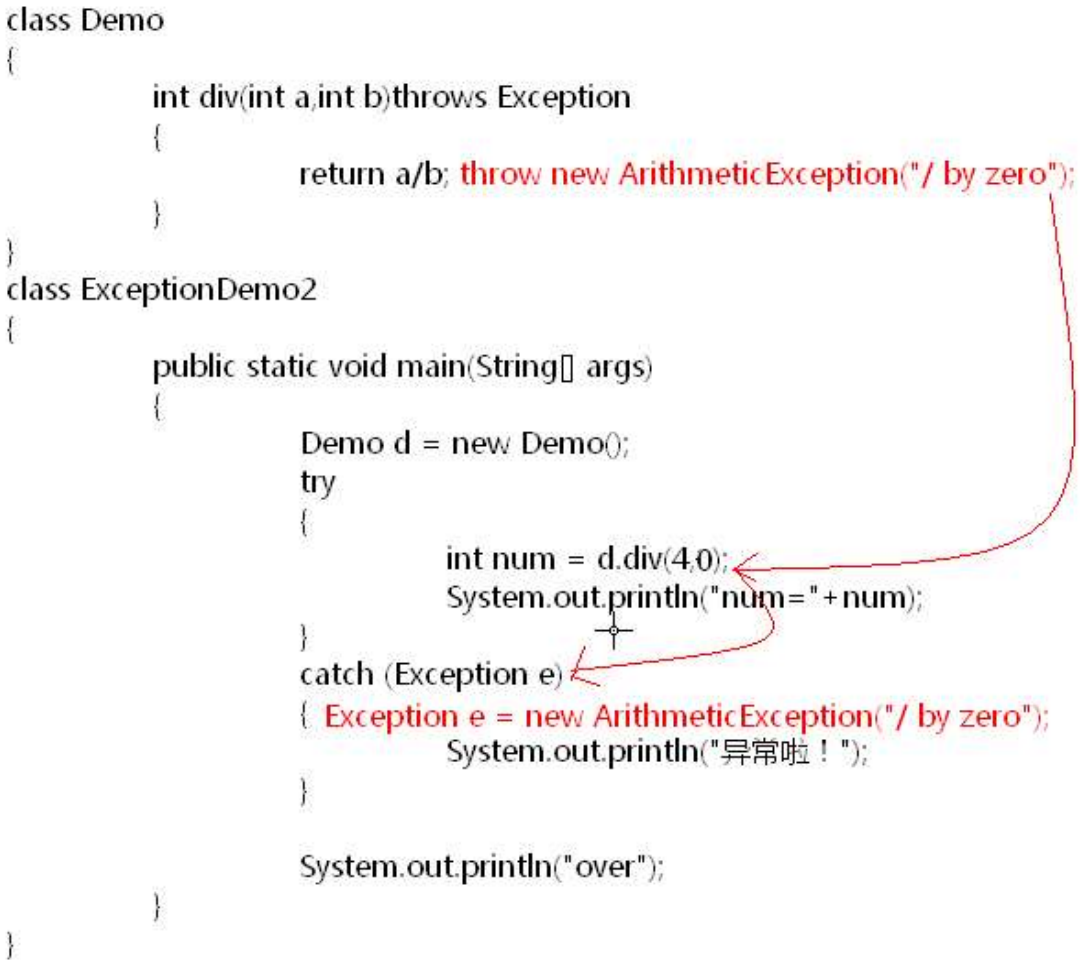
```
public class Demo
{
	public static void main(String[] args) throws Exception
	{
		try
		{
			new exception().div(4, 0);
		}
		catch(Exception e) //该参数是一个“类”，变量是e
		{
			System.out.println("异常啦！ ");
		}
		System.out.println("Over!");
	}
}
```

运行结果：

```
异常啦!
Over!
```

//从运行结果可以看出，对异常进行针对性处理之后，可以继续向下执行

异常出现的步骤，如下图，



当执行到 return 4/0 语句时，抛出异常“throw new ArithmeticException("/by Zero")”,接着语句“int num = d.div(4,0)”就失效了，接着就会把异常抛给catch来捕获，然后打印出“异常啦！”。之后，继续执行语句打印出“over！”

```
class exception
{
	int div(int a,int b) throws Exception //声明异常
	{
		//自定义一个异常，这里用的是关键字 throw,不是throws
		if(b==0)
			throw new ArithmeticException("除零啦！废了！ ");//抛出异常对象
```

```
		return a/b;
	}
}
public class Demo
{
	public static void main(String[] args) throws Exception
	{
		try
		{
			new exception().div(4, 0);
		}
		catch(Exception e)
		{
			System.out.println("异常啦！ ");
			System.out.println(e.toString());//异常名称+异常信息
			System.out.println(e.getMessage());//异常信息
			e.printStackTrace();//名字+信息+位置。jvm默认收到处理异常就是调用这个方法，将信息显示在屏幕上。

//以上的toString(), getMessage(), printStackTrace()方法不是类Exception中的，而是来自于Exception的父类Throwable中的方法。具体可以查看API文档。
		}
		System.out.println("Over!");
	}
}
```

上面的代码中用了两个关键字 throw和throws ，那么这两个关键字有什么区别呢?
1、位置不同
throws用在**函数上**，后面跟的是异常**类**，可以跟多个。
throw用在**函数内**，后面跟的是异常**对象**

2、功能不同
throws用来**声明异常**，让调用者只知道该功能有可能出现的问题，并由调用者可以给出预先的处理方式
throw**抛出具体问题的对象**，执行到throw功能就已经结束了（即throw后的语句都“废了”），然后直接跳转到调用者，并将具体的问题对象也抛给调用者。也就是说，throw语句独立存在时，下面不要定义其他的语句，因为根本执行不到。

异常体系的特殊情况:
异常体系的最大特点就是体系中的类以及类产生的对象，都具备**可抛性**，可抛性的意思是可以被throw和throws所操作。

异常的原则:
1、功能内部有异常throw抛出，功能上一定要throws声明。内部抛什么，功能上就声明什么（"抛"要和"声明"相对应）。
声明的目的就是为了让调用者处理，如果调用者不处理，则编译失败。

2、特殊情况:
当函数内通过throw抛出了RuntimeException及其子类的异常对象时，函数上可以不用throws声明。那么，不声明的目的是什么呢？不声明的目的就是为了不让调用者处理，而是让调用者的程序停止，从而对代码进行修改，用下面的2个例子说明，

```
//这个例子说明了抛出了异常，并且程序可以继续向下执行
class exception
{
	int div(int a,int b) throws Exception  //用throws声明
	{
		if(b==0)
			throw new Exception("除零啦！废了！ ");
		return a/b;
	}
}
public class Demo
{
```

```
        public static void main(String[] args) throws Exception
        {
                try
                {
                        new exception().div(4, 0);
                }
                catch(Exception e)
                {
                        System.out.println("异常啦！ ");
                        System.out.println(e.toString());
                        System.out.println(e.getMessage());
                        e.printStackTrace();
                }
                System.out.println("Over!,异常结束继续执行下面的语句。");
        }
}
```

运行结果：

```
异常啦！
java.lang.Exception: 除零啦！废了！
除零啦！废了！
java.lang.Exception: 除零啦！废了！
        at exception.div(Demo.java:6)
        at Demo.main(Demo.java:16)
Over!,异常结束继续执行下面的语句。
```

上面的例子说明了在出现异常时，通过throws声明，可以让异常抛出后程序继续向下执行，但是，**有的时候由于异常中的某些数据在程序的后面仍然存在**，这个时候如果继续让程序执行下去，那么后面的程序在执行时实际上已经是错的了。因此，这个时候，我们需要的是在**程序出现异常时，程序可以自动停下来**，让我们来修改代码，那么，我们就需要不用throws声明即可，如下面的代码，

```
class exception
{
        int div(int a,int b)  //不用throws声明
        {
                if(b==0)
                        throw new ArithmeticException("除零啦！废了！ ");
                return a/b;
        }
}
public class Demo
{
        public static void main(String[] args) throws Exception
        {
                exception e = new exception();
                int num = e.div(4, 0);
                //System.out.println("Over!,异常结束,而且不能继续执行下面的语句。");
        }
}
```

通过上面的两个例子，我们可以看出Exception可以分为两种：
1、编译时就会被检测出的异常
2、运行时才会被检测出的异常（编译时不检测），如RuntimeException及RuntimeException的子类。

举例，

```
public class Demo
{
        public static void main(String[] args) throws Exception
        {
                int[] arr = new int[3];
```

```
            System.out.println("element: "+getElement(arr,-2)); //角标为 -2，那么一定会抛出异常
        }
        public static int getElement(int [] arr,int index) //不用throws声明异常，为了运行过程中出了异常，程序可以停下来
        {
            if(index<0 || index>arr.length)
            {
                throw new ArrayIndexOutOfBoundsException(index+"角标不存在");//抛出异常
            }

            int element = arr[index];
            return element;
        }
}
```

运行结果：

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: -2角标不存在
        at Demo.getElement(Demo.java:21)
        at Demo.main(Demo.java:15)
```

自定义异常

在自定义的程序中，如果有了问题，也可以像Java中的异常一样，对问题进行描述。
举例：定义一个功能可以实现除法运算，但是除数不可以为负数。

```
class FushuException extends RuntimeException //为了让该类具有“可抛性”，那么它必须要继承自带的异常类。此外，我们想让该类在运行时检测异常，所以继承了类 RuntimeException
{
        FushuException(String message)
        {
            super(message);//将参数传递给父类，即调用父类中的函数就可以了，如果没有这一句，参数"除数小于零啦！"就传递不出去
        }
}
class exception
{
        int div(int a,int b)
        {
            if(b<0)
                throw new FushuException("除数小于零啦！");//Exception类中没有专门修饰“除数不可以小于0”的异常，所以该异常需要自己定义
            if(b==0)
                throw new ArithmeticException("除零啦！废了！");
            return a/b;
        }
}
public class Demo
{
        public static void main(String[] args) throws Exception
        {
            exception e = new exception();
            try
            {
                System.out.println(e.div(4, -5));
            }
            catch(Exception d) //捕获异常
            {
                System.out.println(d.toString());
            }
            System.out.println("Over!");
        }
}
```

finally的作用
无论是否有异常发生，都要对资源进行释放，而资源释放的动作就定义在finally代码块中

```
public class Demo
{
	public static void main(String[] args)
	{
		try
		{
			int num = 4/0;
			System.out.println(num);
		}
		catch(Exception e)
		{
			System.out.println(e.toString());
			//System.exit(0); 这个语句用于退出jvm，只有在这种情况下，finally代码块也不执行
		}
		finally
		{
			System.out.println("finally");
		}
		System.out.println("over!");
	}
}
```

运行结果：

```
java.lang.ArithmeticException: / by zero
finally
over!
```

异常的针对性处理方式：

try
{ }
catch()
{ }
finally
{ }

的几种组合方式：

1、没有资源需要释放（即没有finally代码块），仅仅是处理异常

```
try
{
}
catch()
{

}
```

2、一个try多个catch，一般对应的是被调用的函数抛出多个异常，要分别进行处理

```
try
{
}
catch()
{

}
catch()
{
```

```
}
catch()
{

}
finally
{

}
```

注意：在多catch语法上特殊的地方，如果catch中的异常类存在子父类，**父类的catch一定要放在子类的catch的下面**，否则编译会失败

3、不一定要处理异常，但是有资源需要释放

```
try
{
}
finally
{

}
```

4、既处理异常，又释放资源

```
try
{
}
catch()
{

}
finally
{

}
```

覆盖中异常的使用：

子类方法覆盖父类方法只能抛出父类方法异常或该异常的子类。如果父类方法抛出多个异常，子类只能抛出父类异常的子集。原则就一个：就是子类的异常必须要在父类的异常处理控制中。

包

1、对文件进行分类管理
2、给类提供多层命名空间
3、写在程序文件的第一行
4、类名的全称是 包名.类名
5、包也是一种封装形式。（相当于文件夹）

包之间的访问：
假设现在有两个包，mypack，mypack2，他们中的程序分别如下：

```
package mypack;
public class Demo
{
	public static void main(String[] args)
	{
		mypack2.Demo2 d2 = new mypack2.Demo2(); //有了包以后，类的名称应该是：包名.类名
		d2.show2();
	}
}
```

运行结果：
show2

```
package mypack2;
public class Demo2  //包中的类只有是公共的（用关键字public修饰过），才能被别的包中的程序访问
{
	public void show2() //同样对于方法，也要是public的，才能被其他的包调用
	{
		System.out.println("show2");
	}
}
```

包之间的继承：
对于上面的两个包，之间可以实现继承，

```
package mypack2;
public class Demo2  //包中的类只有是公共的（用关键字public修饰过），才能被别的包中的程序访问
{
	public void show2() //同样对于方法，也要是public的，才能被其他的包调用
	{
		System.out.println("show2");
	}
    public static void show3()
	{
		System.out.println("被另一个包调用");
	}
}
```

```
package mypack;
public class Demo extends mypack2.Demo2
{
	public static void main(String[] args)
	{
		mypack2.Demo2 d2 = new mypack2.Demo2(); //有了包以后，类的名称应该是：包名.类名
		d2.show2();

		show3(); //继承来的方法
	}
}
```

import关键字导入“类”

```
package mypack;
import mypack2.Demo2;
// import mypack.*    ,利用通配符* 来导入包中所有的类，但是这种方法不常用

public class Demo extends mypack2.Demo2
{
	public static void main(String[] args)
	{
		// mypack2.Demo2 d2 = new mypack2.Demo2();  要写包名
		Demo2 d2 = new Demo2(); // 有了import关键字导入包中的类之后，就不用写包名了

		d2.show2();
	}
}
```