第九章：异常处理

教学内容：

1. java 异常处理
2. throw 语句
3. throws 语句
4. 运行时异常

一、 java 异常处理

什么是异常：

- 运行时发生的错误称为异常。处理这些异常就称为异常处理。
- 一旦引发异常，程序将突然中止，且控制将返回操作系统(JVM)。
- 发生异常后此前分配的所有资源都将保留在相同的状态，这将导致资源漏洞。

java 的异常处理：

当一个程序出现错误时，它可能的情况有 3 种：

语法错误：运行时错误和逻辑错误。语法错误是指代码的格式错了，或者某个字母输错了；

运行时错误：是指在程序运行的时候出现的一些没有想到的错误，如：空指针异常，数组越界，除数为零等；

逻辑错误：是指运行结果与预想的结果不一样，这是一种很难调试的错误。而 java 中的异常处理机制主要是指处理运行时错误，即异常就是运行时错误。

产生异常的原因有 3 中：

1.java 内部发生错误，java 虚拟机产生的异常。

2.编写程序的时候由于错误引起的异常，如：空指针异常，数组越界等。

3.通过 throw 语句生成的异常。这种异常通常称为“检查异常”，用来告知方法的调用着相关信息。

异常示例：

```
public class Test {

	public void say() {
		System.out.println("Test is say!");
	}

	public static void main(String[] args) {
		Test t = null;

		t.say();//运行时会发生空指针异常
	}

}
```

```
Exception in thread "main" java.lang.NullPointerException
	at com.mxp.iweb02.Test.main(Test.java:12)
```

```
public class Test {

	public static void main(String[] args) {
		System.out.println(1/0);//会发生除数为0异常
	}

}
```

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at com.mxp.iweb02.Test.main(Test.java:6)
```

```
public class Test {

    public static void main(String[] args) {
        try {
            Class.forName("com.mxp.test.Test");//会发生类加载异常
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }

}
```

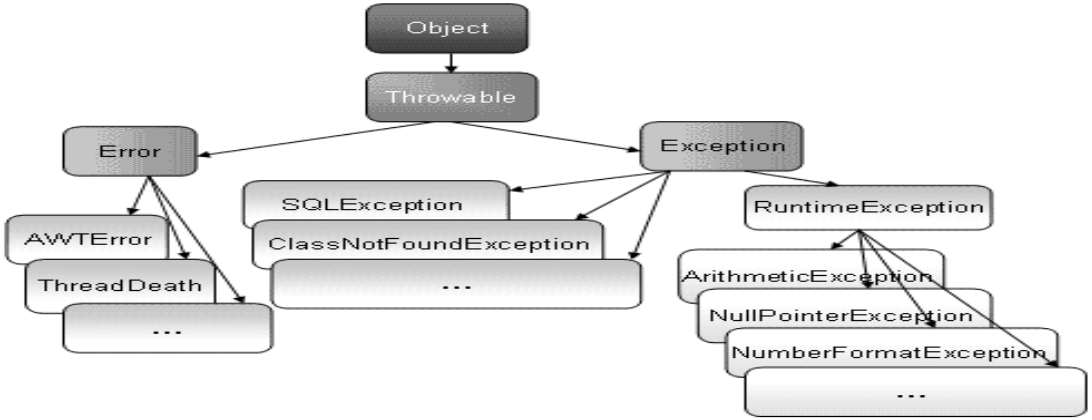
```
java.lang.ClassNotFoundException: com.mxp.test.Test
        at java.net.URLClassLoader$1.run(URLClassLoader.java:366)
        at java.net.URLClassLoader$1.run(URLClassLoader.java:355)
        at java.security.AccessController.doPrivileged(Native Method)
        at java.net.URLClassLoader.findClass(URLClassLoader.java:354)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:423)
        at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:308)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:356)
        at java.lang.Class.forName0(Native Method)
        at java.lang.Class.forName(Class.java:188)
        at com.mxp.iweb02.Test.main(Test.java:7)
```

java 通过面向对象的方法处理异常。在一个方法的运行过程中如果出现了异常，这个方法就会产生代表该异常的一个对象，把它交给运行时系统，运行时系统寻找相应的代码来处理这一异常。其中，生成异常对象，并把它交个运行时系统的过程称为抛出（throw）。运行时系统在方法的调用栈中查找，直到找到能处理该异常的对象的过程称为捕获（catch）。

java 异常处理的基础：

Java 异常处理机制采用一个统一和相对简单的抛出和处理错误的机制。如果一个方法本身能引发异常，当所调用的方法出现异常时，调用者可以捕获异常使之得到处理；也可以回避异常，这时异常将在调用的堆栈中向下传递，直到被处理。

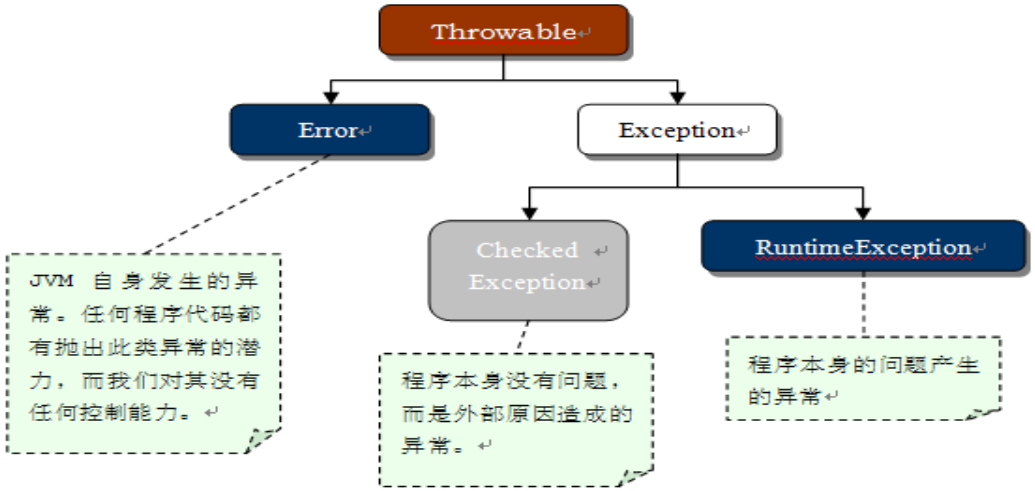
异常的体系结构：



Error：用于 Java 运行时系统来显示与运行时系统本身有关的错误 ；

Throwable：所有异常类型都是内置类 Throwable 的子类 ；

Exception：用于用户程序可能捕获的异常，也是用来创建用户异常类型子类的类。



java 的异常处理机制：

把各种不同类型的异常情况进行分类，用 Java 类来表示异常情况，这种类被称为异常类。把异常情况表示成异常类，可以充分发挥类的可扩展和可重用的优势。

异常流程的代码和正常流程的代码分离，提高了程序的可读性，简化了程序的结构。可以灵活的处理异常，如果当前方法有能力处理异常，就捕获并处理它，否则只需抛出异常，由方法调用者来处理它。

1. try{} ctach{}语句块处理异常

```
public class Test {

    public void say(){
        try {//try块包含了可能发生异常的代码
            System.out.println(10/0);
```

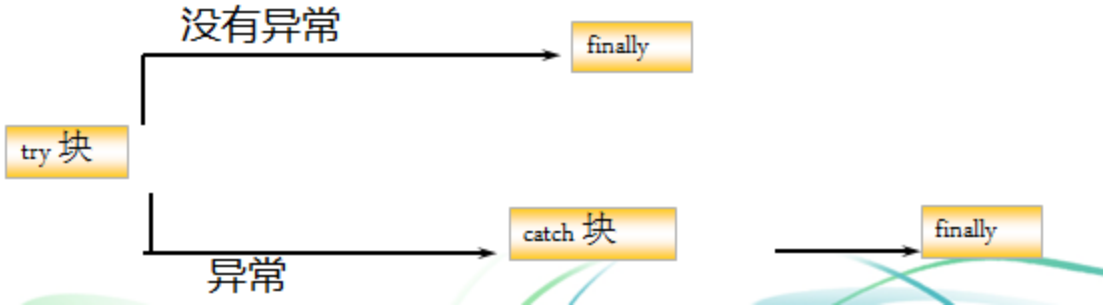
```
        } catch (ArithmeticException e) {//具体的异常类型或者它的父类，如Exception
            e.printStackTrace();//输出堆栈的跟踪信息
        }
    }

    public static void main(String[] args) {

        new Test().say();

    }

}
```

2. 多个 catch 块

```
public class Test {

    public void say() {
        int[] arr = { 1 };
        /*
         * 当单独执行下面一条输出语句时，会找到对应的异常类的catch块，并输出异常信息
         * 当两条语句一起执行的时候，System.out.println(arr[1])将不会被执行
         */
        try {
            System.out.println(10 / 0);// 会发生除数为0异常
            System.out.println(arr[1]);// 会发生数组越界异常
        } catch (ArithmeticException e) {
            System.out.println("除数为0");
```

```
            e.printStackTrace();
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("数组越界");
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {

        new Test().say();

    }

}
```

3. finally 块：确保了异常发生时所有的清楚工作都可以的到完成，无论是否出现异常，finally 块中的语句都会被执行。




```
public class Test {

    public int say(int i) {
```

```
            try {
                System.out.println(10 / i);
            } catch (Exception e) {
                e.printStackTrace();
                return 1;
            } finally {

            }
            return 2;
        }

        public static void main(String[] args) {

            System.out.println("say方法：" + new Test().say(1));

        }

}
```

分析：

1. 将 `return 2` 语句放在方法的最后一行，finally 块之外，发生异常时候 say 方法的返回值为 1，没有异常的时候 say 方法的返回值为 2；
2. 将 `return 2` 语句放在 finally 块中，无论是否发生异常，say 方法返回值都时 2；

常见异常及用途：

异常	说明
RuntimeException	java.lang包中多数异常的基类
ArithmeticException	算术错误，如除以0
IllegalArgumentException	方法收到非法参数
ArrayIndexOutOfBoundsException	数组下标出界
NullPointerException	试图访问null对象引用
SecurityException	试图违反安全性
ClassNotFoundException	不能加载请求的类

异常	说明
AWTException	AWT 中的异常
IOException	I/O 异常的根类
FileNotFoundException	不能找到文件
EOFException	文件结束
IllegalAccessException	对类的访问被拒绝
NoSuchMethodException	请求的方法不存在
InterruptedException	线程中断

二、 throw 语句

- 异常是通过关键字 throw 抛出，程序可以用 throw 语句引发明确的异常。如：

```
try {
        if(flag=null)    {
          throw new NullPointerException();
```

```
        }
    }
```

- **throw 语句的操作数一定是 Throwable 类类型或 Throwable 子类类型的一个对象。**

三、 throws 语句

如果一个方法不想处理异常，可以通过 throws 语句将异常抛向上级调用方法。

```
public class Test {

    //类中ClassNotFoundException异常类型不处理，而由后来的调用者进行处理，可以throws多个异常类型，用逗号隔开
    public void say() throws ClassNotFoundException {
        Class.forName("con.mxp.test.Test");
    }

    public static void main(String[] args) {
        Test t = new Test();

        try {
            t.say();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }

}
```

四、 运行时异常

RuntimeException 类及其子类都称为运行时异常，这种异常的特点是 Java 编译器不会检查它，也就是说，当程序中可能出现这类异常，即使没有用 try-catch 语句捕获它，也没有用 throws 子句声明抛出它，也会编译通过。例如当以下 divide()方法的参数 b 为 0，执行"a/b"操作时会出现 ArrithmeticException 异常，它属于运行时异常，Java 编译器不会检查它。

```
public class Test {

    public void divide(int a, int b) {
        System.out.println(a / b);//当b为0的时候发生异常
    }

}
```

区分运行时异常和检查异常：

- 运行时异常表示无法让程序恢复运行的异常，导致这种异常的原因通常是由于执行了错误操作。一旦出现了错误操作，建议终止程序，因此 Java 编译器不检查这种异常。
- 检查异常表示程序可以处理的异常，如果抛出异常的方法本身不能处理它，那么方法调用者应该去处理它，从而使程序恢复运行，不至于终止程序。
- 如果一个方法可能出现受检查异常，要么用 try-catch 语句捕获，要么用 throws 子句声明将它抛出，否则会导致编译错误。

运行时异常示例：

```
public class Test {

    public void say(){
        int[] arr = {1,2,3,4};
        for (int i = 0; i <= arr.length; i++) {
            System.out.println(arr[i]);//会发生异常
        }
    }
}
```

需要修改程序代码：

```
public class Test {

    public void say(){
        int[] arr = {1,2,3,4};
        for (int i = 0; i < arr.length; i++) {
            System.out.println(arr[i]);//会发生异常
        }
    }
}
```

检查异常：

```
public class Test {

    //该方法可能发生ClassNotFoundException异常
    public void say()throws ClassNotFoundException{
```

```
    }

    public void tell(){
        say();//会发生编译错误
    }
}
```

修改程序代码：

```
public class Test {

    //该方法可能发生ClassNotFoundException异常
    public void say()throws ClassNotFoundException{

    }

    public void tell(){
        try {
            say();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

throw 和 throws 的区别：

1）throw 是语句抛出一个异常；throws 是方法抛出一个异常； throws 可以单独使用，但 throw 不能；

2）throws 出现在方法函数头；而 throw 出现在函数体；

3）throws 表示出现异常的一种可能性，并不一定会发生这些异常；throw 则是抛出了异常，执行 throw 则一定抛出了某种异常；

4）两者都是消极处理异常的方式（这里的消极并不是说这种方式不好），只是抛出或者可能抛出异常，但是不会由函数去处理异常，真正的处理异常由函数的上层调用处理。

五、 自定义异常

1.内置异常不可能始终足以捕获所有错误，因此需要用户自定义的异常类；

2.用户自定义的异常类应为 Exception 类（或者 Exception 类的子类）的子类；

3.创建的任何用户自定义的异常类都可以获得 Throwable 类定义的方法 ；

```
public class Test {

    public int[] arr(int size) {
        int[] arr = null;
        if(size<0){
            try {
                throw new ArraySizeException();//发生异常
            } catch (ArraySizeException e) {
                e.printStackTrace();
            }
        }
        arr=new int[size];
```

```
        return arr;
    }

}
```