异常

异常

认识异常

接下来，我们学习一下异常，学习异常有利于我们处理程序中可能出现的问题。什么是异常？

我们阅读下面的代码，通过这段代码来认识异常。 我们调用一个方法时，经常一不小心就出异常了，然后在控制台打印一些异常信息。其实打印的这些异常信息，就叫做异常。

例如：

```
1  public class ExceptionTest01 {
2
3      public static void main(String[] args) {
4          System.out.println(Integer.valueOf("abc"));
5      }
6  }
7
```

运行之后会出现如下的页面：

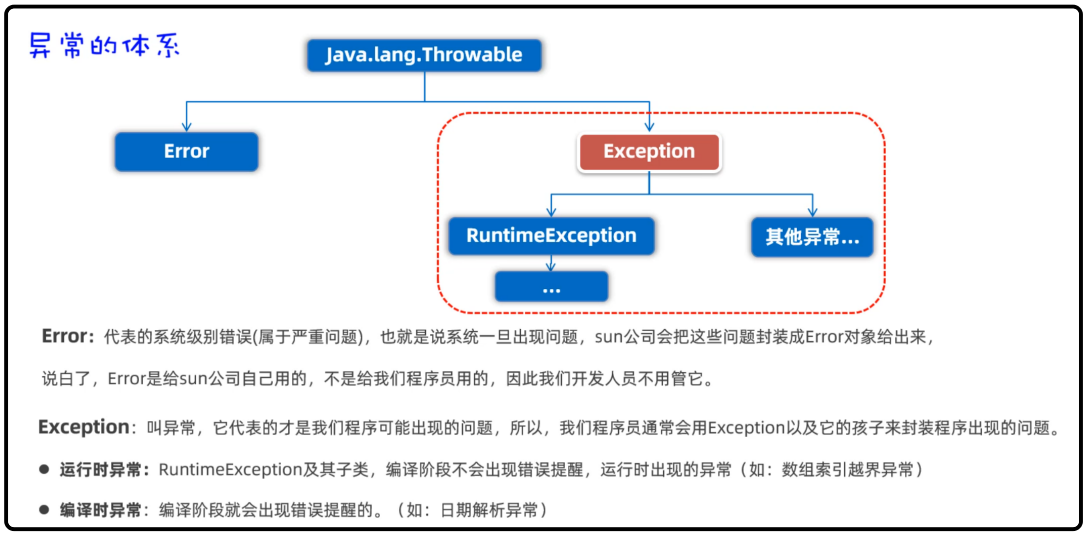
```
Exception in thread "main" java.lang.NumberFormatException Create breakpoint : For input string: "abc"
    at java.base/java.lang.NumberFormatException.forInputString(NumberFormatException.java:67)
    at java.base/java.lang.Integer.parseInt(Integer.java:665)
    at java.base/java.lang.Integer.valueOf(Integer.java:992)
```

那肯定有同学就纳闷了，我写代码天天出异常，我知道这是异常啊！我们这里学习异常，其实是为了告诉你异常是怎么产生的？只有你知道异常是如何产生的，才能避免出现异常。以及产生异常之后如何处理。

因为在设计这个方法的时候，在方法中对调用者传递的参数进行校验，如果校验数据不合法，就会将异常信息封装成一个对象并抛出，JVM接收到异常对象之后，就把异常打印了。

```
1  public static Integer valueOf(String s) throws NumberFormatException {
2      return Integer.valueOf(parseInt(s, 10));
3  }
```

因为写代码时经常会出现问题，Java的设计者们早就为我们写好了很多个异常类，来描述不同场景下的问题。而有些类是有共性的所以就有了异常的继承体系



Error：代表的系统级别错误(属于严重问题)，也就是说系统一旦出现问题，sun公司会把这些问题封装成Error对象给出来，

说白了，Error是给sun公司自己用的，不是给我们程序员用的，因此我们开发人员不用管它。

Exception：叫异常，它代表的才是我们程序可能出现的问题，所以，我们程序员通常会用Exception以及它的孩子来封装程序出现的问题。

- **运行时异常**：RuntimeException及其子类，编译阶段不会出现错误提醒，运行时出现的异常（如：数组索引越界异常）
- **编译时异常**：编译阶段就会出现错误提醒的。（如：日期解析异常）

所有异常类都是Throwable类的子类，它派生出两个子类，`Error`类和`Exception`类。

（1）`Error`类：表示程序无法恢复的严重错误或者恢复起来比较麻烦的错误，例如内存溢出、动态链接失败、虚拟机错误等。应用程序不应该主动抛出这种类型的错误，通常由虚拟机自动抛出。如果出现这种错误，最好的处理方式是让程序安全退出。在进行程序设计时，我们更应关注Exception类。

（2）`Exception`类：由Java应用程序抛出和处理的非严重错误，例如文件未找到、网络连接问题、算术错误（如除以零）、数组越界、加载不存在的类、对空对象进行操作、类型转换异常等。Exception类的不同子类对应不同类型的异常。Exception类又可分为两大类异常：

不受检异常：也称为unchecked异常，包括RuntimeException及其所有子类。对这类异常并不要求强制进行处理，例如算术异常ArithmeticException等。本章将重点讲解不受检异常。

受检异常：也称为checked异常，指除了不受检异常外，其他继承自Exception类的异常。对这类异常要求在代码中进行显式处理。

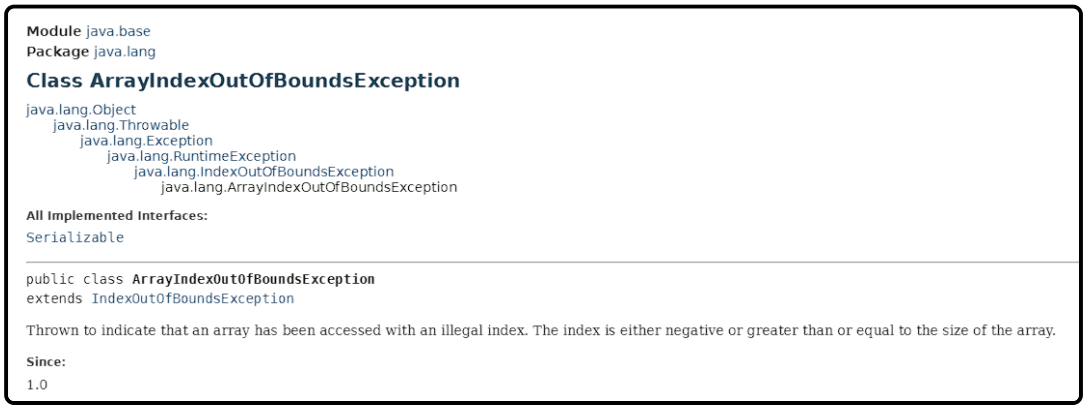
Java提供了多种异常类，下表列举了一些常见的异常类及其用途。在当前阶段，你只需要初步了解这些异常类即可。在以后的编程中，可以根据系统报告的异常信息，分析异常类型来判断程序出现了什么问题。

异常类名	异常分类	说明
Exception	设计时异常	异常层次结构的根类。
IOException	设计时异常	IO异常的根类，属于非运行时异常。
FileNotFoundException	设计时异常	文件操作时，找不到文件。属于非运行时异常。
RuntimeException	运行时异常	运行时异常的根类，RuntimeException及其子类，不要求必须处理。
ArithmeticException	运行时异常	算术运算异常，比如：除数为零，属于运行时异常。
IllegalArgumentException	运行时异常	方法接收到非法参数，属于运行时异常。
ArrayIndexOutOfBoundsException	运行时异常	数组越界访问异常，属于运行时异常。
NullPointerException	运行时异常	尝试访问null对象的成员时发生的空指针异常，属于运行时异常 。
ArrayStoreException	运行时异常	数据存储异常，写数组操作时，对象或数据类型不兼容
ClassCastException	运行时异常	类型转换异常
IllegalThreadStateException	运行时异常	试图非法改变线程状态，例如试图启动一已经运行的线程
NumberFormatException	运行时异常	数据格式异常，试图把一字符串非法转换成数值

> 先来演示一个运行时异常产生

```java
int[] arr = {11,22,33};
// 5是一个不存在的索引，所以此时产生ArrayIndexOutOfBoundsExcpetion
System.out.println(arr[5]);
```

下图是API中对ArrayIndexOutOfBoundsExcpetion类的继承体系，以及告诉我们它在什么情况下产生。



Module java.base
Package java.lang

Class ArrayIndexOutOfBoundsException

java.lang.Object
  java.lang.Throwable
    java.lang.Exception
      java.lang.RuntimeException
        java.lang.IndexOutOfBoundsException
          java.lang.ArrayIndexOutOfBoundsException

All Implemented Interfaces:

Serializable

public class **ArrayIndexOutOfBoundsException**
extends IndexOutOfBoundsException

Thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of the array.

Since:

1.0

> 再来演示一个编译时异常

我们在调用SimpleDateFormat对象的parse方法时，要求传递的参数必须和指定的日期格式一致，否则就会出现异常。 Java比较贴心，它为了更加强烈的提醒方法的调用者，设计了编译时异常，它把异常的提醒提前了，你调用方法是否真的有问题，只要可能有问题就给你报出异常提示（红色波浪线）。

编译时异常的目的：意思就是告诉你，你小子注意了！！，这里小心点容易出错，仔细检查一下

有人说，我检查过了，我确认我的代码没问题，为了让它不报错，继续将代码写下去。我们这里有两种解决方案。

- 第一种：使用throws在方法上声明，意思就是告诉下一个调用者，这里面可能有异常啊，你调用时注意一下。

```java
/**
 * 目标：认识异常。
 */
public class ExceptionTest1 {
    public static void main(String[] args) throws ParseException{
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        Date d = sdf.parse("2028-11-11 10:24");
        System.out.println(d);
    }
}
```

- 第二种：使用try...catch语句块异常进行处理。

```java
public class ExceptionTest1 {
    public static void main(String[] args) throws ParseException{
        try {
            SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
            Date d = sdf.parse("2028-11-11 10:24");
            System.out.println(d);
        } catch (ParseException e) {
            e.printStackTrace();
        }
    }
}
```

好了，关于什么是异常，我们就先认识到这里。

异常处理

通过前面两小节的学习，我们已经认识了什么是异常，以及异常的产生过程。接下来就该了解出现异常该如何处理了。

Java的异常处理机制类似于人们对可能发生的意外情况进行预先处理的方式。在程序执行过程中，如果发生了异常，程序会按照预定的处理方式对异常进行处理。处理完异常之后，程序会继续执行。如果异常没有被处理，程序将终止运行。

Java的异常处理机制依靠以下5个关键字：try、catch、finally、throw、throws。这些关键字提供了两种异常处理方式。

- 用try、catch、finally来捕获和处理异常。
  - try块中包含可能会抛出异常的代码。
  - catch块用于捕获并处理指定类型的异常。
  - finally块中的代码无论是否发生异常都会被执行，通常用于释放资源或清理操作。
- 使用throw、throws来抛出异常。
  - throw关键字用于手动抛出异常对象。
  - throws关键字用于在方法声明中指定可能抛出的异常类型，表示该方法可能会抛出该类型的异常，由调用者来处理。

捕获异常

使用try-catch处理异常

Java中提供了try-catch结构进行异常捕获和处理，把可能出现异常的代码放在try语句块中，并使用catch语句块捕获异常。

使用try-catch捕获并处理异常。

```
public class Main {
    public static void main(String[] args) {
        try {
            int i = 1, j = 0, res;
            System.out.println("begin");
            res = i / j;
            System.out.println("end");

        } catch (Exception e) {
            System.out.println("caught");
            e.printStackTrace();
        }
        System.out.println("over");
    }
}
```

运行结果没有输出“end”，输出内容的顺序是“begin”，“caught”，“over”。

try-catch语句块首先执行try语句块中的语句，这时可能会出现以下3种情况：

1. 如果try语句块中的所有语句正常执行完毕，没有发生异常，那么catch语句块中的所有语句都将被忽略。
2. 如果try语句块在执行过程中发生异常，并且这个异常与catch语句块中声明的异常类型匹配，那么try语句块中剩下的代码都将被忽略，而相应的catch语句块将会被执行。匹配是指catch所处理的异常类型与try块所生成的异常类型完全一致或是它的父类。

3. 如果try语句块在执行过程中发生异常，而抛出的异常在catch语句块中没有被声明，那么程序立即终止运行，程序被强迫退出。

4. catch语句块中可以加入用户自定义处理信息，也可以调用异常对象的方法输出异常信息，常用的方法如下：

`void prinStackTrace()`：输出异常的堆栈信息。堆栈信息包括程序运行到当前类的执行流程，它将输出从方法调用处到异常抛出处的方法调用的栈序列。

`String getMessage()`：返回异常信息描述字符串，该字符串描述了异常产生的原因，是`printStackTrace()`输出信息的一部分。

使用try-catch-finally处理异常

try-catch-finally语句块组合使用时，无论try块中是否发生异常，finally语句块中的代码总能被执行。

使用try-catch-finally捕获并处理异常。

```
public class Main {
    public static void main(String[] args) {
        try {
            int i = 1, j = 0, res;
            System.out.println("begin");
            res = i / j;
            System.out.println("end");
        } catch (ArithmeticException e) {
            System.out.println("caught");
            e.printStackTrace();
        } finally {
            System.out.println("finally");
        }
        System.out.println("over");
    }
}
```

输出结果输出了“finally”。如果将j的值改为不是0，再次运行该程序，也会输出“finally”。

try-catch-finally语句块执行流程大致分为如下两种情况。

1. 如果try语句块中所有语句正常执行完毕，程序不会进入catch语句块执行，但是finally语句块会被执行。

2. 如果try语句块在执行过程中发生异常，程序会进入到catch语句块捕获异常， finally语句块也会被执行。

3. try-catch-finally结构中try语句块是必须存在的，catch、finally语句块为可选，但两者至少出现其中之一。

> 即使在catch语句块中存在return语句，finally语句块中的语句也会执行。发生异常时的执行顺序是，先执行catch语句块中return之前的语句，再执行finally语句块中的语句，最后执行catch语句块中的return语句退出。

finally语句块中语句不执行的唯一情况是在异常处理代码中执行了`System.exit(1)`，退出Java虚拟机。

```
public class Test {
    public static void main(String[] args) {
        System.out.println(method()); // 2
    }

    public static int method() {
        int i = 1;
        try {
            i++;
            System.out.println("try block, i = " + i); // i = 2
            return i;
        } catch (Exception e) {
            i++;
            System.out.println("catch block i = " + i);
            return i;
```

```
        } finally {
            i = 10;
            System.out.println("finally block i = " + i); // i = 10
        }
    }
}
```

```
public class Test {
    public static void main(String[] args) {
        System.out.println(method()); // 10
    }

    public static int method() {
        int i = 1;
        try {
            i++;
            System.out.println("try block, i = " + i); // i = 2
            return i;
        } catch (Exception e) {
            i++;
            System.out.println("catch block i = " + i);
            return i;
        } finally {
            i = 10;
            System.out.println("finally block i = " + i); // i = 10
            return i;
        }
    }
}
```

使用多重catch处理异常

当一段代码可能引发多种类型的异常时，可以在一个try语句块后面跟随多个catch语句块，分别处理不同类型的异常。

> catch语句块的排列顺序必须是从子类到父类，最后一个一般是Exception类。这是因为在异常处理中，catch语句块会按照从上到下的顺序进行匹配，系统会检测每个catch语句块处理的异常类型，并执行第一个与异常类型匹配的catch语句块。如果将父类异常放在前面，子类异常的catch语句块将永远不会被执行，因为父类异常的catch语句块已经处理了异常。

一旦系统执行了与异常类型匹配的catch语句块，并执行其中的一条catch语句后，其后的catch语句块将被忽略，程序将继续执行紧随catch语句块的代码。

总结起来，异常处理的顺序和匹配原则非常重要。子类异常应该放在前面，父类异常应该放在后面，以确保正确的异常处理顺序。

多重catch语句块。

```
public class Main {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        try {
            System.out.println("计算开始");
            int i, j, res;
            System.out.println("请输入被除数");
            i = input.nextInt();
            System.out.println("请输入除数");
            j = input.nextInt();
            res = i / j;
            System.out.println(i + "/" + j + "=" + res);
            System.out.println("计算结束");
        } catch (InputMismatchException e) {
            System.out.println("除数和被除数必须都是整数");
```

```
        } catch (ArithmeticException e) {
            System.out.println("除数不能为零");
        } catch (Exception e) {
            System.out.println("其他异常" + e.getMessage());
        } finally {
            System.out.println("感谢使用本程序");
        }
        System.out.println("程序结束");
    }
}
```

抛出异常

使用throws声明抛出异常

try-catch-finally处理的是在一个方法内部发生的异常，在方法内部直接捕获并处理。如果在一个方法体内抛出了异常，并希望调用者能够及时地捕获异常，Java语言中通过关键字throws声明某个方法可能抛出的各种异常，以通知调用者。throws可以同时声明多个异常，之间用逗号隔开。

```
public void test() throws Exception {

}
```

使用throw抛出异常

除了系统自动抛出异常外，在编程过程中，有些问题是系统无法自动发现并解决的，如年龄不在正常范围之内，性别输入的不是“男”或“女”等，此时需要程序员而不是系统来自行抛出异常，把问题提交给调用者去解决。在Java语言中，可以使用throw关键字来自行抛出异常。

```
throw new Exception("message")
```

- 如果 throw 语句抛出的异常是 Checked 异常，则该 throw 语句要么处于 try 块里，显式捕获该异常，要么放在一个带 throws 声明抛出的方法中，即把该异常交给该方法的调用者处理；
- 如果 throw 语句抛出的异常是 Runtime 异常，则该语句无须放在 try 块里，也无须放在带 throws 声明抛出的方法中；程序既可以显式使用 try…catch来捕获并处理该异常，也可以完全不理会该异常，把该异常交给该方法调用者处理

自行抛出Runtime 异常比自行抛出Checked 异常的灵活性更好。同样，抛出 Checked 异常则可以让编译器提醒程序员必须处理该异常。

> 1. 作用不同：throw用于程序员自行产生并抛出异常，throws用于声明该方法内抛出了异常。
> 2. 使用位置不同：throw位于方法体内部，可以作为单独的语句使用；throws必须跟在方法参数列表的后面，不能单独使用。
> 3. 内容不同：throw抛出一个异常对象，只能是一个；throws后面跟异常类，可以跟多个。

自定义异常

经过刚才的学习已经认识了什么是异常了，但是无法为编码过程全部问题都提供异常类，如果企业自己的某种问题，想通过异常来表示，那就需要自己来定义异常类了。

我们通过一个实际场景，来给大家演示自定义异常。

> 需求：写一个saveAge(int age)方法，在方法中对参数age进行判断，如果age<0或者>=150就认为年龄不合法，如果年龄不合法，就给调用者抛出一个年龄非法异常。
>
> 分析：Java的API中是没有年龄非常这个异常的，所以我们可以自定义一个异常类，用来表示年龄非法异常，然后再方法中抛出自定义异常即可。

- 先写一个异常类AgeIllegalException（这是自己取的名字，名字取得很奈斯），继承

```
// 1、必须让这个类继承自Exception，才能成为一个编译时异常类。
public class AgeIllegalException extends Exception{
    public AgeIllegalException() {
    }

    public AgeIllegalException(String message) {
        super(message);
    }
}
```

- 再写一个测试类，在测试类中定义一个saveAge(int age)方法，对age判断如果年龄不在0~150之间，就抛出一个AgeIllegalException异常对象给调用者。

```
public class ExceptionTest2 {
    public static void main(String[] args) {
        // 需求：保存一个合法的年
        try {
            saveAge2(225);
            System.out.println("saveAge2底层执行是成功的！");
        } catch (AgeIllegalException e) {
            e.printStackTrace();
            System.out.println("saveAge2底层执行是出现bug的！");
        }
    }

    //2、在方法中对age进行判断，不合法则抛出AgeIllegalException
    public static void saveAge(int age){
        if(age > 0 && age < 150){
            System.out.println("年龄被成功保存：  " + age);
        }else {
            // 用一个异常对象封装这个问题
            // throw 抛出去这个异常对象
            throw new AgeIllegalRuntimeException("/age is illegal, your age is " + age);
        }
    }
}
```

- 注意咯，自定义异常可能是编译时异常，也可以是运行时异常

```
1.如果自定义异常类继承Excpetion，则是编译时异常。
    特点：方法中抛出的是编译时异常，必须在方法上使用throws声明，强制调用者处理。

2.如果自定义异常类继承RuntimeException，则运行时异常。
    特点：方法中抛出的是运行时异常，不需要在方法上用throws声明。
```

异常链：有时候我们会捕获一个异常后再抛出另一个异常
顾名思义就是将异常发生的原因一个传一个串起来，即把底层的异常信息传给上层，这样逐层抛出。

定义testOne，testTwo，testThree方法，testTwo对testOne抛出的异常进行捕获，testThree对testTwo抛出的异常进行捕获：

```
```

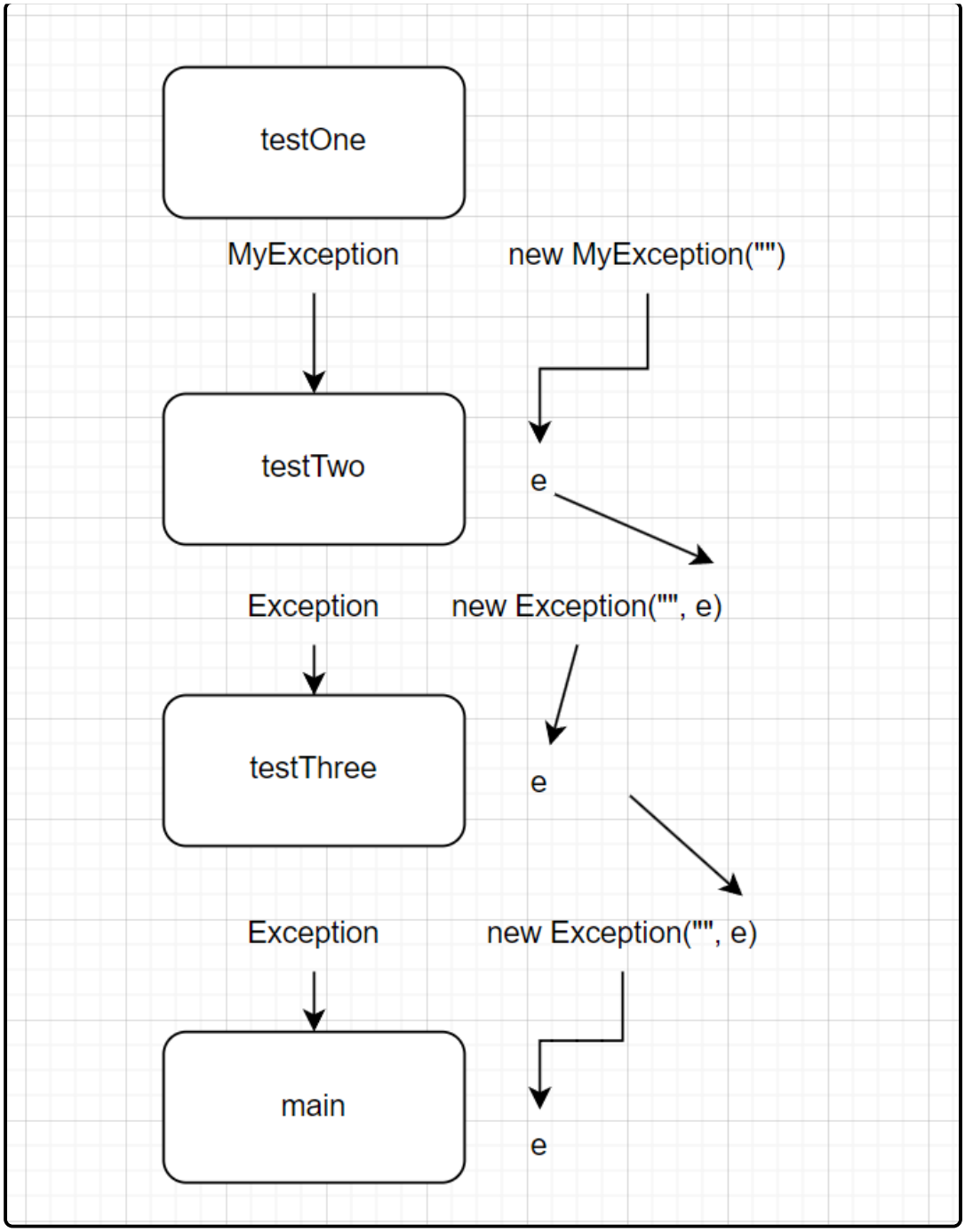
```java
public class TryDemoFive {
    public static void main(String[] args) {
        try {
            testThree();
        } catch (Exception e) {
            e.printStackTrace(); //
        }
    }

    public static void testOne() throws MyException {
        throw new MyException("我是一个异常");
    }

    public static void testTwo() throws Exception {
        try {
            testOne();
        } catch (Exception e) {
            throw new Exception("我是新产生的异常1");
        }
    }

    public static void testThree() throws Exception {
        try {
            testTwo();
        } catch (Exception e) {
            throw new Exception("我是新产生的异常2");
        }
    }
}
```

- 直接在新抛出的异常中添加原来的异常信息

```java
throw new Exception("我是新产生的异常1", e);
```

- 在要抛出的对象中使用`initCause()`方法，添加上一个产生异常的信息;`getCause()`可以获取当前异常对象的上一个异常对象

```java
Exception e2 = new Exception("我是新产生的异常2");
e2.initCause(e);
throw e2;
```



好了，到此我们关于异常的知识就全部学习完了。