

# guisu , 程序人生。 逆水行舟，不进则退。


能干的人解决问题。智慧的人避开问题(A clever person solves a problem. A wise person avoids it)

目录视图


摘要视图

RSS 订阅

个人资料



真实的归宿

访问: 3259937次  
积分: 24002  
等级:   
排名: 第158名

原创: 212篇 转载: 2篇  
译文: 0篇 评论: 1068条

文章分类

操作系统 (5)

Linux (22)

MySQL (12)

PHP (42)

架构 (5)

PHP内核 (11)

技术人生 (8)

数据结构与算法 (30)

云计算hadoop (25)

网络知识 (7)

c/c++ (23)

memcache (5)

HipHop (2)

计算机原理 (4)

Java (7)

socket网络编程 (8)

设计模式 (26)

AOP (2)

重构 (11)

重构与模式 (1)

搜索引擎Search Engine (15)

大数据处理 (12)

HTML5 (1)

Android (1)

webserver (3)

NOSQL (7)

NOSQL Mongo (0)

分布式 (1)

数据结构与算法 xi (0)

协议 (1)

信息论的熵 (0)

Bitbucket 让 pull request变得更强大，可即刻提升团队代码质量 云计算行业圆桌论坛 前端精品课程免费看，写课评赢心动大礼！

深入理解java异常处理机制

标签: file php ubuntu 服务器 任务

2011-01-20 18:44 101507人阅读 评论(90) 收藏 举报

分类: Java (6)

版权声明：本文为博主原创文章，未经博主允许不得转载。

目录(?) [+]

1. 引子

try...catch...finally恐怕是大家再熟悉不过的语句了，而且感觉用起来也是很简单，逻辑上似乎也是很容易理解。不过，我亲自体验的“教训”告诉我，这个东西可不是想象中的那么简单、听话。不信？那你看看下面的代码，“猜猜”它执行后的结果会是什么？不要往后看答案、也不许执行代码看真正答案哦。如果你的答案是正确，那么这篇文章你就不用浪费时间看啦。

[java] view plain copy print ?

```
<span style="background-color: rgb(255, 255, 255);">package Test;

public class TestException {
    public TestException() {
    }

    boolean testEx() throws Exception {
        boolean ret = true;
        try {
            ret = testEx1();
        } catch (Exception e) {
            System.out.println("testEx, catch exception");
            ret = false;
            throw e;
        } finally {
            System.out.println("testEx, finally; return value=" + ret);
            return ret;
        }
    }

    boolean testEx1() throws Exception {
        boolean ret = true;
        try {
            ret = testEx2();
            if (!ret) {
                return false;
            }
            System.out.println("testEx1, at the end of try");
            return ret;
        } catch (Exception e) {
            System.out.println("testEx1, catch exception");
            ret = false;
            throw e;
        } finally {
            System.out.println("testEx1, finally; return value=" + ret);
            return ret;
        }
    }

    boolean testEx2() throws Exception {
        boolean ret = true;
        try {
            int b = 12;
            int c;
            for (int i = 2; i >= -2; i--) {
                c = b / i;
                System.out.println("i=" + i);
            }
        }
    }
}
```

关于php的libevent扩展的应用 (0)

libevent简单介绍 (0)

SOA (0)

文章存档

2015年12月 (2)

2015年10月 (4)

2015年05月 (2)

2015年04月 (1)

2015年01月 (2)

展开

阅读排行

八大排序算法 (258047)

mysql 多表联合查询效率 (139051)

深入理解java异常处理机 (101426)

socket阻塞与非阻塞，同 (100773)

hbase安装配置（整合到 (88978)

设计模式（十八）策略模 (80931)

Nginx工作原理和优化、 (76581)

Hadoop Hive sql语法详 (75129)

Linux的SOCKET编程详 (73468)

Java输入输出流 (69625)

评论排行

深入理解java异常处理机 (90)

八大排序算法 (85)

socket阻塞与非阻塞，同 (52)

硬盘的读写原理 (40)

设计模式（十八）策略模 (39)

设计模式（一）工厂模式 (30)

海量数据处理算法—Bit (26)

PHP SOCKET编程 (23)

UML图中类之间的关系 (23)

设计模式（十七）状态模 (21)

推荐文章

\* HDFS如何检测并删除多余副本块

\* Project Perfect让Swift在服务器端跑起来—让Perfect更Rails (五)

\* 数据库性能优化之SQL语句优化

\* R语言 非标准化求值（Non-standard evaluation, NSE）

\* 机器学习系列(7)\_机器学习路线图（附资料）

\* 网络性能评价方法

最新评论

设计模式（十七）状态模式State syp19821016: @xuchuangfeng: 要区分状态和行为，行为是基于状态上的行为，增加pause状态，只是增加状...

```
        return true;
    } catch (Exception e) {
        System.out.println("testEx2, catch exception");
        ret = false;
        throw e;
    } finally {
        System.out.println("testEx2, finally; return value=" + ret);
        return ret;
    }
}

public static void main(String[] args) {
    TestException testException1 = new TestException();
    try {
        testException1.testEx();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

你的答案是什么？是下面的答案吗？

```
i=2
i=1
testEx2. catch exception
testEx2. finally: return value=false
testEx1. catch exception
testEx1. finally: return value=false
testEx. catch exception
testEx. finally: return value=false
```

如果你的答案真的如上面所说，那么你错啦。^ ^，那就建议你仔细看一看这篇文章或者拿上面的代码按各种不同的情况修改、执行、测试，你会发现有很多事情不是原来想象中的那么简单的。现在公布正确答案：

```
i=2
i=1
testEx2. catch exception
testEx2. finally: return value=false
testEx1. finally: return value=false
testEx, finally; return value=false
```

注意说明：

finally 语句块不应该出现 应该出现return。上面的return ret最好是其他语句来处理相关逻辑。

## 2.JAVA异常

异常指不期而至的各种状况，如：文件找不到、网络连接失败、非法参数等。异常是一个事件，它发生在程序运行期间，干扰了正常的指令流程。Java通过API中Throwable类的众多子类描述各种不同的异常。因而，Java异常都是对象，是Throwable子类的实例，描述了出现在一段编码中的 错误条件。当条件生成时，错误将引发异常。

Java异常类层次结构图：

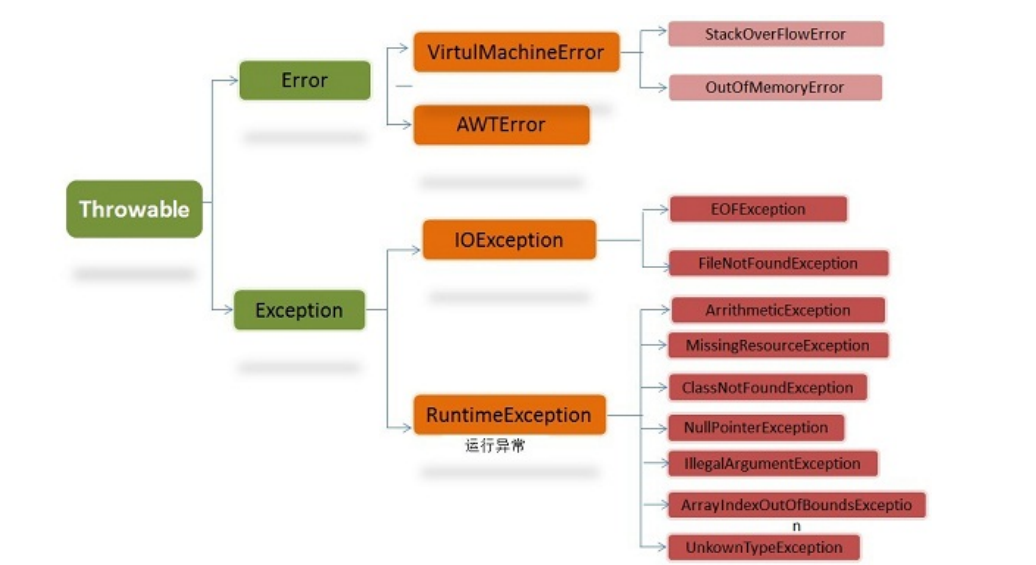


图1 Java异常类层次结构图

在Java中，所有的异常都有一个共同的祖先Throwable（可抛出）。Throwable指定代码中可用异常传播机制通过Java应用程序传输的任何问题的共性。

**Throwable：**有两个重要的子类：Exception（异常）和Error（错误），二者都是Java异常处理的重要子类，各自都包含大量子类。

**Error（错误）：**是程序无法处理的错误，表示运行应用程序中较严重问题。大多数错误与代码编写者执行的操作

设计模式 (十七) 状态模式State  
syp19821016: 要区分状态和行  
为, 行为是基于状态上的行为,  
增加pause状态, 只是增加状  
态, 你要考虑的是电梯的ope...

设计模式 (一) 工厂模式Factory  
debugbird: 写的很好!!!

八大排序算法  
行者小朱: 楼主总结的不错

设计模式 (九) 外观模式Facade  
12期-王金龙: 学习了

HTTP详解(2)-请求、响应、缓存  
lvxin1204: 受教了

网络互联参考模型 (详解)  
lvxin1204: 谢谢教授!

高性能Mysql主从架构的复制原理  
何静媛: 好厉害

搜索引擎-倒排索引基础知识  
七月这天:  
http://blog.csdn.net/malefactor/artic

搜索引擎-倒排索引基础知识  
七月这天: @hguisu:少了一张  
图, "出现位置信息"没有。

友情链接

图灵机器人: 聊天api的最佳选择

作无关, 而表示代码运行时 JVM (Java 虚拟机) 出现的问题。例如, Java虚拟机运行错误 (Virtual MachineError), 当 JVM 不再有继续执行操作所需的内存资源时, 将出现 OutOfMemoryError。这些异常发生时, Java虚拟机 (JVM) 一般会选择线程终止。

。这些错误表示故障发生于虚拟机自身、或者发生在虚拟机试图执行应用时, 如Java虚拟机运行错误 (Virtual MachineError)、类定义错误 (NoClassDefFoundError) 等。这些错误是不可查的, 因为它们在应用程序的控制和处理能力之外, 而且绝大多数是程序运行时不允许出现的状况。对于设计合理的应用程序来说, 即使确实发生了错误, 本质上也不应该试图去处理它所引起的异常状况。在 Java中, 错误通过Error的子类描述。

**Exception (异常):**是程序本身可以处理的异常。

Exception 类有一个重要的子类 RuntimeException。RuntimeException 类及其子类表示“JVM 常用操作”引发的错误。例如, 若试图使用空值对象引用、除数为零或数组越界, 则分别引发运行时异常 (NullPointerException、ArithmeticException) 和 ArrayIndexOutOfBoundsException。

**注意: 异常和错误的区别: 异常能被程序本身可以处理, 错误是无法处理。**

通常, Java的异常(包括Exception和Error)分为**可查的异常 (checked exceptions)**和**不可查的异常 (unchecked exceptions)**。

**可查异常 (编译器要求必须处置的异常):** 正确的程序在运行中, 很容易出现的、情理可容的异常状况。可查异常虽然是异常状况, 但在一定程度上它的发生是可以预计的, 而且一旦发生这种异常状况, 就必须采取某种方式进行处理。

除了RuntimeException及其子类以外, 其他的Exception类及其子类都属于可查异常。这种异常的特点是Java编译器会检查它, 也就是说, 当程序中可能出现这类异常, 要么用try-catch语句捕获它, 要么用throws子句声明抛出它, 否则编译不会通过。

**不可查异常(编译器不要求强制处置的异常):**包括运行时异常 (RuntimeException与其子类) 和错误 (Error)。

Exception 这种异常分两大类**运行时异常**和**非运行时异常(编译异常)**。程序中应当尽可能去处理这些异常。

**运行时异常:** 都是RuntimeException类及其子类异常, 如NullPointerException(空指针异常)、IndexOutOfBoundsException(下标越界异常)等, **这些异常是不检查异常**, 程序中可以选择不处理。这些异常一般是由程序逻辑错误引起的, 程序应该从逻辑角度尽可能避免这类异常的发生。

运行时异常的特点是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。

**非运行时异常 (编译异常):** 是RuntimeException以外的异常, 类型上都属于Exception类及其子类。从程序语法角度讲是必须进行处理的异常, 如果不处理, 程序就不能编译通过。如IOException、SQLException等以及用户自定义的Exception异常, 一般情况下不自定义**检查异常**。

## 4.处理异常机制

在 Java 应用程序中, 异常处理机制为: **抛出异常, 捕捉异常**。

**抛出异常:** 当一个方法出现错误引发异常时, 方法创建异常对象并交付运行时系统, 异常对象中包含了异常类型和异常出现时的程序状态等异常信息。运行时系统负责寻找处置异常的代码并执行。

**捕获异常:** 在方法抛出异常之后, 运行时系统将转为寻找合适的异常处理器 (exception handler)。潜在的异常处理器是异常发生时依次存留在调用栈中的方法的集合。当异常处理器所能处理的异常类型与方法抛出的异常类型相符时, 即为合适 的异常处理器。运行时系统从发生异常的方法开始, 依次回查调用栈中的方法, 直至找到含有合适异常处理器的方法并执行。当运行时系统遍历调用栈而未找到合适 的异常处理器, 则运行时系统终止。同时, 意味着Java程序的终止。

对于运行时异常、错误或可查异常, Java技术所要求的异常处理方式有所不同。

由于运行时异常的不可查性, 为了更合理、更容易地实现应用程序, Java规定, 运行时异常将由Java运行时系统自动抛出, 允许应用程序忽略运行时异常。

对于方法运行中可能出现的Error, 当运行方法不欲捕捉时, Java允许该方法不做任何抛出声明。因为, 大多数Error异常属于永远不能被允许发生的状况, 也属于合理的应用程序不该捕捉的异常。

对于所有的可查异常, Java规定: 一个方法必须捕捉, 或者声明抛出方法之外。也就是说, 当一个方法选择不捕捉可查异常时, 它必须声明将抛出异常。

能够捕捉异常的方法，需要提供相符类型的异常处理器。所捕捉的异常，可能是由于自身语句所引发并抛出的异常，也可能是由某个调用的方法或者Java运行时系统等抛出的异常。也就是说，**一个方法所能捕捉的异常，一定是Java代码在某处所抛出的异常。简单地说，异常总是先被抛出，后被捕捉的。**

任何Java代码都可以抛出异常，如：自己编写的代码、来自Java开发环境包中代码，或者Java运行时系统。无论是谁，都可以通过Java的throw语句抛出异常。

从方法中抛出的任何异常都必须使用throws子句。

捕捉异常通过try-catch语句或者try-catch-finally语句实现。

总体来说，Java规定：对于可查异常必须捕捉、或者声明抛出。允许忽略不可查的RuntimeException和Error。

#### 4.1 捕获异常：try、catch 和 finally

##### 1. try-catch语句

在Java中，异常通过try-catch语句捕获。其一般语法形式为：

```
[java] view plain copy print ?
try {
    // 可能会发生异常的代码
} catch (Type1 id1){
    // 捕获并处置try抛出的异常类型Type1
}
catch (Type2 id2){
    // 捕获并处置try抛出的异常类型Type2
}
```

关键词try后的一对大括号将一块可能发生异常的代码包起来，称为监控区域。Java方法在运行过程中出现异常，则创建异常对象。将异常抛出监控区域之外，由Java运行时系统试图寻找匹配的catch子句以捕获异常。若有匹配的catch子句，则运行其异常处理代码，try-catch语句结束。

匹配的原则是：如果抛出的异常对象属于catch子句的异常类，或者属于该异常类的子类，则认为生成的异常对象与catch块捕获的异常类型相匹配。

##### 例1 捕捉throw语句抛出的“除数为0”异常。

```
[java] view plain copy print ?
public class TestException {
    public static void main(String[] args) {
        int a = 6;
        int b = 0;
        try { // try监控区域

            if (b == 0) throw new ArithmeticException(); // 通过throw语句抛出异常
            System.out.println("a/b的值是: " + a / b);
        }
        catch (ArithmeticException e) { // catch捕捉异常
            System.out.println("程序出现异常，变量b不能为0。");
        }
        System.out.println("程序正常结束。");
    }
}
```

运行结果：程序出现异常，变量b不能为0。

程序正常结束。

例1 在try监控区域通过if语句进行判断，当“除数为0”的错误条件成立时引发ArithmeticException异常，创建ArithmeticException异常对象，并由throw语句将异常抛给Java运行时系统，由系统寻找匹配的异常处理器catch并运行相应异常处理代码，打印输出“程序出现异常，变量b不能为0。”try-catch语句结束，继续程序流程。

事实上，“除数为0”等ArithmeticException，是RuntimeException的子类。而运行时异常将由运行时系统自动抛出，不需要使用throw语句。

##### 例2 捕捉运行时系统自动抛出“除数为0”引发的ArithmeticException异常。

```
[java] view plain copy print ?
public static void main(String[] args) {
    int a = 6;
    int b = 0;
    try {
        System.out.println("a/b的值是: " + a / b);
    } catch (ArithmeticException e) {
```

```

        System.out.println("程序出现异常, 变量b不能为0。");
    }
    System.out.println("程序正常结束。");
}
}

```

运行结果：程序出现异常，变量b不能为0。

程序正常结束。

例2 中的语句：

System.out.println("a/b的值是: " + a/b);

在运行中出现“除数为0”错误，引发ArithmeticException异常。运行时系统创建异常对象并抛出监控区域，转而匹配合适的异常处理器catch，并执行相应的异常处理代码。

由于检查运行时异常的代价远大于捕捉异常所带来的益处，运行时异常不可查。Java编译器允许忽略运行时异常，一个方法可以既不捕捉，也不声明抛出运行时异常。

例3 不捕捉、也不声明抛出运行时异常。

```

[java] view plain copy print ?
public class TestException {
    public static void main(String[] args) {
        int a, b;
        a = 6;
        b = 0; // 除数b 的值为0
        System.out.println(a / b);
    }
}

```

运行结果：

Exception in thread "main" java.lang.ArithmeticException: / by zero  
at Test.TestException.main(TestException.java:8)

例4 程序可能存在除数为0异常和数组下标越界异常。

```

[java] view plain copy print ?
public class TestException {
    public static void main(String[] args) {
        int[] intArray = new int[3];
        try {
            for (int i = 0; i <= intArray.length; i++) {
                intArray[i] = i;
                System.out.println("intArray[" + i + "] = " + intArray[i]);
                System.out.println("intArray[" + i + "] 模 " + (i - 2) + " 的值: "
                    + intArray[i] % (i - 2));
            }
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("intArray数组下标越界异常。");
        } catch (ArithmeticException e) {
            System.out.println("除数为0异常。");
        }
        System.out.println("程序正常结束。");
    }
}

```

运行结果：

intArray[0] = 0

intArray[0]模 -2的值: 0

intArray[1] = 1

intArray[1]模 -1的值: 0

intArray[2] = 2

除数为0异常。

程序正常结束。

例4 程序可能会出现除数为0异常，还可能会出现数组下标越界异常。程序运行过程中ArithmeticException异常类型是先行匹配的，因此执行相匹配的catch语句：

```
[java] view plain copy print ?
catch (ArithmeticException e){
    System.out.println("除数为0异常。");
}
```

需要注意的是，一旦某个catch捕获到匹配的异常类型，将进入异常处理代码。一经处理结束，就意味着整个try-catch语句结束。其他的catch子句不再有匹配和捕获异常类型的机会。

Java通过异常类描述异常类型，异常类的层次结构如图1所示。对于有多个catch子句的异常程序而言，应该尽量将捕获底层异常类的catch子句放在前面，同时尽量将捕获相对高层的异常类的catch子句放在后面。否则，捕获底层异常类的catch子句将可能会被屏蔽。

RuntimeException异常类包括运行时各种常见的异常，ArithmeticException类和ArrayIndexOutOfBoundsException类都是它的子类。因此，RuntimeException异常类的catch子句应该放在最后面，否则可能会屏蔽其后的特定异常处理或引起编译错误。

## 2. try-catch-finally语句

try-catch语句还可以包括第三部分，就是finally子句。它表示无论是否出现异常，都应当执行的内容。try-catch-finally语句的一般语法形式为：

```
[java] view plain copy print ?
try {
    // 可能会发生异常的程序代码
} catch (Type1 id1) {
    // 捕获并处理try抛出的异常类型Type1
} catch (Type2 id2) {
    // 捕获并处理try抛出的异常类型Type2
} finally {
    // 无论是否发生异常，都将执行的语句块
}
```

例5 带finally子句的异常处理程序。

```
[java] view plain copy print ?
public class TestException {
    public static void main(String args[]) {
        int i = 0;
        String greetings[] = { " Hello world !", " Hello World !! ",
                                " HELLO WORLD !!!" };
        while (i < 4) {
            try {
                // 特别注意循环控制变量i的设计，避免造成无限循环
                System.out.println(greetings[i++]);
            } catch (ArrayIndexOutOfBoundsException e) {
                System.out.println("数组下标越界异常");
            } finally {
                System.out.println("-----");
            }
        }
    }
}
```

运行结果：

Hello world !

-----

Hello World !!

-----

HELLO WORLD !!!

-----

数组下标越界异常

-----

在例5中，请特别注意try子句中语句块的设计，如果设计为如下，将会出现死循环。如果设计为：



```
[java] view plain copy print ?
try {
    System.out.println (greetings[i]); i++;
}
```

小结:

**try** 块: 用于捕获异常。其后可接零个或多个**catch**块, 如果没有**catch**块, 则必须跟一个**finally**块。

**catch** 块: 用于处理try捕获到的异常。

**finally** 块: 无论是否捕获或处理异常, **finally**块里的语句都会被执行。当在**try**块或**catch**块中遇到**return**语句时, **finally**语句块将在方法返回之前被执行。在以下4种特殊情况下, **finally**块不会被执行:

- 1) 在**finally**语句块中发生了异常。
- 2) 在前面的代码中用了**System.exit()**退出程序。
- 3) 程序所在的线程死亡。
- 4) 关闭CPU。

### 3. try-catch-finally 规则(异常处理语句的语法规则) :

- 1) 必须在 **try** 之后添加 **catch** 或 **finally** 块。**try** 块后可同时接 **catch** 和 **finally** 块, 但至少有一个块。
- 2) 必须遵循块顺序: 若代码同时使用 **catch** 和 **finally** 块, 则必须将 **catch** 块放在 **try** 块之后。
- 3) **catch** 块与相应的异常类的类型相关。
- 4) 一个 **try** 块可能有多个 **catch** 块。若如此, 则执行第一个匹配块。即Java虚拟机会把实际抛出的异常对象依次和各个**catch**代码块声明的异常类型匹配, 如果异常对象为某个异常类型或其子类的实例, 就执行这个**catch**代码块, 不会再执行其他的 **catch**代码块
- 5) 可嵌套 **try-catch-finally** 结构。
- 6) 在 **try-catch-finally** 结构中, 可重新抛出异常。
- 7) 除了下列情况, 总将执行 **finally** 做为结束: **JVM 过早终止 (调用 System.exit(int))**; 在 **finally** 块中抛出一个未处理的异常; 计算机断电、失火、或遭遇病毒攻击。

### 4. try、catch、finally语句块的执行顺序:

- 1) 当try没有捕获到异常时: **try**语句块中的语句逐一被执行, 程序将跳过**catch**语句块, 执行**finally**语句块和其后的语句;
- 2) 当try捕获到异常, **catch**语句块里没有处理此异常的情况: 当**try**语句块里的某条语句出现异常时, 而没有处理此异常的**catch**语句块时, 此异常将会抛给JVM处理, **finally**语句块里的语句还是会被执行, 但**finally**语句块后的语句不会被执行;
- 3) 当try捕获到异常, **catch**语句块里有处理此异常的情况: 在**try**语句块中是按照顺序来执行的, 当执行到某一条语句出现异常时, 程序将跳到**catch**语句块, 并与**catch**语句块逐一匹配, 找到与之对应的处理程序, 其他的**catch**语句块将不会被执行, 而**try**语句块中, 出现异常之后的语句也不会被执行, **catch**语句块执行完后, 执行**finally**语句块里的语句, 最后执行**finally**语句块后的语句;

图示**try**、**catch**、**finally**语句块的执行:

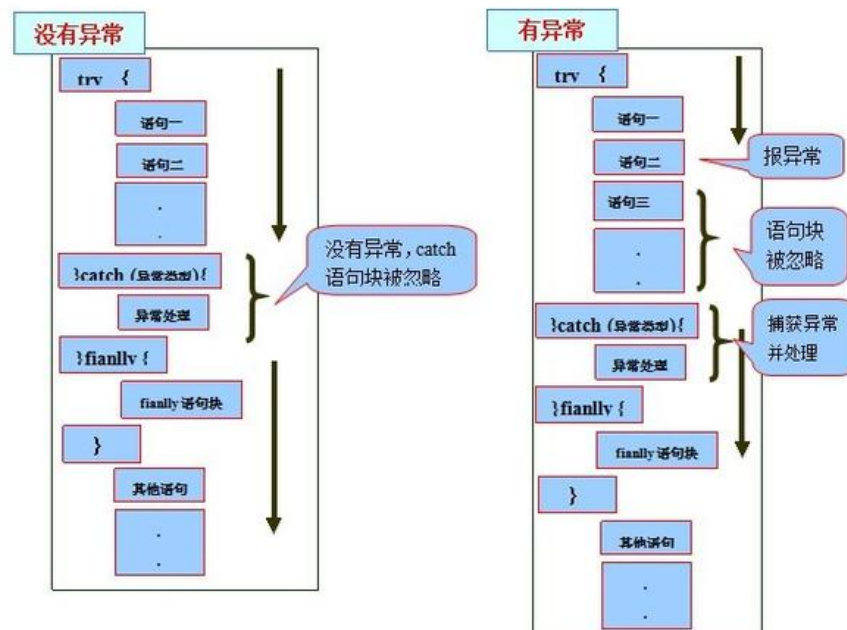


图2 图示try、catch、finally语句块的执行

## 4.2 抛出异常

任何Java代码都可以抛出异常，如：自己编写的代码、来自Java开发环境包中代码，或者Java运行时系统。无论是谁，都可以通过Java的throw语句抛出异常。从方法中抛出的任何异常都必须使用throws子句。

### 1. throws抛出异常

如果一个方法可能会出现异常，但没有能力处理这种异常，可以在方法声明处用throws子句来声明抛出异常。例如汽车在运行时可能会出现故障，汽车本身没办法处理这个故障，那就让开车的人来处理。

throws语句用在方法定义时声明该方法要抛出的异常类型，如果抛出的是Exception异常类型，则该方法被声明为抛出所有的异常。多个异常可使用逗号分割。throws语句的语法格式为：

```
[java] view plain copy print ?
methodname throws Exception1,Exception2,...,ExceptionN
{
}
```

方法名后的throws Exception1,Exception2,...,ExceptionN 为声明要抛出的异常列表。当方法抛出异常列表的异常时，方法将不对这些类型及其子类类型的异常作处理，而抛向调用该方法的方法，由他去处理。例如：

```
[java] view plain copy print ?
import java.lang.Exception;
public class TestException {
    static void pop() throws NegativeArraySizeException {
        // 定义方法并抛出NegativeArraySizeException异常
        int[] arr = new int[-3]; // 创建数组
    }

    public static void main(String[] args) { // 主方法
        try { // try语句处理异常信息
            pop(); // 调用pop()方法
        } catch (NegativeArraySizeException e) {
            System.out.println("pop()方法抛出的异常"); // 输出异常信息
        }
    }
}
```

使用throws关键字将异常抛给调用者后，如果调用者不想处理该异常，可以继续向上抛出，但最终要有能够处理该异常的调用者。

pop方法没有处理异常NegativeArraySizeException，而是由main函数来处理。

Throws抛出异常的规则：

1) 如果是不可查异常（unchecked exception），即Error、RuntimeException或它们的子类，那么可以不使用throws关键字来声明要抛出的异常，编译仍能顺利通过，但在运行时会被系统抛出。

2) 必须声明方法可抛出的任何可查异常（checked exception）。即如果一个方法可能出现受可查异常，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

3) 仅当抛出了异常，该方法的调用者才必须处理或者重新抛出该异常。当方法的调用者无力处理该异常的时候，应该继续抛出，而不是囫圇吞枣。

4) 调用方法必须遵循任何可查异常的处理和声明规则。若覆盖一个方法，则不能声明与覆盖方法不同的异常。声明的任何异常必须是被覆盖方法所声明异常的同类或子类。

例如：

```
[java] view plain copy print ?
void method1() throws IOException{} //合法
//编译错误，必须捕获或声明抛出IOException
void method2(){
    method1();
}

//合法，声明抛出IOException
void method3()throws IOException {
    method1();
}

//合法，声明抛出Exception，IOException是Exception的子类
void method4()throws Exception {
    method1();
}
```



```

}

//合法，捕获IOException
void method5(){
    try{
        method1();
    }catch(IOException e){...}
}

//编译错误，必须捕获或声明抛出Exception
void method6(){
    try{
        method1();
    }catch(IOException e){throw new Exception();}
}

//合法，声明抛出Exception
void method7()throws Exception{
    try{
        method1();
    }catch(IOException e){throw new Exception();}
}

```

判断一个方法可能会出现异常的依据如下：

- 1) 方法中有throw语句。例如，以上method7()方法的catch代码块有throw语句。
- 2) 调用了其他方法，其他方法用throws子句声明抛出某种异常。例如，method3()方法调用了method1()方法，method1()方法声明抛出IOException，因此，在method3()方法中可能会出现IOException。

## 2. 使用throw抛出异常

throw总是出现在函数体中，用来抛出一个Throwable类型的异常。程序会在throw语句后立即终止，它后面的语句执行不到，然后在包含它的所有try块中（可能在上层调用函数中）从里向外寻找含有与其匹配的catch子句的try块。

我们知道，异常是异常类的实例对象，我们可以创建异常类的实例对象通过throw语句抛出。该语句的语法格式为：

```
throw new exceptionname;
```

例如抛出一个IOException类的异常对象：

```
throw new IOException;
```

要注意的是，throw 抛出的只能是可抛出类Throwable 或者其子类的实例对象。下面的操作是错误的：

```
throw new String("exception");
```

这是因为String 不是Throwable 类的子类。

如果抛出了检查异常，则还应该在方法头部声明方法可能抛出的异常类型。该方法的调用者也必须检查处理抛出的异常。

如果所有方法都层层上抛获取的异常，最终JVM会进行处理，处理也很简单，就是打印异常消息和堆栈信息。如果抛出的是Error或RuntimeException，则该方法的调用者可选择处理该异常。

```

[java] view plain copy print ?
package Test;
import java.lang.Exception;
public class TestException {
    static int quotient(int x, int y) throws MyException { // 定义方法抛出异常
        if (y < 0) { // 判断参数是否小于0
            throw new MyException("除数不能为负数"); // 异常信息
        }
        return x/y; // 返回值
    }
    public static void main(String args[]) { // 主方法
        int a = 3;
        int b = 0;
        try { // try语句包含可能发生异常的语句
            int result = quotient(a, b); // 调用方法quotient()
        } catch (MyException e) { // 处理自定义异常
            System.out.println(e.getMessage()); // 输出异常信息
        } catch (ArithmeticException e) { // 处理ArithmeticException异常
            System.out.println("除数不能为0"); // 输出提示信息
        } catch (Exception e) { // 处理其他异常
            System.out.println("程序发生了其他的异常"); // 输出提示信息
        }
    }
}

class MyException extends Exception { // 创建自定义异常类
    String message; // 定义String类型变量
    public MyException(String ErrorMessage) { // 父类方法
        message = ErrorMessage;
    }

    public String getMessage() { // 覆盖getMessage()方法
        return message;
    }
}

```

## 4.3 异常链

1) 如果调用`quotient(3,-1)`，将发生`MyException`异常，程序调转到`catch (MyException e)`代码块中执行；

2) 如果调用`quotient(5,0)`，将会因“除数为0”错误引发`ArithmeticException`异常，属于运行时异常类，由Java运行时系统自动抛出。`quotient()`方法没有捕捉`ArithmeticException`异常，Java运行时系统将沿方法调用栈查到`main`方法，将抛出的异常上传至`quotient()`方法的调用者：

```
int result = quotient(a, b); // 调用方法quotient()
```

由于该语句在`try`监控区域内，因此传回的“除数为0”的`ArithmeticException`异常由Java运行时系统抛出，并匹配`catch`子句：

```
catch (ArithmeticException e) { // 处理ArithmeticException异常
    System.out.println("除数不能为0"); // 输出提示信息
}
```

处理结果是输出“除数不能为0”。Java这种向上传递异常信息的处理机制，形成异常链。

Java方法抛出的可查异常将依据调用栈、沿着方法调用的层次结构一直传递到具备处理能力的调用方法，最高层次到`main`方法为止。如果异常传递到`main`方法，而`main`不具备处理能力，也没有通过`throws`声明抛出该异常，将可能出现编译错误。

3) 如还有其他异常发生，将使用`catch (Exception e)`捕捉异常。由于`Exception`是所有异常类的父类，如果将`catch (Exception e)`代码块放在其他两个代码块的前面，后面的代码块将永远得不到执行，就没有什么意义了，所以`catch`语句的顺序不可掉换。

#### 4.4 Throwable类中的常用方法

注意：`catch`关键字后面括号中的`Exception`类型的参数`e`。`Exception`就是`try`代码块传递给`catch`代码块的变量类型，`e`就是变量名。`catch`代码块中语句`"e.getMessage();"`用于输出错误性质。通常异常处理常用3个函数来获取异常的有关信息：

`getCause()`：返回抛出异常的原因。如果 `cause` 不存在或未知，则返回 `null`。

`getMessage()`：返回异常的消息信息。

`printStackTrace()`：对象的堆栈跟踪输出至错误输出流，作为字段 `System.err` 的值。

有时为了简单会忽略掉`catch`语句后的代码，这样`try-catch`语句就成了一种摆设，一旦程序在运行过程中出现了异常，就会忽略处理异常，而错误发生的原因很难查找。

## 5.Java常见异常

在Java中提供了一些异常用来描述经常发生的错误，对于这些异常，有的需要程序员进行捕获处理或声明抛出，有的是由Java虚拟机自动进行捕获处理。Java中常见的异常类：

### 1. runtimeException子类：

1、`java.lang.ArrayIndexOutOfBoundsException`

数组索引越界异常。当对数组的索引值为负数或大于等于数组大小时抛出。

2、`java.lang.ArithmeticException`

算术条件异常。譬如：整数除零等。

3、`java.lang.NullPointerException`

空指针异常。当应用试图在要求使用对象的地方使用了`null`时，抛出该异常。譬如：调用`null`对象的实例方法、访问`null`对象的属性、计算`null`对象的长度、使用`throw`语句抛出`null`等等

4、`java.lang.ClassNotFoundException`

找不到类异常。当应用试图根据字符串形式的类名构造类，而在遍历`CLASSPATH`之后找不到对应名称的`class`文件时，抛出该异常。

5、`java.lang.NegativeArraySizeException` 数组长度为负异常

6、`java.lang.ArrayStoreException` 数组中包含不兼容的值抛出的异常

7、`java.lang.SecurityException` 安全性异常

8、`java.lang.IllegalArgumentException` 非法参数异常

2. IOException

- IOException: 操作输入流和输出流时可能出现的异常。
- EOFException      文件已结束异常
- FileNotFoundException      文件未找到异常

3. 其他

- ClassCastException      类型转换异常类
- ArrayStoreException      数组中包含不兼容的值抛出的异常
- SQLException      操作数据库异常类
- NoSuchFieldException      字段未找到异常
- NoSuchMethodException      方法未找到抛出的异常
- NumberFormatException      字符串转换为数字抛出的异常
- StringIndexOutOfBoundsException      字符串索引超出范围抛出的异常
- IllegalAccessException      不允许访问某类异常
- InstantiationException      当应用程序试图使用Class类中的newInstance() 方法创建一个类的实例，而指定的类对象无法被实例化时，抛出该异常

6.自定义异常

使用Java内置的异常类可以描述在编程时出现的大部分异常情况。除此之外，用户还可以自定义异常。用户自定义异常类，只需继承Exception类即可。

在程序中使用自定义异常类，大体可分为以下几个步骤。

(1) 创建自定义异常类。

(2) 在方法中通过throw关键字抛出异常对象。

(3) 如果在当前抛出异常的方法中处理异常，可以使用try-catch语句捕获并处理；否则在方法的声明处通过throws关键字指明要抛出给方法调用者的异常，继续进行下一步操作。

(4) 在出现异常方法的调用者中捕获并处理异常。

在上面的“使用throw抛出异常”例子已经提到了。

顶 踩

97 4

上一篇 [Linux 内存管理](#)

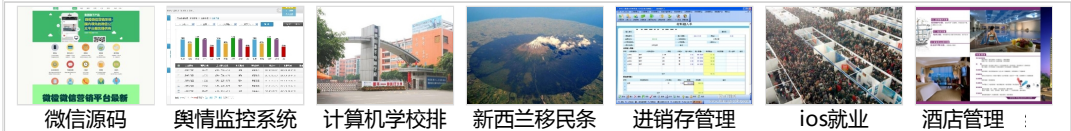
下一篇 [Memcache存储大数据的问题](#)

我的同类文章

Java（6）					
• <a href="#">Android必备的Java知识点</a>	2012-11-27	阅读 7869	• <a href="#">Java中的Map List Set等集合</a>	2012-06-08	阅读 12497
• <a href="#">UML图中类之间的关系:依赖、关联、聚合、组合</a>	2012-06-07	阅读 23418	• <a href="#">Java线程</a>	2012-04-23	阅读 8877
• <a href="#">Java输入输出流</a>	2012-04-01	阅读 69524	• <a href="#">深入理解java嵌套类和内部类</a>	2012-06-08	阅读 6434

猜你在找

- [Java基础语法之基础知识-Java基础视频\\_深入浅出精华#](#)
- [Java基础核心技术：基本语法\(day01-day04\)](#)
- [Java基础核心技术：面向对象编程\(day05-day07\)](#)
- [Java基础核心技术：Java反射机制\(day19-day20\)](#)
- [深入浅出Java的反射](#)



查看评论

74楼 [青蛙的世界](#) 2016-02-14 22:15发表



多个嘴，刚查阅好多博客才理解的，不一定完全正确：  
1.throw和return都可以使方法退出，而且可以互相覆盖  
2.finally里如果有return会把catch里面throw出来的异常覆盖掉  
3.程序运行关于栈和堆的问题，大家最好弄懂，无论那个编程语言都会涉及到，c里面叫指针，更高级语言里面叫引用，不懂这些有很多问题好难懂。了解这个问题的时候不要因为别人举例的语言不是java就抛弃，其实高级语言基础语法都差不多，不是蛮难看懂但是思想好重要

73楼 [q904672375](#) 2016-01-25 16:37发表



好啊 非常详实 让我发现了平时根本注意不到的一些问题~~

72楼 [FLY\\_1993](#) 2016-01-04 11:49发表



受教了！

71楼 [键盘乱舞](#) 2015-12-28 13:22发表



```
public static void main(String[] args) {  
    System.out.println(test());  
}
```

```
public static String test(){  
    String name = "123";  
    try {  
        throw new Exception();  
        //return name;  
    } catch (Exception e) {  
        return name;  
    } finally {  
        name = "456";  
    }  
}
```

请问这个为什么输出是 123，不是在try或者catch中遇到return时，首先执行finally吗

Re: [Rainnnbow](#) 2016-01-07 16:18发表



```
回复dy5623405: public int inc(){  
    int x;  
    try {  
        x = 1;  
        return x;  
    } catch (Exception e) {  
        x = 2;  
        return x;  
    } finally {  
        x = 3;  
        // System.out.println("x" + x);  
        // return x;  
    }  
}
```

这是《深入理解Java虚拟机-JVM高级特性与最佳实践》第二版书中的例子（P187~P188）。出现这种情况的原因是：在没有出线异常的情况下，先执行了x=1;然后执行return x;时，首先是将x的一个副本保存在本地变量表中，执行return之前必须执行finally中的操作：x=3;将x的值设置为了3，但是return时是将本地变量表中保存的x的那个副本拿出来放到栈顶返回。故没出异常时，返回值为1；出Exception异常或其子类异常时，返回值是2；如果出现非Exception异常，则执行完x=3之后，抛出异常，没有返回值。

Re: [Rainnnbow](#) 2016-01-07 16:27发表



回复Rainnnbow：对于这种异常时的执行情况，可以通过javap -verbose命令输出方法反编译后的字节码与异常表，从字节码层面上就能清晰的看出执行过程了。

Re: [Trinity\\_b](#) 2015-12-29 16:02发表




```
回复dy5623405: static StringBuffer test0(){  
    StringBuffer name = new StringBuffer("123");
```


```
try {
    throw new Exception();
} catch (Exception e) {
    System.out.println("catch name: " + name.hashCode());
    return name;
}finally{
    name.append("456");
    name = new StringBuffer("abc");
    System.out.println("finally name: " + name.hashCode());
}
```

只是我的理解。返回的不是副本，而是内存地址。  
"当在try块或catch块中遇到return语句时，finally语句块将在方法返回之前被执行" catch的return已经优先决定了要返回的内存，对于基本类型string其实每次都相当于一个new，每次的赋值都会新生成一个内存地址。

Re: Rainnnbow 2016-01-07 16:21发表

 回复Trinity\_b: 你的这种理解是错误的，返回的就是副本。

Re: stellari 2016-02-09 16:22发表


 回复Rainnnbow: 返回的确实是副本，但是这里最好解释清楚是“引用变量（在栈上）的副本”，而并不会创建“被引用对象（在堆上）的副本”。《深入》一书的例子用在这里未必合适，因为它用的变量都是primitive类型，没有显示出“引用”和“对象”的关系。@Trinity\_b的理解其实也没错，只是没表述好。他说的“优先决定”其实就是“保存在栈上待返回的引用变量副本不会在finally中被更改的意思”。

Re: Rainnnbow 2016-01-07 16:14发表


```
回复Trinity_b: public int inc(){
    int x;
    try {
        x = 1;
        return x;
    } catch (Exception e) {
        x = 2;
        return x;
    } finally{
        x = 3;
        // System.out.println("x: " + x);
        // return x;
    }
}
```

这是《深入理解Java虚拟机-JVM高级特性与最佳实践》第二版书中的例子（P187~P188）。出现这种情况的原因是：在没有出线异常的情况下，先执行了x=1;然后执行return x时，首先是将x的一个副本保存在本地变量表中，执行return之前必须执行finally中的操作：x=3;将x的值设置为了3，但是return时是将本地变量表中保存的x的那个副本拿出来放到栈顶返回。故没出异常时，返回值为1；出Exception异常或其子类异常时，返回值为2；如果出现非Exception异常，则执行完x=3之后，抛出异常，没有返回值。


70楼 xiaocui235 2015-11-02 20:46发表

 写的太详细了，简直比教材都好，看书看不懂，看了这篇文章，对Java异常有了更加深入的了解，大赞


Re: Rainnnbow 2016-01-07 13:24发表

 回复xiaocui235: Java编程思想书中的“通过异常处理错误”一章讲解的非常透彻。我看完那一章的所有内容现在回头又看了一遍这篇文章，感觉Bruce Eckel真的是大师。


69楼 Xiejian\_4986 2015-10-19 19:39发表

 4.4中第二个方法应该是getMessage()吧


68楼 Xiejian\_4986 2015-10-19 19:36发表

 写的很详细，还有实主！

67楼 1zk 2015-10-13 13:51发~~

 第一个例子，上面说fianlly中的return会吃掉catch中的异常，那第一个中的异常为什么没有被吃掉？请求楼主回答，这是怎么回事？

66楼 Kanth 2015-10-09 14:20发表

 好吧，看到一楼的回复我发现问题了！这样的写法也是够狗血的了...膜拜楼主

65楼 [Kanth](#) 2015-10-09 14:14发表



一个问题，我把代码拷到eclipse中运行后，结果就是你说的错误答案！求解释？为什么这个答案是错误的？

64楼 [qq\\_31811445](#) 2015-10-07 09:34发表



看到最后还是没有理解第一道题目正确答案为什么是这个啊？

有人能帮忙解答一下吗？整个文章真心不错，但是我看完了，还是没有能理解第一道题目，谢谢了

Re: [Kanth](#) 2015-10-09 14:21发表



回复[qq\\_31811445](#)：建议看下一楼大神的回复。。。文章总结的还可以！但是这个例子实在是让人无语。有谁会写出这样的代码呢...

63楼 [NARUTO](#) 2015-09-22 11:27发表



大神，你这第一个例子我不太懂呢，能不能就这个结果给个详细的解答，谢了~~

Re: [Kanth](#) 2015-10-09 14:23发表



回复[liushuai19910409](#)：看一楼回复就明白了...

62楼 [benjieyu](#) 2015-09-17 10:48发表



写的很好，很详细。

61楼 [十一期王娜](#) 2015-09-01 20:24发表



写的真棒啊

60楼 [liaohailin1989](#) 2015-08-26 16:33发表



finally类不能被继承。。。。。。然后就没有子类了么？

59楼 [liaohailin1989](#) 2015-08-26 16:32发表



写的很好很好哒

58楼 [-琥珀川-](#) 2015-08-26 14:47发表



finally中使用return是一种很不好的编程风格，它会覆盖掉所有的其它返回，并且吃掉catch中抛出的异常。可否详细解释一下

57楼 [minihuolu](#) 2015-08-21 17:49发表



一直没有弄明白异常，直到看了这篇文章。大赞！

56楼 [qq\\_20600425](#) 2015-08-20 16:41发表



这个 return 比较误导人啊。

55楼 [qq\\_28168611](#) 2015-08-19 23:28发表



受教了

54楼 [chenjinge7](#) 2015-08-18 12:16发表



学习了，明白了异常机制！

53楼 [x541534530](#) 2015-08-13 01:53发表



还差一点，为什么在finally块中使用return会suppress Exception：参考如下

Reports usage of return statements in finally block. Such return discards the exception being thrown and causes the whole method to complete normally, which is usually not the desired outcome. Break and continue statements, which break out of the finally block are also reported

Re: [1zk](#) 2015-10-13 13:47发表



上面说finally中的return会吃掉catch中的异常，为什么第一个catch中的异常没有被吃掉？请求楼主解答，这到底是怎么回事？

52楼 [micro\\_hz](#) 2015-08-10 15:51发表



讲的非常详细啊，支持。

51楼 [qq\\_30183991](#) 2015-07-28 16:44发表



写的很清楚，学习了。



50楼 庄子来了 2015-07-27 20:30发表



```
try {  
    System.out.println (greetings[i]); i++;  
}
```

请问各位大神，这里为什么会发生无限循环呢？

Re: 木杉 2015-08-08 18:37发表



回复uouop: 因为执行到print的时候，如果i=3，就数组越界了，直接进入catch块，不会执行i的自增，导致无法跳出while循环

49楼 明MIKEWOO 2015-07-25 22:00发表



高手！顶礼膜拜！异常处理机制，写的太棒了，谢谢，学习中.....

48楼 3冷水泡面 2015-07-24 20:52发表



引用“mc46790090”的评论:  
回复cangchen: 刚才翻了《Thinking in Java》书中提到...

嗯，只要抛出异常，throw后面的语句就不会被执行了。。

47楼 3冷水泡面 2015-07-24 20:50发表



引用“yongmi”的评论:  
finally中使用return是一种很不好的编程风格，它会覆盖掉所有的其它返回，并且吃掉catch...

有道理！

46楼 3冷水泡面 2015-07-24 20:46发表



牛人。。

45楼 dami2933 2015-07-23 15:28发表



写的真心不错，以前学java基础的时候看到Exception 就头疼，写了半年程序以后，再回头看你这篇帖子，讲的很详细，让人理解的很透彻。

44楼 一蓑烟雨\_bupt 2015-07-17 13:46发表



mark

43楼 FreMaN2011 2015-07-03 15:00发表



写的太好了!!!看了一下就懂了!谢谢楼主!!

42楼 Leader\_Bai 2015-06-09 10:29发表



我只想说，写的真的很好，很认真，对我万分有帮助，谢谢！！！！

41楼 tuspark2015 2015-05-30 22:43发表



为啥不去看看这篇文章呢？这篇文章讲解的更详细，而且格式也清晰：<http://swiftlet.net/archives/998>

40楼 jary0524 2015-05-28 22:04发表



写的真好，很全面，很详细！

39楼 chn-yang 2015-05-12 14:44发表



写得真好

38楼 chn-yang 2015-05-12 14:44发表



写得真好

37楼 闪电飞狐 2015-05-02 12:15发表



写的真心不错啊，感谢楼主

36楼 王家何先生 2015-04-23 09:42发表



不错，写的很好，感谢楼主

35楼 [Zempty9758](#) 2015-03-25 19:59发表



问题是我看到最后还是不知道何为正常处理异常？只是打印个异常栈错误信息就得了还是应该怎么做？

34楼 [liunian110110](#) 2015-03-05 13:31发表



文章写得很详细，也很好。那张图片有点问题中间的那个ArithmeticException，拼写错误，应该为ArithmeticException，多了个r。

33楼 [深蓝developer](#) 2015-01-23 00:26发表



高手！顶礼膜拜！等哪天跳槽了，一定到你们公司面试。

32楼 [风骚的裤脚](#) 2015-01-13 14:35发表



学习了

31楼 [ggbwqy242](#) 2015-01-09 23:12发表



zanzan

30楼 [binghejinjun](#) 2015-01-06 16:21发表



其实只要看了一楼的回复，答案就明了了，不过还是感谢楼主分享！

29楼 [Jaster\\_wisdom](#) 2014-12-28 17:14发表



哇哇，真的很详细，赞。

28楼 [l498948617](#) 2014-12-25 17:29发表



不错，学习了~

27楼 [yejiayanivy](#) 2014-12-04 11:00发表



好像有点错误哦，Exception下面的异常除了IOException和RuntimeException还有其他异常的，具体可以查API里面的Exception还有就是，Java异常类层次结构图中，ClassNotFoundException不是RuntimeException的吧，它是直接继承了Exception的啊

26楼 [allenhcool](#) 2014-11-23 16:51发表



mark to my mind

25楼 [lyzsy](#) 2014-11-12 17:06发表



写的太好了

24楼 [hanamingming](#) 2014-11-10 11:35发表



写的非常详细。收藏。

23楼 [whatsis](#) 2014-11-05 20:05发表



我是过来点赞的，写的好详细。

22楼 [cike8899](#) 2014-10-29 17:05发表



```
try{
    System.out.println (greetings[i]); i++;
}
```

为什么i永远都等于3呢，为什么不会增加呢？  
我是菜鸟，望大神指点！

Re: [qq\\_20149545](#) 2014-11-26 08:03发表



回复cike8899：没看 图2 图示try、catch、finally语句块的执行？  
System.out.println (greetings[i]);出现异常后直接执行catch的语句，i++;被跳过了。可以把i++;放到打印语句前面试试  
这篇文章写得很好

Re: [qq\\_20149545](#) 2014-11-26 07:35发表



回复cike8899：没看 图2 图示try、catch、finally语句块的执行？  
System.out.println (greetings[i]);出现异常后直接执行catch的语句，i++;被跳过了。可以把i++;放到打印语句前面试试  
这篇文章写得很好

21楼 [kchiu](#) 2014-10-23 10:10发表



您写的太好了 互勉！！

20楼 [awoshiwo](#) 2014-10-21 11:15发表



谢谢分享！mark是什么意思啊？

19楼 [陈丽娜](#) 2014-10-15 09:06发表



这也太详细了！

18楼 [等不到天亮Deng时光](#) 2014-10-12 20:13发表



楼主态度太认真了，可惜没看完，Mark一下

17楼 [陈梦洲](#) 2014-10-09 12:55发表



写的非常棒！这是我第一次看完文章后专门登录再评论，以后应该多多说出自己的感谢，IT人的分享精神，值得称赞！

16楼 [fomeiherz](#) 2014-10-06 21:25发表



写的很好，学习了。。。另外很喜欢你的头像

15楼 [学习园](#) 2014-08-26 21:21发表



mark

14楼 [mc46790090](#) 2014-08-21 20:28发表



引用“cangchen”的评论：

```
boolean testEx2() throws Exception {
    boo...
```

1楼已经回答了，**finally**中使用**return**会吃掉**catch**中抛出的异常。我也感觉博主写着写着把引子里的问题扔一边了。。。不知道下面的看客是不是看着看着就被博主给带歪了，我找了全文都没有找到引子中的问题所在，感谢1楼，我做了实验，的确是

Re: [真实的归宿](#) 2014-08-22 10:02发表



回复mc46790090：异常处理确实不应该出现**return**，谢谢。

13楼 [aronj旭](#) 2014-08-20 11:31发表



写得太棒了，，，非常实用详细，，大赞LZ

12楼 [just00play](#) 2014-08-14 16:37发表



学习了，thanks

11楼 [cangchen](#) 2014-07-31 11:04发表



```
boolean testEx2() throws Exception {
    boolean ret = true;
    try{
        int b = 12;
        int c;
        for (int i = 2; i >= -2; i--){
            c = b / i;
            System.out.println("i=" + i);
        }
        return true;
    } catch (Exception e) {
        System.out.println("testEx2, catch exception");
        ret = false;
        throw e;
    }
}
```

请问，虽然在该方法里捕获了该异常，但**throw e** 向上层抛出的**e**对象，上层不处理吗？

Re: [mc46790090](#) 2014-08-21 20:41发表



回复cangchen：刚才翻了《Thinking in Java》书中提到，**finally**语句块中如果重新抛出异常，会覆盖掉之前**try**块中捕获的异常，**finally**块中使用**return**会丢失**try**块中捕获的异常，当然**finally**块中，不能同时出现抛出异常和**return**语句

Re: [mc46790090](#) 2014-09-19 23:21发表



回复mc46790090：抱歉，之前的留言可能给大家造成误解“**finally**块中，不能同时出现抛出异常和**return**语句”

今晚又做了实验，`finally`语句块中可以同时抛出异常和`return`语句，只不过不可以将某个语句置为不可达：

例如

```
for(int i = 0; i < 4; i++){
    if(i == 2) throw new MyException("");
}
return;
}
```

也是可以的，同`try`块，抛出异常之后，下面的`return`语句就不会执行了，不知道博主博文里面的“在以下4种特殊情况下，`finally`块不会被执行：

1）在`finally`语句块中发生了异常。

”

这句话是不是表示`finally`不会被【完全】执行，运行时异常，在编译器是无法判断的。因此肯定是会执行的，只是执行会中断而已。再次感谢博主分享博文！

10楼 [sum\\_\\_mer](#) 2014-07-25 14:44发表



异常处理机制，写的太棒了，谢谢，学习中.....

9楼 [SelfMedicated](#) 2014-07-17 11:03发表



能干的人解决问题。智慧的人绕开问题(A clever person solves a problem. A wise person avoids it)

不是绕开吧？是避免让问题发生

8楼 [冲浪的水手](#) 2014-07-08 00:00发表



Fuck，写的太好了，膜拜!!!

7楼 [384410505](#) 2014-07-02 11:58发表



mark

6楼 [simonssphy](#) 2014-05-29 19:43发表



很好很好！

5楼 [Edward\\_Van](#) 2014-04-23 10:58发表



讲的非常详细和系统、很好的文章、

4楼 [Jesse\\_fishing](#) 2014-04-14 21:15发表



不错！

3楼 [ZTAngel](#) 2014-04-14 15:06发表



学习ing

2楼 [孤独涉世人](#) 2014-03-30 18:17发表



写的很好，学习了。

1楼 [yongmi](#) 2013-03-23 20:20发表



`finally`中使用`return`是一种很不好的编程风格，它会覆盖掉所有的其它返回，并且吃掉`catch`中抛出的异常。

您还没有登录,请[\[登录\]](#)或[\[注册\]](#)

\* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

全部主题   [Hadoop](#)   [AWS](#)   [移动游戏](#)   [Java](#)   [Android](#)   [iOS](#)   [Swift](#)   [智能硬件](#)   [Docker](#)   [OpenStack](#)  
[VPN](#)   [Spark](#)   [ERP](#)   [IE10](#)   [Eclipse](#)   [CRM](#)   [JavaScript](#)   [数据库](#)   [Ubuntu](#)   [NFC](#)   [WAP](#)   [jQuery](#)  
[BI](#)   [HTML5](#)   [Spring](#)   [Apache](#)   [.NET](#)   [API](#)   [HTML](#)   [SDK](#)   [IIS](#)   [Fedora](#)   [XML](#)   [LBS](#)   [Unity](#)  
[Splashtop](#)   [UML](#)   [components](#)   [Windows Mobile](#)   [Rails](#)   [QEMU](#)   [KDE](#)   [Cassandra](#)   [CloudStack](#)  
[FTC](#)   [coremail](#)   [OPhone](#)   [CouchBase](#)   [云计算](#)   [iOS6](#)   [Rackspace](#)   [Web App](#)   [SpringSide](#)   [Maemo](#)  
[Compuware](#)   [大数据](#)   [aptech](#)   [Perl](#)   [Tornado](#)   [Ruby](#)   [Hibernate](#)   [ThinkPHP](#)   [HBase](#)   [Pure](#)   [Solr](#)  
[Angular](#)   [Cloud Foundry](#)   [Redis](#)   [Scala](#)   [Django](#)   [Bootstrap](#)

