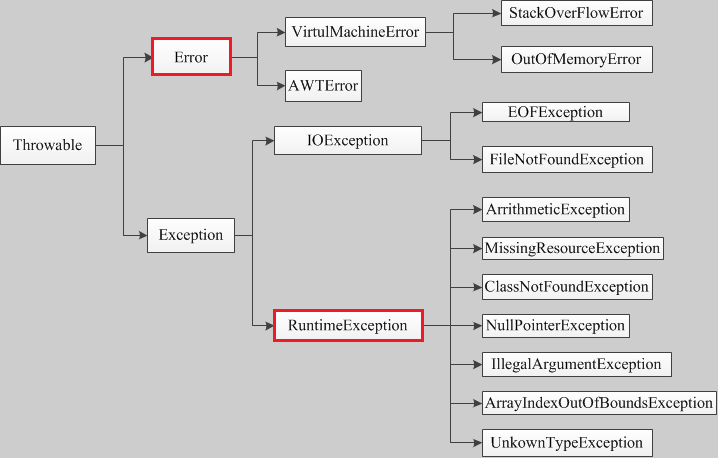
# 简介

1. **程序运行时，发生的不被期望的事件，阻止了程序按照程序员的预期正常执行，这就是异常。**
2. Java 中的异常可以是函数中的语句执行时引发的，也可以是程序员通过 throw 语句手动抛出的，**只要在 Java 程序中产生了异常，就会用一个对应类型的异常对象来封装异常，JRE 就会试图寻找异常处理程序来处理异常**。
3. Throwable 类是 Java 异常类型的顶层父类，一个对象只有是 Throwable 类的（直接或者间接）实例，他才是一个异常对象，才能被异常处理机制识别。JDK 中内建了一些常用的异常类，我们也可以自定义异常。

# 异常的分类

## Java中Throwable类结构图



1. 错误：Error 类以及他的子类的实例，**代表了 JVM 本身的错误**。错误不能被程序员通过代码处理，Error 很少出现。因此，程序员应该关注 Exception 为父类的分支下的各种异常类。
2. 异常：Exception 以及他的子类，代表程序运行时发送的各种不期望发生的事件。**可以被 Java 异常处理机制使用，是异常处理的核心。**

**图片纠正：**

**Exception类可分为RuntimeException，IOException，ClassNotFoundException，**

**ClassNotSupportedException等**

## 根据Javac处理机制分类

1. **非检查异常（unckecked exception）：Error 和 RuntimeException 以及他们的子类。**javac 在编译时，不会提示和发现这样的异常，不要求在程序处理这些异常。所以如果愿意，我们可以编写代码处理（使用 try...catch...finally）这样的异常，也可以不处理。对于这些异常，我们应该修正代码，而不是去通过异常处理器处理 。这样的异常发生的原因多半是代码写的有问题。如除 0 错误 ArithmeticException，错误的强制类型转换错误 ClassCastException，数组索引越界 ArrayIndexOutOfBoundsException，使用了空对象 NullPointerException 等等。
2. **检查异常（checked exception）：除了 Error 和 RuntimeException 的其它异常。**javac 强制要求程序员为这样的异常做预备处理工作（使用 try...catch...finally 或者 throws）。在方法中要么用 try-catch 语句捕获它并处理，要么用 throws 子句声明抛出它，否则编译不会通过。**这样的异常一般是由程序的运行环境导致的**。因为程序可能被运行在各种未知的环境下，而程序员无法干预用户如何使用他编写的程序，于是程序员就应该为这样的异常时刻准备着。

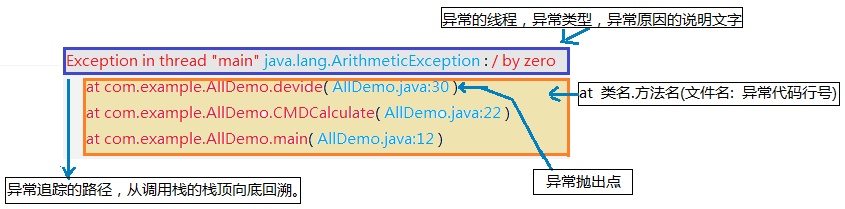
如 SQLException , IOException,ClassNotFoundException 等。

需要明确的是：**检查和非检查是对于 javac （编译期）来说的，这样就很好理解和区分了。**

# 异常的处理

## 非检查异常

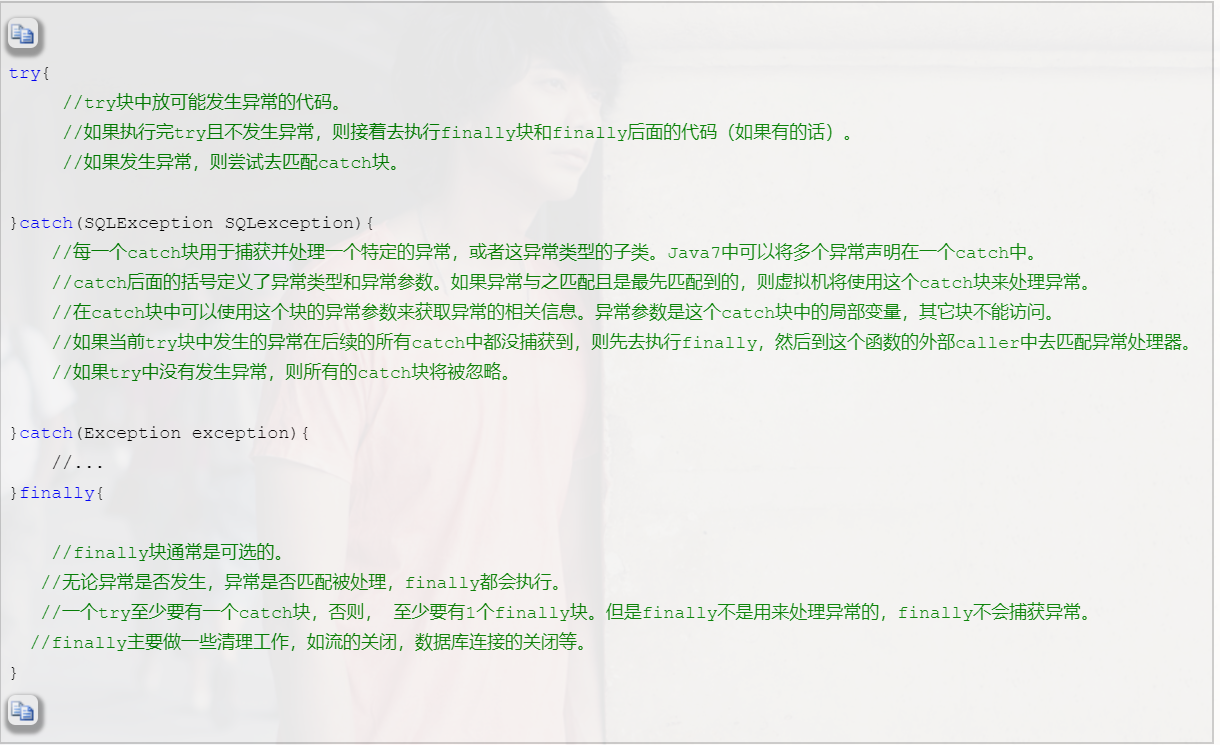
异常是在执行某个函数时引发的，而函数又是层级调用，形成调用栈的，因为，只要一个函数发生了异常，那么他的所有的 caller 都会被异常影响。当这些被影响的函数以异常信息输出时，就形成的了异常追踪栈。**异常最先发生的地方，叫做异常抛出点。**



1. 当一处地方发生异常，则调用它的函数也会发生异常，一直到最外层的函数。这种行为叫做异常的冒泡，异常的冒泡是为了在当前发生异常的函数或者这个函数的 caller 中找到最近的异常处理程序。如果没有使用任何异常处理机制，因此异常最终由 main 函数抛给 JRE，导致程序终止。

## 异常处理的基本语法

* 1. 捕获异常 try...catch...finally 语句块



1. try 块中的局部变量和 catch 块中的局部变量（包括异常变量），以及 finally 中的局部变量，他们之间不可共享使用。

1. 每一个 catch 块用于处理一个异常。异常匹配是按照 catch 块的顺序从上往下寻找的，只有第一个匹配的 catch 会得到执行。匹配时，不仅运行精确匹配，也支持父类匹配，因此，如果同一个 try 块下的多个 catch 异常类型有父子关系，应该将子类异常放在前面，父类异常放在后面，这样保证每个 catch 块都有存在的意义。

1. **Java中，异常处理的任务就是将执行控制流从异常发生的地方转移到能够处理这种异常的地方去。**也就是说，当一个函数的某条语句发生异常时，这条语句的后面的语句不会再执行，它失去了焦点。执行流跳转到最近的匹配的异常处理 catch 代码块去执行，异常被处理完后，执行流会接着在 “处理了这个异常的 catch 代码块” 后面接着执行。
2. 有的编程语言当异常被处理后，控制流会恢复到异常抛出点接着执行，这种策略叫做：resumption model of exception handling（恢复式异常处理模式 ）

而Java 则是让执行流恢复到处理了异常的 catch 块后接着执行，这种策略叫做：termination model of exception handling（终结式异常处理模式）

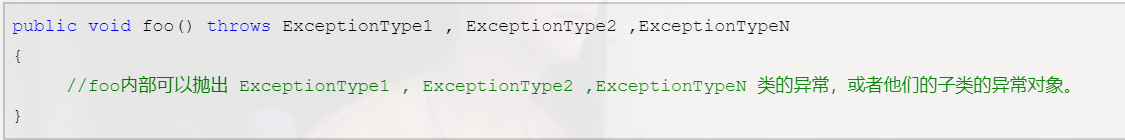
1. 在try{return}catch{}finally{return}中有return

**任何执行 try 或者 catch 中的 return 语句之前，都会先执行 finally 语句，如果 finally 存在的话。**

**如果 finally 中有 return 语句，那么程序就 return 了，所以 finally 中的 return 是一定会被 return 的**。

编译器把 finally 中的 return 实现为一个 warning。

## Throws抛出异常



1. throws 声明：如果一个方法内部的代码会抛出检查异常（checked exception），而方法自己又没有完全处理掉，则 javac 保证你必须在方法的签名上使用 throws 关键字声明这些可能抛出的异常，否则编译不通过。
2. throws 是另一种处理异常的方式，它不同于 try...catch...finally，throws 仅仅是将函数中可能出现的异常向调用者声明，而自己则不具体处理。

采取这种异常处理的原因可能是：方法本身不知道如何处理这样的异常，或者说让调用者处理更好，调用者需要为可能发生的异常负责。

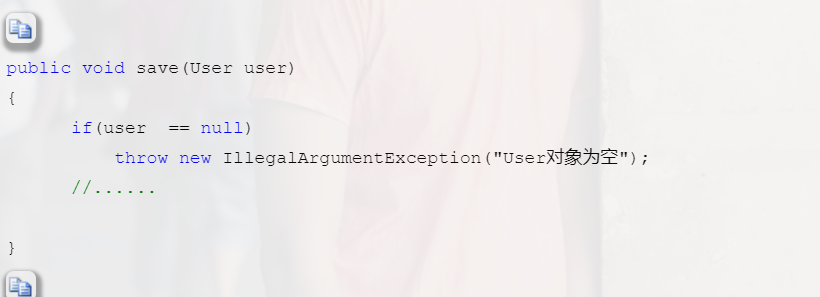
## final语句

1. finally 块不管异常是否发生，只要对应的 try 执行了，则它一定也执行。只有一种方法让 finally 块不执行：System.exit()。因此 finally 块通常用来做资源释放操作：关闭文件，关闭数据库连接等等。

良好的编程习惯是：在 try 块中打开资源，在 finally 块中清理释放这些资源。

1. finally 块没有处理异常的能力。处理异常的只能是 catch 块。
2. 在同一 try...catch...finally 块中 ，如果 try 中抛出异常，且有匹配的 catch 块，则先执行 catch 块，再执行 finally 块。如果没有 catch 块匹配，则先执行 finally，然后去外面的调用者中寻找合适的 catch 块。
3. 在同一 try...catch...finally 块中 ，try 发生异常，且匹配的 catch 块中处理异常时也抛出异常，那么后面的 finally 也会执行：首先执行 finally 块，然后去外围调用者中寻找合适的 catch 块。

## Throw异常抛出语句



throw exceptionObject

程序员也可以通过 throw 语句手动显式的抛出一个异常。throw 语句的后面必须是一个异常对象。

throw 语句必须写在函数中，执行 throw 语句的地方就是一个异常抛出点，它和由 JRE 自动形成的异常抛出点没有任何差别。

# 异常的链化

1. 出现背景：

在一些大型的，模块化的软件开发中，一旦一个地方发生异常，则如骨牌效应一样，将导致一连串的异常。假设 B 模块完成自己的逻辑需要调用 A 模块的方法，如果 A 模块发生异常，则 B 也将不能完成而发生异常，但是 B 在抛出异常时，会将 A 的异常信息掩盖掉，这将**使得异常的根源信息丢失**。异常的链化可以将多个模块的异常串联起来，使得异常信息不会丢失。

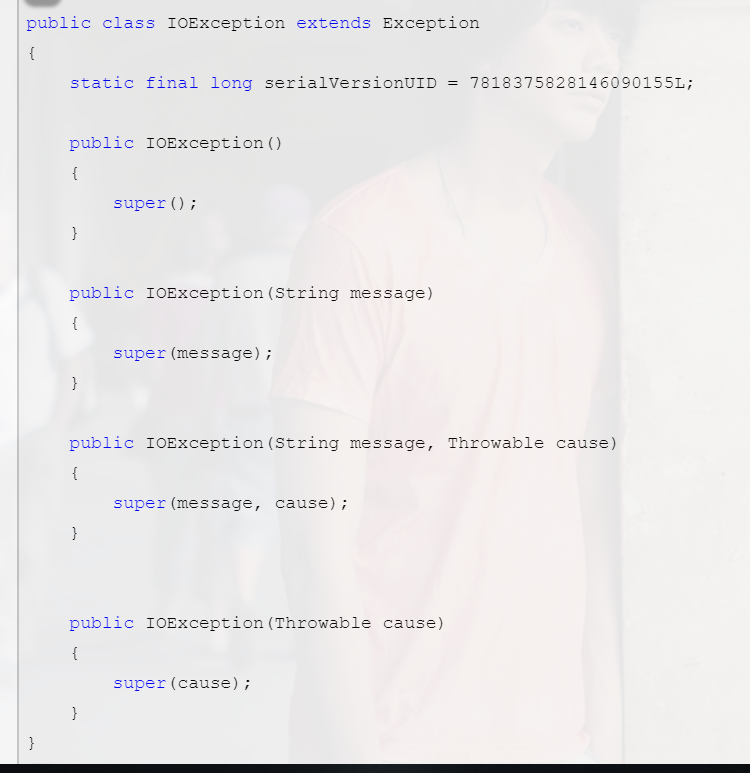
1. 定义：

以一个异常对象为参数构造新的异常对象。新的异对象将包含先前异常的信息。

1. 简单来说就是，最底层出现异常，将异常信息写在一个新异常中，然后抛出给上一层的调用函数。这样最外层的函数抛出的异常中，就有所有的异常信息。

# 自定义异常

1. 如果要自定义异常类，则扩展 Exception 类即可，因此这样的自定义异常都属于检查异常（checked exception）。如果要自定义非检查异常，则扩展自 RuntimeException。
2. 按照国际惯例，自定义的异常应该总是包含如下的构造函数：
3. 一个无参构造函数
4. 一个带有 String 参数的构造函数，并传递给父类的构造函数。
5. 一个带有 String 参数和 Throwable 参数，并都传递给父类构造函数
6. 一个带有 Throwable 参数的构造函数，并传递给父类的构造函数。
7. IOException的异常源码，自定义异常的标准写法



# 抛出异常注意事项

1. 当子类重写父类的带有 throws 声明的函数时，其 throws 声明的异常必须在父类异常的可控范围内——用于处理父类的 throws 方法的异常处理器，必须也适用于子类的这个带 throws 方法 。这是为了支持多态。
2. Java 程序可以是多线程的。每一个线程都是一个独立的执行流，独立的函数调用栈。如果程序只有一个线程，那么没有被任何代码处理的异常 会导致程序终止。如果是多线程的，那么没有被任何代码处理的异常仅仅会导致异常所在的线程结束。

也就是说，**Java 中的异常是线程独立的，线程的问题应该由线程自己来解决**，而不要委托到外部，也不会直接影响到其它线程的执行。

# final块和return

1. finally 中的 return 会覆盖 try 或者 catch 中的返回值。

如果finally中有返回值。

1. finally 中的 return 会抑制（消灭）前面 try 或者 catch 块中的异常。

如果finally中有return正常值，try或catch中的异常不会被捕获。

1. finally 中的异常会覆盖（消灭）前面 try 或者 catch 中的异常。

如果finally中有抛出异常，则不会捕获其他代码块的异常。

1. 编码习惯：
2. 不要在 fianlly 中使用 return。
3. 不要在 finally 中抛出异常。
4. 减轻 finally 的任务，不要在 finally 中做一些其它的事情，finally 块仅仅用来释放资源是最合适的。
5. 将尽量将所有的 return 写在函数的最后面，而不是 try ... catch ... finally 中。