## Java异常简介

　　Java异常是Java提供的一种识别及响应错误的一致性机制。

　　Java异常机制可以使程序中异常处理代码和正常业务代码分离，保证程序代码更加优雅，并提高程序健壮性。在有效使用异常的情况下，异常能清晰的回答what, where, why这3个问题：异常类型回答了“什么”被抛出，异常堆栈跟踪回答了“在哪“抛出，异常信息回答了“为什么“会抛出。

Java异常机制用到的几个关键字：try、catch、finally、throw、throws。

• try -- 用于监听。将要被监听的代码(可能抛出异常的代码)放在try语句块之内，当try语句块内发生异常时，异常就被抛出。

• catch -- 用于捕获异常。catch用来捕获try语句块中发生的异常。

• finally -- finally语句块总是会被执行。它主要用于回收在try块里打开的物力资源(如数据库连接、网络连接和磁盘文件)。只有finally块，执行完成之后，才会回来执行try或者catch块中的return或者throw语句，如果finally中使用了return或者throw等终止方法的语句，则就不会跳回执行，直接停止。

• throw -- 用于抛出异常。

• throws -- 用在方法签名中，用于声明该方法可能抛出的异常。

### try、catch、finally三个语句块应注意的问题：

第一、try、catch、finally三个语句块均不能单独使用，三者可以组成 try...catch...finally、try...catch、try...finally三种结构，catch语句可以有一个或多个，finally语句最多一个。

第二、try、catch、finally三个代码块中变量的作用域为代码块内部，分别独立而不能相互访问。如果要在三个块中都可以访问，则需要将变量定义到这些块的外面。

第三、多个catch块时候，只会匹配其中一个异常类并执行catch块代码，而不会再执行别的catch块，并且匹配catch语句的顺序是由上到下；

### throw与throws：

throw关键字是用于方法体内部，用来抛出一个Throwable类型的异常。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。有关异常的转译会在下面说明。

throws关键字用于方法体外部的方法声明部分，用来声明方法可能会抛出某些异常。仅当抛出了检查异常，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，而不是囫囵吞枣一般在catch块中打印一下堆栈信息做个勉强处理。

### Throwable类中的常用方法

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

### 异常处理的一般原则：

1、 能处理的异常就早处理，抛出但不能处理的就想法消化掉或者转换为RuntimeException处理。因为对于一个应用系统来说，抛出大量异常是有问题的，应该从程序开发角度尽可能的控制异常发生的可能。

2、 对于检查异常，如果不能行之有效的处理，还不如转换为RuntimeException抛出。这样也让上层的代码有选择的余地――可处理也可不处理。

3、 对于一个应用系统来说，应该有自己的一套异常处理框架，这样当异常发生时，也能得到统一的处理风格，将优雅的异常信息反馈给用户

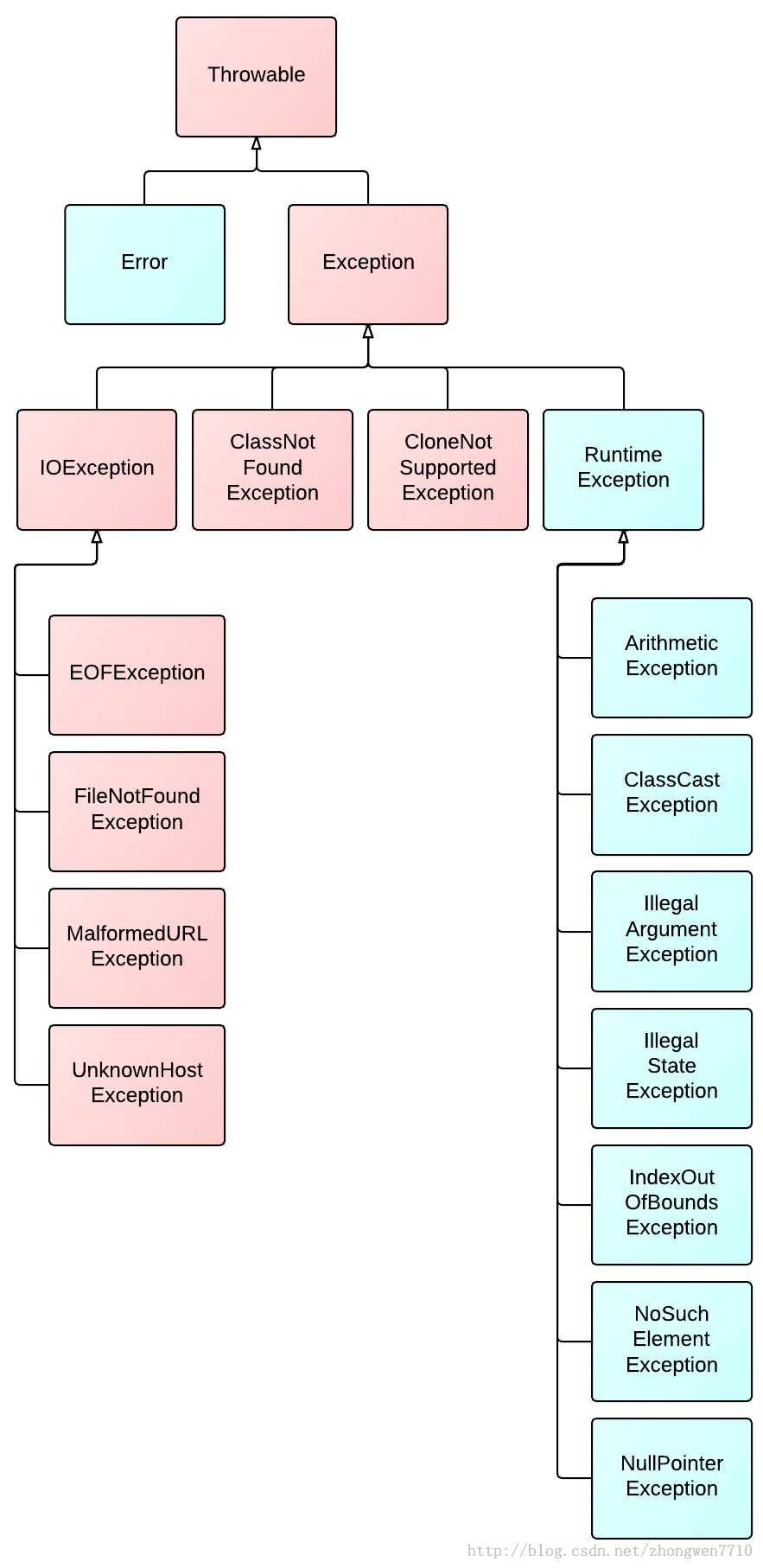
## Java异常框架

Java把异常当作对象来处理，并定义一个基类java.lang.Throwable作为所有异常的超类。在Java API中已经定义了许多异常类，这些异常类分为两大类，错误Error和异常Exception。Java异常体系结构呈树状，其层次结构图如图 1所示：

Java异常架构图:



下图红色为检查异常，就是eclipse要提示你是try catch 还是throws。



### 概述：

1、Error与Exception

　　Error是程序无法处理的错误，比如OutOfMemoryError、ThreadDeath等。这些异常发生时，Java虚拟机(JVM)一般会选择线程终止。

　　Exception是程序本身可以处理的异常，这种异常分两大类运行时异常和非运行时异常。程序中应当尽可能去处理这些异常。

2、运行时异常和非运行时异常

　　运行时异常都是RuntimeException类及其子类异常，如NullPointerException、 IndexOutOfBoundsException等，这些异常是不检查异常，程序中可以选择捕获处理，也可以不处理。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。

非运行时异常是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。

### 分述：

1. Throwable

　　Throwable是 Java 语言中所有错误或异常的超类。

　　Throwable包含两个子类: Error 和 Exception。它们通常用于指示发生了异常情况。

　　Throwable包含了其线程创建时线程执行堆栈的快照，它提供了printStackTrace()等接口用于获取堆栈跟踪数据等信息。

2. Exception

　　Exception及其子类是 Throwable 的一种形式，它指出了合理的应用程序想要捕获的条件。

3. RuntimeException

　　RuntimeException是那些可能在 Java 虚拟机正常运行期间抛出的异常的超类。这些异常一般是由逻辑错误引起的，程序应该从逻辑角度避免这些异常的发生；

编译器不会检查RuntimeException异常。例如，除数为零时，抛出ArithmeticException异常。RuntimeException是ArithmeticException的超类。当代码发生除数为零的情况时，倘若既"没有通过throws声明抛出ArithmeticException异常"，也"没有通过try...catch...处理该异常"，也能通过编译。这就是我们所说的"编译器不会检查RuntimeException异常"！

出现运行时异常时，如果没有捕获这个异常（catch处理），系统会把异常一直往上层抛出，一直到最上层，如果是多线程就由Thread.run()抛出，如果是单线程就被main()抛出，抛出之后，如果是线程，这个线程就会退出；如果是主程序抛出的异常，那么这整个程序也就退出了。如果不对运行时异常进行处理，那么出现运行时异常时，要么是线程中止，要么是主程序中止；

　　如果代码会产生RuntimeException异常，则需要通过修改代码进行避免。例如，若会发生除数为零的情况，则需要通过代码避免该情况的发生！

4. Error

　　和Exception一样，Error也是Throwable的子类。它用于指示合理的应用程序不应该试图捕获的严重问题，大多数这样的错误都是异常条件。一般是指Java虚拟机相关的问题，如系统崩溃、虚拟机出现错误、动态链接失败等；

　　和RuntimeException一样，编译器也不会检查Error。

**Java将可抛出(Throwable)的结构分为三种类型：被检查的异常(Checked Exception)，运行时异常(RuntimeException)和错误(Error)。**

(01) 运行时异常

定义: RuntimeException及其子类都被称为运行时异常。

特点: 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时，倘若既"没有通过throws声明抛出它"，也"没有用try-catch语句捕获它"，还是会编译通过。例如，除数为零时产生的ArithmeticException异常，数组越界时产生的IndexOutOfBoundsException异常，fail-fail机制产生的ConcurrentModificationException异常等，都属于运行时异常。

　　虽然Java编译器不会检查运行时异常，但是我们也可以通过throws进行声明抛出，也可以通过try-catch对它进行捕获处理。

　　如果产生运行时异常，则需要通过修改代码来进行避免。例如，若会发生除数为零的情况，则需要通过代码避免该情况的发生！

(02) 被检查的异常

定义: Exception类本身，以及Exception的子类中除了"运行时异常"之外的其它子类都属于被检查异常。

特点: Java编译器会检查它。此类异常，要么通过throws进行声明抛出，要么通过try-catch进行捕获处理，否则不能通过编译。例如，CloneNotSupportedException就属于被检查异常。当通过clone()接口去克隆一个对象，而该对象对应的类没有实现Cloneable接口，就会抛出CloneNotSupportedException异常。被检查异常通常都是可以恢复的。

(03) 错误

定义: Error类及其子类。

特点: 和运行时异常一样，编译器也不会对错误进行检查。

**当资源不足、约束失败、或是其它程序无法继续运行的条件发生时，就产生错误。**程序本身无法修复这些错误的。例如，VirtualMachineError就属于错误。

　　按照Java惯例，我们是不应该实现任何新的Error子类的！

对于上面的3种结构，我们在抛出异常或错误时，到底该哪一种？《Effective Java》中给出的建议是：对于可以恢复的条件使用被检查异常，对于程序错误使用运行时异常。

## 异常的转译与异常链：

1、异常转译的原理

所谓的异常转译就是将一种异常转换另一种新的异常，也许这种新的异常更能准确表达程序发生异常。

在Java中有个概念就是异常原因，异常原因是导致当前抛出异常的那个异常对象，几乎所有带异常原因的异常构造方法都使用Throwable类型做参数，这 也就为异常的转译提供了直接的支持，因为任何形式的异常和错误都是Throwable的子类。比如将SQLException转换为另外一个新的异常 DAOException，可以这么写：

先自定义一个异常DAOException：

public class DAOException extends RuntimeException {

public DAOException(String message, Throwable cause) {

super(message, cause);

}

比如有一个SQLException类型的异常对象e，要转换为DAOException，可以这么写：

DAOException daoEx = new DAOException ( "SQL异常", e);

异常转译是针对所有继承Throwable超类的类而言的，从编程的语法角度讲，其子类之间都可以相互转换。但是，从合理性和系统设计角度考虑，可将异常 分为三类：Error、Exception、RuntimeException；

为什么要这么做呢？笔者认为，异常的处理存在着一套哲学思想：对于一个应用系统来说，系统所发生的任何异常或者错误对操作用户来说都是系统"运行时"异常，都是这个应用系统内部的异常。这也是异常转译和应用系统异常框架设计的指导原则。在系统中大量处理非检查异常的负面影响很多，最重要的一个方面就是代码可读性降低，程序编写复杂，异常处理的代码也很苍白无力。因此，很有必要将这些检查异常Exception和错误Error转换为RuntimeException异常，让程序员根据情况来决定是否捕获和处理所发生的异常。

三种情况：

①：Error到Exception：将错误转换为异常，并继续抛出。例如Spring WEB框架中，将org.springframework.web.servlet.DispatcherServlet的doDispatch()方法中，将捕获的错误转译为一个NestedServletException异常。这样做的目的是为了最大限度挽回因错误发生带来的负面影响。因为一个 Error常常是很严重的错误，可能会引起系统挂起。

②：Exception到RuntimeException：将检查异常转换为RuntimeException可以让程序代码变得更优雅，让开发人员集中精力设计更合理的程序代码，反过来也增加了系统发生异常的可能性。

③：Error到RuntimeException：目的还是一样的。把所有的异常和错误转译为不检查异常，这样可以让代码更为简洁，还有利于对错误和异常信息的统一处理。

1、 异常链

异常链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。Java API文档中给出了一个简单的模型：

1. **try** {
2. lowLevelOp();
3. } **catch** (LowLevelException le) {
4. **throw** (HighLevelException)
5. **new** HighLevelException().initCause(le);
6. }

当程序捕获到了一个底层异常le，在处理部分选择了继续抛出一个更高级别的新异常给此方法的调用 者。这样异常的原因就会逐层传递。这样，位于高层的异常递归调用getCause()方法，就可以遍历各层的异常原因。这就是Java异常链的原理。异常 链的实际应用很少，发生异常时候逐层上抛不是个好注意，上层拿到这些异常又能奈之何？而且异常逐层上抛会消耗大量资源，因为要保存一个完整的异常链信息。

Java异常处理总结

## 回顾全文，总结一下Java异常处理的要点：

1、 异常是程序运行过程出现的错误，在Java中用类来描述，用对象来表示具体的异常。

Java将其区分为Error与Exception，Error是程序无力处理的错误，Exception是程序可以处理的错误。

异常处理是为了程序的健壮性。

2、 Java异常类来自于Java API定义和用户扩展。通过继承Java API异常类可以实现异常的转译。

3、 异常能处理就处理，不能处理就抛出，最终没有处理的异常JVM会进行处理。

4、 异常可以传播，也可以相互转译，但应该根据需要选择合理的异常转译的方向。

5、 对于一个应用系统，设计一套良好的异常处理体系很重要。这一点在系统设计的时候就应该考虑到

**Java内部类的实例化：**

**static描述：A.B c = new A.B();**

**非static描述：A.B c = new A().new B();**