Java 与 C++ 的区别

Java 是纯粹的面向对象语言，所有的对象都继承自 java.lang.Object，C++ 为了兼容 C 即支持面向对象也支持

面向过程。

Java 通过虚拟机从而实现跨平台特性，但是 C++ 依赖于特定的平台。

Java 没有指针，它的引用可以理解为安全指针，而 C++ 具有和 C 一样的指针。

Java 支持自动垃圾回收，而 C++ 需要手动回收。

Java 不支持多重继承，只能通过实现多个接口来达到相同目的，而 C++ 支持多重继承。

Java 不支持操作符重载，虽然可以对两个 String 对象执行加法运算，但是这是语言内置支持的操作，不属于操

作符重载，而 C++ 可以。

Java 的 goto 是保留字，但是不可用，C++ 可以使用 goto。

Java 不支持条件编译，C++ 通过 #ifdef #ifndef 等预处理命令从而实现条件编译。

## &和&&的区别？

答：&运算符有两种用法：(1)按位与；(2)逻辑与。&&运算符是短路与运算。逻辑与跟短路与的差别是非常巨大的，虽然二者都要求运算符左右两端的布尔值都是true整个表达式的值才是true。&&之所以称为短路运算是因为，如果&&左边的表达式的值是false，右边的表达式会被直接短路掉，不会进行运算。很多时候我们可能都需要用&&而不是&，例如在验证用户登录时判定用户名不是null而且不是空字符串，应当写为：username != null &&!username.equals(“”)，二者的顺序不能交换，更不能用&运算符，因为第一个条件如果不成立，根本不能进行字符串的equals比较，否则会产生NullPointerException异常。

注意：逻辑或运算符（|）和短路或运算符（||）的差别也是如此。

String和StringBuilder、StringBuffer的区别？  
答：Java平台提供了两种类型的字符串：String和StringBuffer/StringBuilder，它们可以储存和操作字符串。其中String是只读字符串，也就意味着String引用的字符串内容是不能被改变的。而StringBuffer/StringBuilder类表示的字符串对象可以直接进行修改。StringBuilder是Java 5中引入的，它和StringBuffer的方法完全相同，区别在于它是在单线程环境下使用的，因为它的所有方面都没有被synchronized修饰，因此它的效率也比StringBuffer要高。

**Object 通用方法**

**equals()**

1. 等价关系

2. 等价与相等

**hashCode()**

hashCode() 返回散列值，而 equals() 是用来判断两个对象是否等价。等价的两个对象散列值一定相同，但是散列值相同的两个对象不一定等价。在覆盖 equals() 方法时应当总是覆盖 hashCode() 方法，保证等价的两个对象散列值也相等。

**toString()**

默认返回 ToStringExample@4554617c 这种形式，其中 @ 后面的数值为散列码的无符号十六进制表示。

**clone()**

**1. cloneable**

clone() 是 Object 的 protected 方法，它不是 public，一个类不显式去重写 clone()，其它类就不能直接去调用该类

浅拷贝：拷贝对象和原始对象的引用类型引用同一个对象。

深拷贝：拷贝对象和原始对象的引用类型引用不同对象。

**clone() 的替代方案**

使用 clone() 方法来拷贝一个对象即复杂又有风险，它会抛出异常，并且还需要类型转换。Effective Java 书上讲到，最好不要去使用 clone()，可以使用拷贝构造函数或者拷贝工厂来拷贝一个对象。

**在调用一个方法时，先从本类中查找看是否有对应的方法，如果没有查找到再到父类中查看，看是否有继承来的方法。否则就要对参数进行转型，转成父类之后看是否有对应的方法。总的来说，方法调用的优先级为：**

this.func(this)

super.func(this)

this.func(super)

super.func(super)

**关键字**

**（一）Final**

1.数据

声明数据为常量，可以是编译时常量，也可以是在运行时被初始化后不能被改变的常量。

对于基本类型，final 使数值不变；

对于引用类型，final 使引用不变，也就不能引用其它对象，但是被引用的对象本身是可以修改的。

2. 方法

声明方法不能被子类重写。

private 方法隐式地被指定为 final，如果在子类中定义的方法和基类中的一个 private 方法签名相同，此时子类的方法不是重写基类方法，而是在子类中定义了一个新的方法。

3. 类

声明类不允许被继承。

**（二）static**

1. 静态变量

静态变量：又称为类变量，也就是说这个变量属于类的，类所有的实例都共享静态变量，可以直接通过类名来

访问它。静态变量在内存中只存在一份。

实例变量：每创建一个实例就会产生一个实例变量，它与该实例同生共死。

2. 静态方法

静态方法在类加载的时候就存在了，它不依赖于任何实例。所以静态方法必须有实现，也就是说它不能是抽象方法。

只能访问所属类的静态字段和静态方法，方法中不能有 this 和 super 关键字。

3. 静态语句块

静态语句块在类初始化时运行一次。

4. 静态内部类

非静态内部类依赖于外部类的实例，而静态内部类不需要。

静态内部类不能访问外部类的非静态的变量和方法。

5. 静态导包

在使用静态变量和方法时不用再指明 ClassName，从而简化代码，但可读性大大降低。

6. 初始化顺序

静态变量和静态语句块优先于实例变量和普通语句块，静态变量和静态语句块的初始化顺序取决于它们在代码中的顺

序。

存在继承的情况下，初始化顺序为：

父类（静态变量、静态语句块）

子类（静态变量、静态语句块）

父类（实例变量、普通语句块）

父类（构造函数）

子类（实例变量、普通语句块）

子类（构造函数）

**反射**

每个类都有一个 Class 对象，包含了与类有关的信息。当编译一个新类时，会产生一个同名的 .class 文件，该文件内容保存着 Class 对象。类加载相当于 Class 对象的加载，类在第一次使用时才动态加载到 JVM 中。也可以使用Class.forName("com.mysql.jdbc.Driver") 这种方式来控制类的加载，该方法会返回一个 Class 对象。

反射可以提供运行时的类信息，并且这个类可以在运行时才加载进来，甚至在编译时期该类的 .class 不存在也可以加载进来。Class 和 java.lang.reflect 一起对反射提供了支持，java.lang.reflect 类库主要包含了以下三个类：

Field ：可以使用 get() 和 set() 方法读取和修改 Field 对象关联的字段；

Method ：可以使用 invoke() 方法调用与 Method 对象关联的方法；

Constructor ：可以用 Constructor 创建新的对象。

反射的优点：

可扩展性 ：应用程序可以利用全限定名创建可扩展对象的实例，来使用来自外部的用户自定义类。

类浏览器和可视化开发环境 ：一个类浏览器需要可以枚举类的成员。可视化开发环境（如 IDE）可以从利用反射中可用的类型信息中受益，以帮助程序员编写正确的代码。

调试器和测试工具 ： 调试器需要能够检查一个类里的私有成员。测试工具可以利用反射来自动地调用类里定义的可被发现的 API 定义，以确保一组测试中有较高的代码覆盖率。

反射的缺点：

尽管反射非常强大，但也不能滥用。如果一个功能可以不用反射完成，那么最好就不用。在我们使用反射技术时，下面几条内容应该牢记于心。

性能开销 ：反射涉及了动态类型的解析，所以 JVM 无法对这些代码进行优化。因此，反射操作的效率要比那些非反射操作低得多。我们应该避免在经常被执行的代码或对性能要求很高的程序中使用反射。

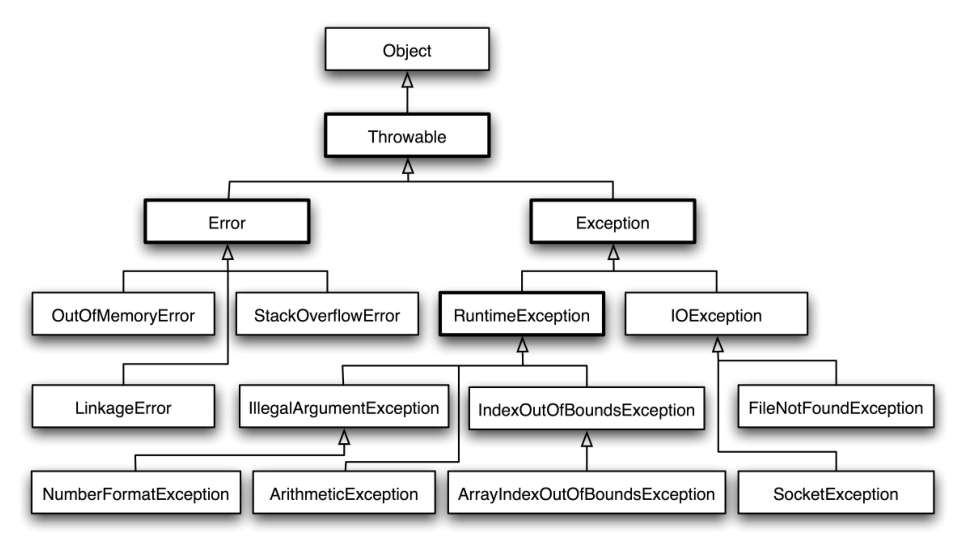
安全限制 ：使用反射技术要求程序必须在一个没有安全限制的环境中运行。如果一个程序必须在有安全限制的环境中运行，如 Applet，那么这就是个问题了。

内部暴露 ：由于反射允许代码执行一些在正常情况下不被允许的操作（比如访问私有的属性和方法），所以使用反射可能会导致意料之外的副作用，这可能导致代码功能失调并破坏可移植性。反射代码破坏了抽象性，因此当平台发生改变的时候，代码的行为就有可能也随着变化。

# Java异常分类

一、基本概念

看java的异常结构图



Throwable是所有异常的根，java.lang.Throwable

Error是错误，java.lang.Error

Exception是异常，java.lang.Exception

二、Exception

一般分为Checked异常和Runtime异常，所有RuntimeException类及其子类的实例被称为Runtime异常，不属于该范畴的异常则被称为CheckedException。

①Checked异常

只有java语言提供了Checked异常，Java认为Checked异常都是可以被处理的异常，所以Java程序必须显示处理Checked异常。如果程序没有处理Checked异常，该程序在编译时就会发生错误无法编译。这体现了Java的设计哲学：没有完善错误处理的代码根本没有机会被执行。对Checked异常处理方法有两种

1 当前方法知道如何处理该异常，则用try...catch块来处理该异常。

2 当前方法不知道如何处理，则在定义该方法是声明抛出该异常。

总异常类，既有checkedException又有RuntimeException，所以其中的checkedException必须处理

我们比较熟悉的Checked异常有

Java.lang.ClassNotFoundException

Java.lang.NoSuchMetodException

java.io.IOException

②RuntimeException

Runtime如除数是0和数组下标越界等，其产生频繁，处理麻烦，若显示申明或者捕获将会对程序的可读性和运行效率影响很大。所以由系统自动检测并将它们交给缺省的异常处理程序。当然如果你有处理要求也可以显示捕获它们。

我们比较熟悉的RumtimeException类的子类有

Java.lang.ArithmeticException

Java.lang.ArrayStoreExcetpion

Java.lang.ClassCastException

Java.lang.IndexOutOfBoundsException

Java.lang.NullPointerException

三、Error

当程序发生不可控的错误时，通常做法是通知用户并中止程序的执行。与异常不同的是Error及其子类的对象不应被抛出。

Error是throwable的子类，代表编译时间和系统错误，用于指示合理的应用程序不应该试图捕获的严重问题。

Error由Java虚拟机生成并抛出，包括动态链接失败，虚拟机错误等。程序对其不做处理。