final关键字的作用

# final的简单认识、

## final的简单理解

final的简单理解为**终态，不可变**。(线程安全：多线程共享数据，若共享对象是不可变的，肯定是线程安全的)

## final关键字的认识

谈到final关键字，想必很多人都不陌生，**在使用匿名内部类的时候可能会经常用到final关键字。**另外，Java中的String类就是一个典型的final类，java.lang.Number包中的大部分类都是final类，枚举类都是final类。

# final关键字的基本用法

在Java中，final关键字可以用来修饰**类、方法和变量**（包括成员变量和局部变量）。

## 修饰类

**当用final修饰一个类时，表明这个类不能被继承**。也就是说，如果一个类你永远不会让他被继承，就可以用final进行修饰。

**注意**：final类中的所有成员方法都会被隐式地指定为final方法(因为没有继承子类，当然成员方法也就不会被Override重写)；final类中的成员变量可以根据需要设为final(final类中的成员变量在没有指定final的情况下，都不是final的)。

**final类中的成员方法都是final方法，但是成员变量并非final变量，除非final修饰**。

在使用final修饰类的时候，要注意谨慎选择，除非这个类真的在以后不会用来继承或者出于安全的考虑，尽量不要将类设计为final类。

## 修饰方法

下面这段话摘自《Java编程思想》第四版第143页：

“**使用final方法的原因有两个。第一个原因是把方法锁定，以防任何继承类修改它的含义；第二个原因是效率**。在早期的Java实现版本中，会将final方法转为内嵌调用。但是如果方法过于庞大，可能看不到内嵌调用带来的任何性能提升。在最近的Java版本中，不需要使用final方法进行这些优化了。“

因此，**利用final修饰方法作用：禁止该方法在子类中被Override重写(覆盖)。**注意：final方法可以在子类中被重载。**类的private方法会隐式地被指定为final方法。**(因为private方法子类是不可见的，当然就不可能被重写Override了)。

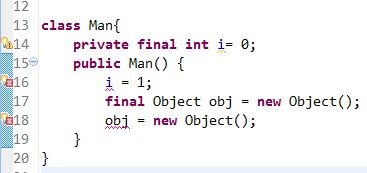
## 修饰变量

修饰变量是final用得最多的地方，也是本文接下来要重点阐述的内容。

final变量的基本语法：

对于一个final变量，**如果是基本数据类型的变量，则其数值一旦在初始化之后便不能更改**；**如果是引用类型的变量，则在对其初始化之后便不能再让其指向另一个对象**。

示例：



上面的一段代码中，对变量i和obj的重新赋值都报错了。

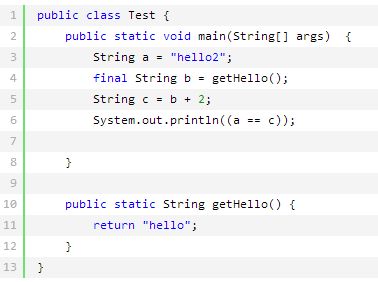
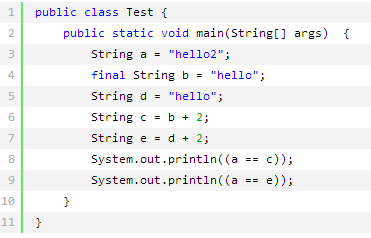
# 深入理解final关键字

final关键字经常在面试题目中出现，存在很多容易混淆的地方。

## 类的final变量与普通变量有什么区别？

当用final作用于类的成员变量时，**成员变量**（注意是类的成员变量，局部变量只需要保证在使用之前被初始化赋值即可）**必须在定义时或者构造器中进行初始化赋值**，**而且final变量一旦被初始化赋值之后，就不能再被赋值了**。

那么final变量和普通变量到底有何区别呢？下面请看一个例子



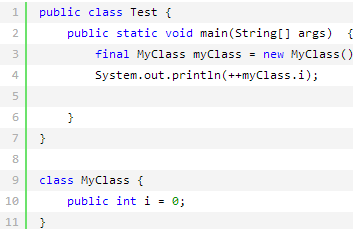
输出结果是true or false？

答案：1. true 、false；2. false

**为什么第一个比较结果为true，而第二个比较结果为fasle**？这里面就是final变量和普通变量的区别了，**当final变量是基本数据类型以及String类型时，如果在编译期间能知道它的确切值，则编译器会把它当做编译期常量使用。**也就是说在用到该final变量的地方，相当于直接访问的这个常量，不需要在运行时确定。这种和C语言中的宏替换有点像。因此在上面的一段代码中，由于变量b被final修饰，因此会被当做编译器常量，所以在使用到b的地方会直接将变量b 替换为它的值。而对于变量d的访问却需要在运行时通过链接来进行。不过要注意，只有在编译期间能确切知道final变量值的情况下，编译器才会进行这样的优化，比如第二段代码就不会进行优化，输出结果为false。

## 被final修饰的引用变量指向的对象内容可变吗？

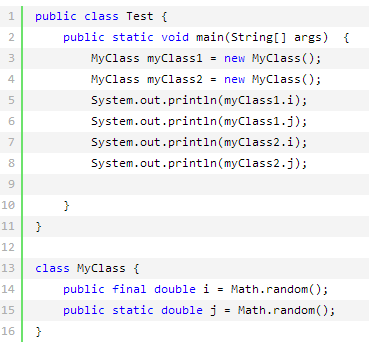
从final变量的基本使用可知：**如果final变量是引用类型的变量，则在对其初始化之后便不能再让其指向另一个对象。但是该final引用变量指向的对象可以改变。**



这段代码可以顺利编译通过并且有输出结果，输出结果为1。这说明引用变量被final修饰之后，虽然不能再指向其他对象，但是它指向的对象的内容是可变的。

## final和static的区别

很多时候会容易把static和final关键字混淆，static作用于成员变量用来表示只保存一份副本，而final的作用是用来保证变量不可变。看下面这个例子：

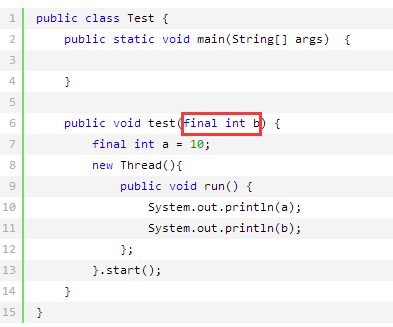


运行这段代码就会发现，每次打印的两个j值都是一样的，而i的值却是不同的。从这里就可以知道final和static变量的区别了。

final变量只是保证不可变；static是在类加载的时候初始化，且只会初始化一次，创建对象的时候不改变static变量，相当于保存了一份副本，共所有对象使用。

## 匿名内部类中使用的外部局部变量为什么只能是final变量？

想必这个问题也曾经困扰过很多人，在讨论这个问题之前，先看下面这段代码：



这段代码会被编译成两个class文件：Test.class和Test1.class。默认情况下，编译器会为匿名内部类和局部内部类起名为Outter1.class。默认情况下，编译器会为匿名内部类和局部内部类起名为Outterx.class（x为正整数）。

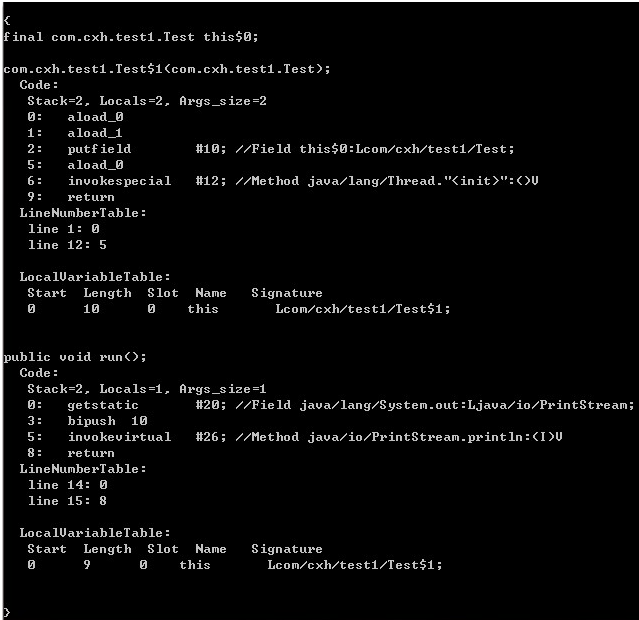


根据上图可知，test方法中的匿名内部类的名字被起为 Test$1。

**上段代码中，如果把变量a和b前面的任一个final去掉，这段代码都编译不过**。

我们先考虑这样一个问题：

当test方法执行完毕之后，变量a的生命周期就结束了，而此时Thread对象的生命周期很可能还没有结束，那么在Thread的run方法中继续访问变量a就变成不可能了，但是又要实现这样的效果，怎么办呢？Java采用了**复制**的手段来解决这个问题。将这段代码的字节码反编译可以得到下面的内容：

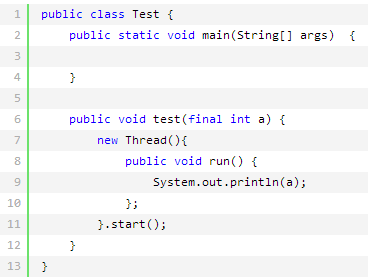


我们看到在run方法中有一条指令：

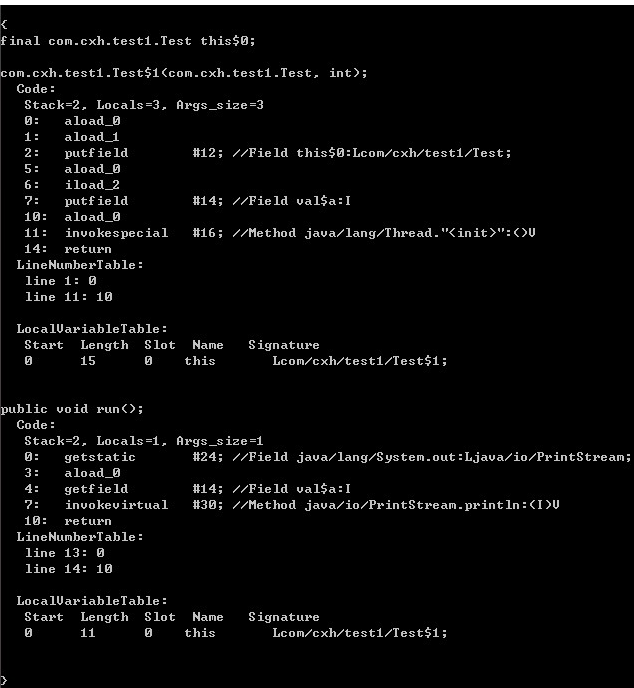
bipush 10

这条指令表示将操作数10压栈，表示使用的是一个本地局部变量。这个过程是在编译期间由编译器默认进行，如果这个变量的值在编译期间可以确定，则编译器默认会在匿名内部类（局部内部类）的常量池中添加一个内容相等的字面量或直接将相应的字节码嵌入到执行字节码中。这样一来，匿名内部类使用的变量是另一个局部变量，只不过值和方法中局部变量的值相等，因此和方法中的局部变量完全独立开。

在来看一个例子：



反编译得到：



我们看到匿名内部类Test$1的构造器含有两个参数，**一个是指向外部类对象的引用，一个是int型变量**，很显然，这里是将变量test方法中的形参a以参数的形式传进来对匿名内部类中的*拷贝（变量a的拷贝）*进行赋值初始化。

**也就说如果局部变量的值在编译期间就可以确定，则直接在匿名内部里面创建一个拷贝。如果局部变量的值无法在编译期间确定，则通过构造器传参的方式来对拷贝进行初始化赋值。**

　从上面可以看出，在run方法中访问的变量a根本就不是test方法中的局部变量a。这样一来就解决了前面所说的 **生命周期不一致的**问题。但是新的问题又来了，既然在run方法中访问的变量a和test方法中的变量a不是同一个变量，当在run方法中改变变量a的值的话，会出现什么情况？

对，会造成数据不一致性，这样就达不到原本的意图和要求。**为了解决这个问题，java编译器就限定必须将变量a限制为final变量，不允许对变量a进行更改（对于引用类型的变量，是不允许指向新的对象），这样数据不一致性的问题就得以解决了。**

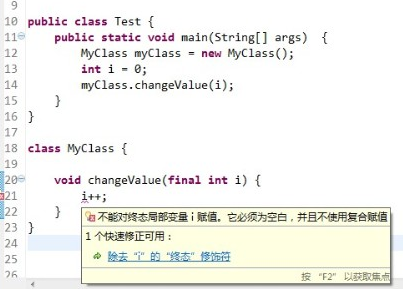
到这里，想必大家应该清楚为何 方法中的局部变量和形参都必须用final进行限定了。

## 关于final变量做方法参数的问题

关于网上流传的”当你在方法中不需要改变作为参数的对象变量时，明确使用final进行声明，会防止你无意的修改而影响到调用方法外的变量“这句话，我个人理解这样说是不恰当的。

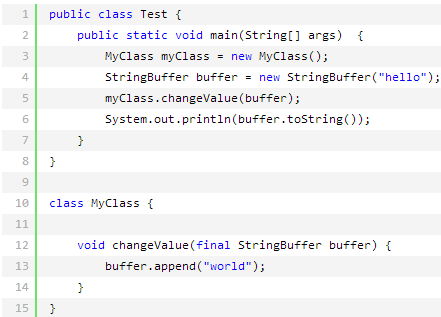
因为无论参数是基本数据类型的变量还是引用类型的变量，使用final声明都不会达到上面所说的效果。

看个例子就清楚了：



上面这段代码好像让人觉得用final修饰之后，就不能在方法中更改变量i的值了。殊不知，方法changeValue和main方法中的变量i根本就不是一个变量，**因为java参数传递采用的是值传递，对于基本类型的变量，相当于直接将变量进行了拷贝**。所以即使没有final修饰的情况下，在方法内部改变了变量i的值也不会影响方法外的i。

再看下面这段代码：



运行这段代码就会发现输出结果为 **helloworld**。很显然，**用final进行修饰并没有阻止在changeValue中改变buffer指向的对象的内容**。(buffer引用变量没有变化，变化的是所指对象的内容)。

有人说假如把final去掉了，万一在changeValue中让buffer指向了其他对象怎么办。有这种想法的朋友可以自己动手写代码试一下这样的结果是什么，如果把final去掉了，然后在changeValue中让buffer指向了其他对象，也不会影响到main方法中的buffer，原因在于java采用的是值传递，对于引用变量，传递的是引用的值，也就是说让实参和形参同时指向了同一个对象，因此让**形参重新指向另一个对象对实参并没有任何影响**。所以关于网上流传的final参数的说法，我个人不是很赞同。

学习自：<http://www.cnblogs.com/dolphin0520/p/3736238.html>