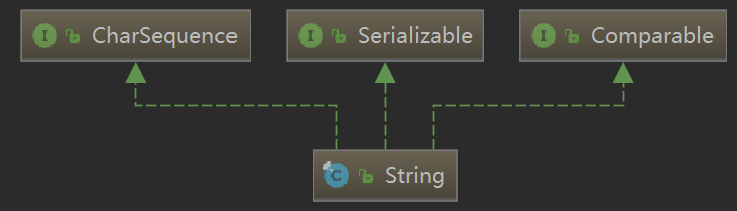
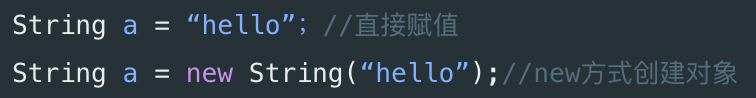
String是java.lang下面的类，它是我们操作字符串最常用的引用类（StringBuffer，StringBuilder）之一。首先我们来看看String类的一个UML图：

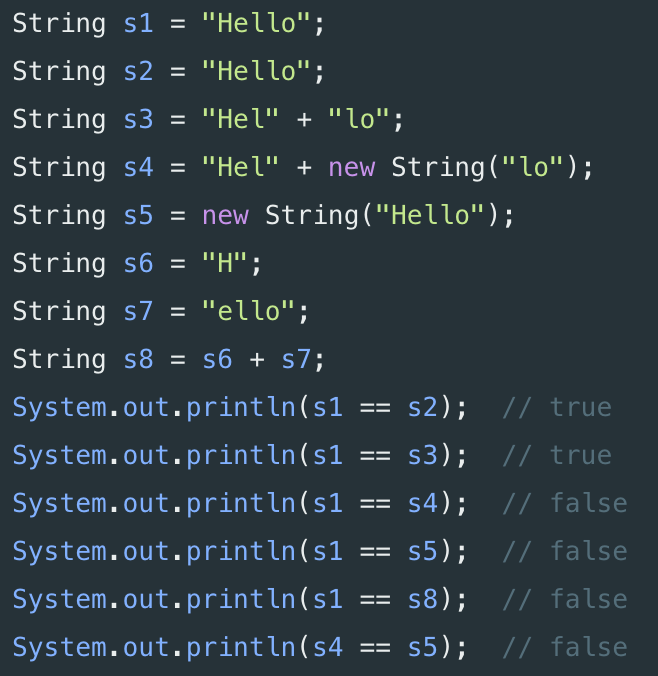


很清楚的看到，它实现了CharSequence（字符序列）接口。CharSequence里面包括length(), charAt(int index), subSequence(int start, int end)，toString()这几个API接口（在jdk1.8中新增了两个default方法，用来转化为字符编码（unicode））。当然，StringBuffer和StringBuilder也实现了CharSequence接口。

String通过维护一个final类型的char数组来实现字符串操作的，也就是说它的大小是不可变的。那么，我们怎么样才能创建一个String呢。既然是一个类，我们先想到构造函数，同时最为引用，又可以直接赋值，因此他就有两种方式进行创建：



首先，我们通过代码以最直观的方式看看两者的区别：



要搞懂这个问题，首先我们要知道它们在内存中的存储位置。首先来看看java的内存图

其中与字符串的创建有关的是方法区（Method Area）、堆区（Heap Area）和栈区（Stack Area）。

**方法区**：存储类信息、常量、静态变量。全局共享。

**堆区**：存放对象和数组。全局共享。

**栈区**：基本数据类型、对象的引用都存放在这。线程私有。

在Jdk1.7以后字符串常量池已经放入了堆内存中，接着我们逐行进行代码分析：

* String s1 = “hello”；

//先创建对象“hello”在字符串常量池中，再返回它的引用给s1。

* String s2 = “hello”；

//先去字符串常量池中寻找是否有相等的字符串，显然s1已经帮他创建好了，它可以直接使用其引用。那么s1和s2所指向的都是同一个地址，所以s1==s2

* String s3 = “Hel” + “lo”；

// s3是一个字符串拼接操作，参与拼接的部分都是字面量(final类型常量)，编译器会进行优化，在编译时s3就变成“Hello”了，所以s1==s3。

* String s4 = “Hel” + new String(“io”);

// s4虽然也是拼接，但“lo”是通过new关键字创建的，在编译期无法知道它的地址，所以不能像s3一样优化。所以必须要等到运行时才能确定，必然新对象的地址和前面的不同。s1!=s4。

* String s5 = new String(“Hello”);

// s5首先在字符串常量池去寻找有没有"Hello"这个字符串，没有就创建一个“Hello”字符串对象在栈中（有这个对象则直接进入下一步），然后new String把这个字符串对象拷贝一份到堆中，返回这个对象的引用。s1!=s5。所以字符串常量池中，没有Hello创建两个对象，有的话则创建一个对象。

* String s8 = s6 + s7;

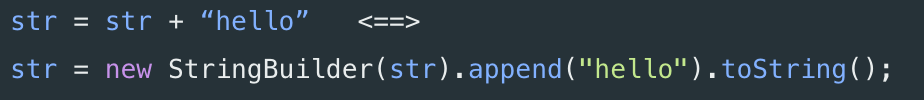
// s8变量拼接，编译期也不知道他们的具体位置，不会做出优化。

**总结：**

new String ()当字符串常量池里面没有该字符串那么会创建两个对象，如果有该字符串则创建一个对象。

**扩展：**

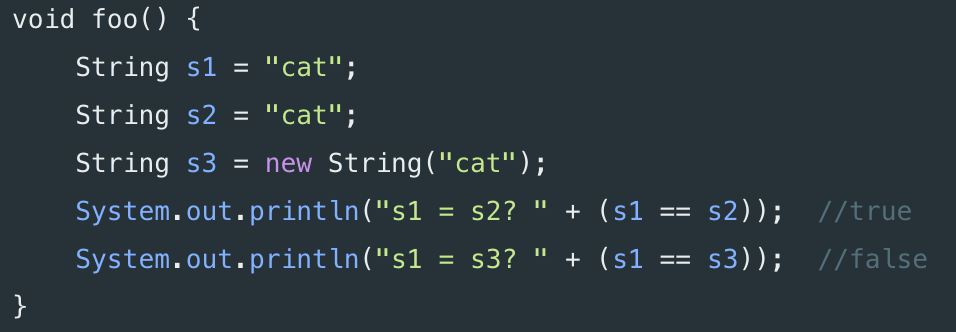
1、在字符串拼接过程中，常常不会用到+号拼接，+号主要实现方法就是使用StringBuilder的append方法进行追加，而它最终通过toString方法转换成String字符串：



这样，你如果继续再str后面拼接，还会创建新的对象，如果出现在for循环里面，后果不堪设想。一般来说直接先声明一个StringBuilder对象，然后用StringBuilder.append()方法即可。

**2、String内部存储**

先看代码：



* 对于使用字面量赋值方式。JVM为了节省空间，s2会首先查找JVM中是否有"cat"的字符串常量。如果已经存在，则直接返回该引用，而无需重新创建对象。
* 对象new创建方式，JVM将分配新空间。

再如下述的代码：



空间堆栈分配如下：

