

public final class String  
 implements java.io.Serializable, Comparable<String>,CharSequence {

/\*\*该值用于字符存储。 \*/  
private final char value[];  
  
/\*\* 缓存字符串的哈希码 \*/  
private int hash; // 默认为0

/\*\*  
 \* 初始化新创建的{@code String}对象，使其代表  
 \*一个空字符序列. 请注意，使用此构造函数是  
 \*不必要的，因为字符串是不可变的.  
 \*/  
public String() {  
 this.value = "".value;  
}

/\*\*  
 \* 初始化新创建的{@code String}对象，使其代表  
 \*与参数相同的字符序列;换句话说，  
 \*新创建的字符串是参数字符串的副本.  
 \*除非需要{@code original}的\*显式副本，否则使用此构造函数是不必要的，因为字符串是不可变的.  
 \*/  
public String(String original) {  
 this.value = original.value;  
 this.hash = original.hash;  
}

}

String}类表示字符串。所有Java程序中的字符串文字，例如“abc”是作为此类的实例实现。所以可以直接赋值String str=“abc”

被final 修饰，则具有final 的特性，不可变不可继承线程安全

Jdk源码注释：字符串是不变的;他们在创建后无法改变。字符串缓冲区支持可变字符串。因为String对象是不可变的，所以可以共享它们。 Java语言为字符串提供特殊支持  
\*连接运算符（＆nbsp; +＆nbsp;），以及转换  
\*其他对象到字符串。字符串连接已实现  
\*通过{@code StringBuilder}（或{@code StringBuffer}）  
\*类及其{@code append}方法。  
\*字符串转换是通过该方法实现的

String的核心点:

成员属性private final char value[]

String类的底层是基于char数组的，对String的操作就是对数组value的操作。如str.length()方法里返回的是数组value的长度value.length.

构造方法，equals方法就是比较2个数组每个位置的字符是否一样，subString截取就是根据数字索引位置截取等等。

构造方法

public String(String original) {  
 this.value = original.value;  
 this.hash = original.hash;  
}

可以看出通过String str= new String(“abc”)是复制了一个传入的对象，所以一般不用构造器方法生成String，直接用赋值方法。

String str1= “abc”； 在编译期，JVM会去常量池来查找是否存在“abc”，如果不存在，就在常量池中开辟一个空间来存储“abc”；如果存在，就不用新开辟空间。然后在栈内存中开辟一个名字为str1的空间，来存储“abc”在常量池中的地址值

String str2 = new String("abc") ;在编译阶段JVM先去常量池中查找是否存在“abc”，如果过不存在，则在常量池中开辟一个空间存储“abc”。在运行时期，通过String类的构造器在堆内存中new了一个空间，然后将String池中的“abc”复制一份存放到该堆空间中，在栈中开辟名字为str2的空间，存放堆中new出来的这个String对象的地址值。

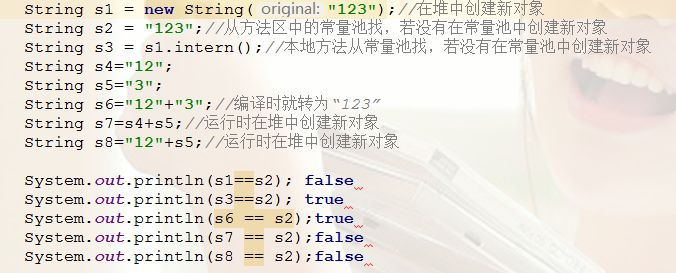
也就是说，前者在初始化的时候可能创建了一个对象，也可能一个对象也没有创建；后者因为new关键字，至少在内存中创建了一个对象，也有可能是两个对象。

String str = “hello";

str = str + "world“；

所以当上文str指向了一个String对象（内容为“hello”），然后对str进行“+”操作，str原来指向的对象并没有变，而是str又指向了另外一个对象（“hello world”），原来的对象还在内存中。

由此也可以看出，频繁的对String对象进行修改，会造成很大的内存开销。此时应该用StringBuffer或StringBuilder来代替String。



字符串的相加问题

1. 先看String s6="12"+"3"，网上给的解释是：在编译期间，这种拼接会被优化，编译器直接帮你拼好，所以s6相当于直接赋值为“123”，所以和上面的s2就一致了，直接去常量池中寻找到s2的内存地址，所以会出现s6==s2；
2. 而相加的过程中一旦出现了对象，就不会做优化，因为这是一个对象，内存不是确定的，没有写死，无法实现优化。而且在相加的过程中，java会先new出一个StringBuilder，然后调用append()方法来将+号两遍的字符串拼接起来，然后toString()之后返回给=号左边的变量，也就是说，最后得到的是一个new出来的字符串

StringBuffer：方法都有**synchronized**可变安全字符串

**public final class** StringBuffer  
 **extends** AbstractStringBuilder  
 **implements** java.io.Serializable, CharSequence  
{

**public** StringBuffer(String str) {  
 **super**(str.length() + 16);  
 append(str);  
}

@Override  
**public synchronized** StringBuffer append(String str) {  
 **toStringCache** = **null**;  
 **super**.append(str);  
 **return this**;  
}

}

StringBuilder：可变字符串方法没**synchronized**，单线程情况下代替StringBuffer

**public final class** StringBuilder  
 **extends** AbstractStringBuilder  
 **implements** java.io.Serializable, CharSequence  
{

**public** StringBuilder(String str) {  
 **super**(str.length() + 16);  
 append(str);  
}

@Override  
**public** StringBuilder append(String str) {  
 **super**.append(str);  
 **return this**;  
}

}