链滴

JVM_07 字符串常量池 StringTable

作者：wlgzs-sjl
原文链接：https://ld246.com/article/1603287122417
来源网站：链滴
许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

1.String的基本特性

- String：字符串，使用一对""引起来表示。
  - String sl = "hello"；//字面量的定义方式
  - String s2 = new String（"hello"）；
- String声明为final的， 不可被继承。
- String实现了Serializable接口：表示字符串是支持序列化的。实现了Comparable接口：表示Strin可以比较大小
- String在jdk8及以前内部定义了final char[],value用于存储字符串数据。jdk9时改为byte[]
  - 结论： String再也不用char[]来存储，改成了byte[] 加上编码标记，节约了一些空间。StringBuffer和StringBuilder也做了一些修改

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

@Stable
private final byte[] value;
}
```

- String：代表不可变的字符序列。简称：不可变性。
  - 当对字符串重新赋值时，需要重写指定内存区域赋值，不能使用原有的value进行赋值。
  - 当对现有的字符串进行连接操作时，也需要重新指定内存区域赋值，不能使用原有的value进行值。
  - 当调用String的replace（）方法修改指定字符或字符串时，也需要重新指定内存区域赋值，不使用原有的value进行赋值。
- 通过字面量的方式（区别于new）给一个字符串赋值，此时的字符串值声明在字符串常量池中。

- **字符串常量池中是不会存储相同内容的字符串的。**

  - String的String Pool 是一个固定大小的Hashtable，默认值大小长度是1009。如果放进StringPol的String非常多， 就会造成Hash冲突严重，从而导致链表会很长，而链表长了后直接会造成的影就是当调用String.intern时性能会大幅下降。
  - 使用一XX： StringTableSize可设置StringTable的长度。
  - 在jdk6中StringTable是固定的，就是1009的长度，所以如果常量池中的字符串过多就会导致效下降很快。StringTableSize设置没有要求。
  - 在jdk7中，StringTable的长度默认值是60013。
  - jdk8开始,1009是StringTable长度可设置的最小值。

2.String的内存分配

- 在Java语言中有8种基本数据类型和一种比较特殊的类型String。这些类型为了使它们在运行过程速度更快、更节省内存，都提供了一种常量池的概念。
- 常量池就类似一个Java系统级别提供的缓存。8种基本数据类型的常量池都是系统协调的，String型的常量池比较特殊。它的主要使用方法有两种。
  - 直接使用双引号声明出来的String对象会直接存储在常量池中。
    - 比如： String info = "abc" ；
  - 如果不是用双引号声明的String对象，可以使用String提供的intern（）方法。这个后面重点谈
- Java 6及以前，字符串常量池存放在永久代。
- Java 7中Oracle的工程师对字符串池的逻辑做了很大的改变，即将字符串常量池的位置调整到Java内。
  - 所有的字符串都保存在堆（Heap）中，和其他普通对象一样，这样可以让你在进行调优应用时需要调整堆大小就可以了。
  - 字符串常量池概念原本使用得比较多，但是这个改动使得我们有足够的理由让我们重新考虑在Jaa 7中使用String. intern（）。
- Java8元空间，字符串常量在堆

StringTable为什么要调整

①永久代permSize默认比较小;

②永久代的垃圾回收频率低;

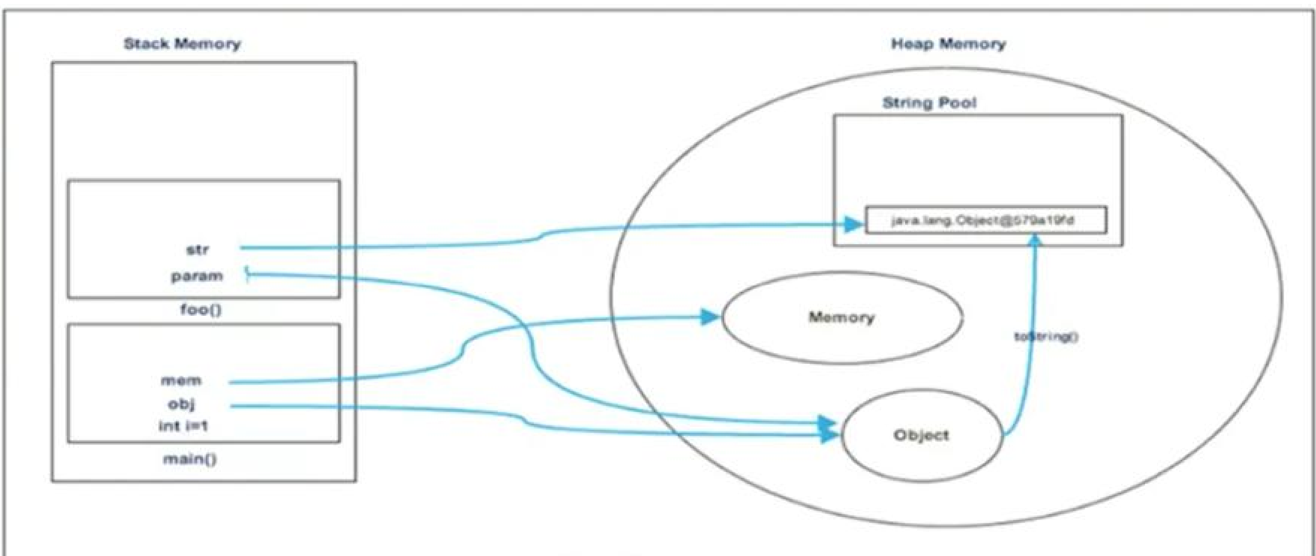
3.String的基本操作

原文链接：JVM_07 字符串常量池 StringTable

```java
@Test
public void test1() {
    System.out.println("1");//2321
    System.out.println("2");
    System.out.println("3");
    System.out.println("4");
    System.out.println("5");
    System.out.println("6");
    System.out.println("7");
    System.out.println("8");
    System.out.println("9");
    System.out.println("10");//2330
    System.out.println("1");//2331
    System.out.println("2");//2331
    System.out.println("3");
    System.out.println("4");
    System.out.println("5");
    System.out.println("6");
    System.out.println("7");
    System.out.println("8");
    System.out.println("9");
    System.out.println("10");//2331
}
```

Java语言规范里要求完全相同的字符串字面量，应该包含同样的Unicode字符序列(包含同一份码点序列的常量)，并且必须是指向同一个String类实例。

```java
class Memory {
    public static void main(String[] args) {//line 1
        int i = 1;//line 2
        Object obj = new Object();//line 3
        Memory mem = new Memory();//line 4
        mem.foo(obj);//line 5
    }//line 9

    private void foo(Object param) {//line 6
        String str = param.toString();//line 7
        System.out.println(str);
    }//line 8
}
```



4.字符串拼接操作

- 常量与常量的拼接结果在常量池，原理是编译期优化。

原文链接：JVM_07 字符串常量池 StringTable

- 常量池中不会存在相同内容的常量。
- **只要其中有一个是变量，结果就在堆中**。变量拼接的原理是StringBuilder
- 如果拼接的结果调用intern（）方法，则主动将常量池中还没有的 **字符串对象**放入池中，并返回对象地址。

```
 @Test
    public void test1(){
        String s1 = "a" + "b" + "c";//编译期优化：等同于"abc"
        String s2 = "abc"; //"abc"一定是放在字符串常量池中，将此地址赋给s2
        /*
         * 最终.java编译成.class,再执行.class
         * String s1 = "abc";
         * String s2 = "abc"
         */
        System.out.println(s1 == s2); //true
        System.out.println(s1.equals(s2)); //true
    }

    @Test
    public void test2(){
        String s1 = "javaEE";
        String s2 = "hadoop";

        String s3 = "javaEEhadoop";
        String s4 = "javaEE" + "hadoop";//编译期优化
        //如果拼接符号的前后出现了变量，则相当于在堆空间中new String()，具体的内容为拼接的结
：javaEEhadoop
        String s5 = s1 + "hadoop";
        String s6 = "javaEE" + s2;
        String s7 = s1 + s2;

        System.out.println(s3 == s4);//true
        System.out.println(s3 == s5);//false
        System.out.println(s3 == s6);//false
        System.out.println(s3 == s7);//false
        System.out.println(s5 == s6);//false
        System.out.println(s5 == s7);//false
        System.out.println(s6 == s7);//false
        //intern():判断字符串常量池中是否存在javaEEhadoop值，如果存在，则返回常量池中javaEEh
doop的地址；
        //如果字符串常量池中不存在javaEEhadoop，则在常量池中加载一份javaEEhadoop，并返回
对象的地址。
        String s8 = s6.intern();
        System.out.println(s3 == s8);//true
    }
```

字符串拼接

```
@Test
    public void test3(){
        String s1 = "a";
        String s2 = "b";
        String s3 = "ab";
```

```
        /*
        如下的s1 + s2 的执行细节：(变量s是我临时定义的）
        ① StringBuilder s = new StringBuilder();
        ② s.append("a")
        ③ s.append("b")
        ④ s.toString()  --> 约等于 new String("ab")

        补充：在jdk5.0之后使用的是StringBuilder,
        在jdk5.0之前使用的是StringBuffer
         */
        String s4 = s1 + s2;//
        System.out.println(s3 == s4);//false
    }

    /*
    1. 字符串拼接操作不一定使用的是StringBuilder!
       如果拼接符号左右两边都是字符串常量或常量引用，则仍然使用编译期优化，即非StringBuilder
方式。
    2. 针对于final修饰类、方法、基本数据类型、引用数据类型的量的结构时，能使用上final的时候
议使用上。
     */
    @Test
    public void test4(){
        final String s1 = "a";
        final String s2 = "b";
        String s3 = "ab";
        String s4 = s1 + s2;
        System.out.println(s3 == s4);//true
    }

    //练习：
    @Test
    public void test5(){
        String s1 = "javaEEhadoop";
        String s2 = "javaEE";
        String s3 = s2 + "hadoop";
        System.out.println(s1 == s3);//false

        final String s4 = "javaEE";//s4:常量
        String s5 = s4 + "hadoop";
        System.out.println(s1 == s5);//true

    }
```

拼接操作与append的效率对比

append效率要比字符串拼接高很多

```
/*
    体会执行效率：通过StringBuilder的append()的方式添加字符串的效率要远高于使用String的字
串拼接方式！
    详情：① StringBuilder的append()的方式：自始至终中只创建过一个StringBuilder的对象
          使用String的字符串拼接方式：创建过多个StringBuilder和String的对象
         ② 使用String的字符串拼接方式：内存中由于创建了较多的StringBuilder和String的对象，内
```

```
占用更大；如果进行GC，需要花费额外的时间。

     改进的空间：在实际开发中，如果基本确定要前前后后添加的字符串长度不高于某个限定值highLe
el的情况下,建议使用构造器实例化：
            StringBuilder s = new StringBuilder(highLevel);//new char[highLevel]
     */
    @Test
    public void test6(){

        long start = System.currentTimeMillis();

//        method1(100000);//4014
        method2(100000);//7

        long end = System.currentTimeMillis();

        System.out.println("花费的时间为：" + (end - start));
    }

    public void method1(int highLevel){
        String src = "";
        for(int i = 0;i < highLevel;i++){
            src = src + "a";//每次循环都会创建一个StringBuilder、String
        }
//        System.out.println(src);

    }

    public void method2(int highLevel){
        //只需要创建一个StringBuilder
        StringBuilder src = new StringBuilder();
        for (int i = 0; i < highLevel; i++) {
            src.append("a");
        }
//        System.out.println(src);
    }
```

5.intern()的使用

如果不是用双引号声明的String对象，可以使用String提供的intern方法： intern方法会从字符串常池中查询当前字符串是否存在，若不存在就会将当前字符串放入常量池中。

- 比如： String myInfo = new String("I love u").intern()；

也就是说，如果在任意字符串上调用String. intern方法，那么其返回结果所指向的那个类实例，必须直接以常量形式出现的字符串实例完全相同。因此，下列表达式的值必定是true：

（"a" + "b" + "c"）.intern（）== "abc";

通俗点讲，Interned String就是确保字符串在内存里只有一份拷贝，这样可以节约内存空间，加快字串操作任务的执行速度。注意，这个值会被存放在字符串内部池（String Intern Pool）。

new String("ab")会创建几个对象,new String("a")+new String(b")呢

```java
public class StringNewTest {
    public static void main(String[] args) {
//        String str = new String("ab");

        String str = new String("a") + new String("b");
    }
}
```

new String("ab")会创建几个对象？看字节码，就知道是两个。

- 一个对象是：new关键字在堆空间创建的
- 另一个对象是：字符串常量池中的对象"ab"。字节码指令：ldc

```
Bytecode    Exception table    Misc
1   0 new #2 <java/lang/String>
2   3 dup
3   4 ldc #3 <ab>
4   6 invokespecial #4 <java/lang/String.<init>>
5   9 astore_1
6  10 return
```

new String("a") + new String("b")呢?

- 对象1：new StringBuilder()
- 对象2： new String("a")
- 对象3： 常量池中的"a"
- 对象4： new String("b")
- 对象5： 常量池中的"b"

```
Bytecode    Exception table    Misc
1   0 new #2 <java/lang/StringBuilder>
2   3 dup
3   4 invokespecial #3 <java/lang/StringBuilder.<init>>
4   7 new #4 <java/lang/String>
5  10 dup
6  11 ldc #5 <a>
7  13 invokespecial #6 <java/lang/String.<init>>
8  16 invokevirtual #7 <java/lang/StringBuilder.append>
9  19 new #4 <java/lang/String>
10 22 dup
11 23 ldc #8 <b>
12 25 invokespecial #6 <java/lang/String.<init>>
13 28 invokevirtual #7 <java/lang/StringBuilder.append>
14 31 invokevirtual #9 <java/lang/StringBuilder.toString>
15 34 astore_1
16 35 return
```

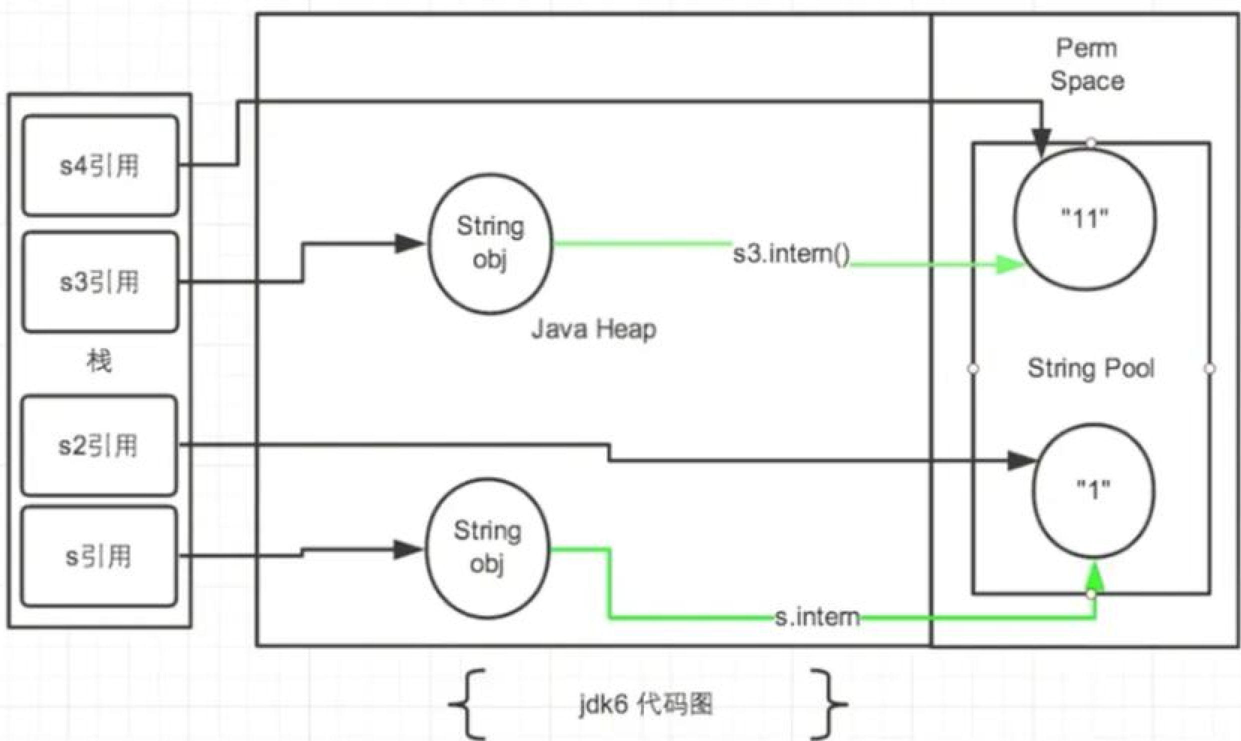
深入剖析： StringBuilder的toString():

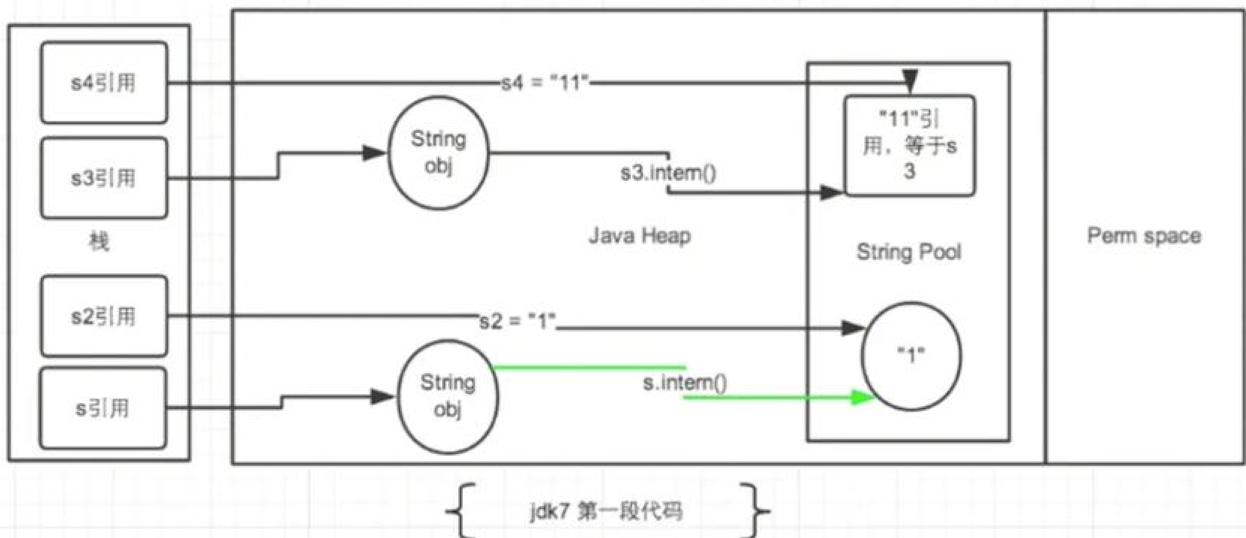
- 对象6 ： new String("ab")
- 强调一下，toString()的调用，在字符串常量池中，没有生成"ab"

关于String.intern()的面试题

```
/**
 * 如何保证变量s指向的是字符串常量池中的数据呢?
 * 有两种方式:
 * 方式一： String s = "shkstart";//字面量定义的方式
 * 方式二： 调用intern()
 *         String s = new String("shkstart").intern();
 *         String s = new StringBuilder("shkstart").toString().intern();
 *
 */
public class StringIntern {
    public static void main(String[] args) {
        String s = new String("1");
        String s1 = s.intern();//调用此方法之前，字符串常量池中已经存在了"1"
        String s2 = "1";
        //s  指向堆空间"1"的内存地址
        //s1 指向字符串常量池中"1"的内存地址
        //s2 指向字符串常量池已存在的"1"的内存地址  所以 s1==s2
        System.out.println(s == s2);//jdk6：false   jdk7/8：false
        System.out.println(s1 == s2);//jdk6: true   jdk7/8：true
        System.out.println(System.identityHashCode(s));//491044090
```

```
        System.out.println(System.identityHashCode(s1));//644117698
        System.out.println(System.identityHashCode(s2));//644117698

        //s3变量记录的地址为：new String("11")
        String s3 = new String("1") + new String("1");
        //执行完上一行代码以后，字符串常量池中，是否存在"11"呢？答案：不存在！！

        //在字符串常量池中生成"11"。如何理解：jdk6:创建了一个新的对象"11",也就有新的地址。
        //         jdk7:此时常量中并没有创建"11",而是创建一个指向堆空间中new String("11")的地址
        s3.intern();
        //s4变量记录的地址：使用的是上一行代码代码执行时，在常量池中生成的"11"的地址
        String s4 = "11";
        System.out.println(s3 == s4);//jdk6：false  jdk7/8：true
    }

}
```





原文链接：JVM_07 字符串常量池 StringTable

拓展

```
public class StringIntern1 {
    public static void main(String[] args) {
        //StringIntern.java中练习的拓展:
        String s3 = new String("1") + new String("1");//new String("11")
        //执行完上一行代码以后，字符串常量池中，是否存在"11"呢？答案：不存在！！
        String s4 = "11";//在字符串常量池中生成对象"11"
        String s5 = s3.intern();
        System.out.println(s3 == s4);//false
        System.out.println(s5 == s4);//true
    }
}
```

总结String的intern（）的使用

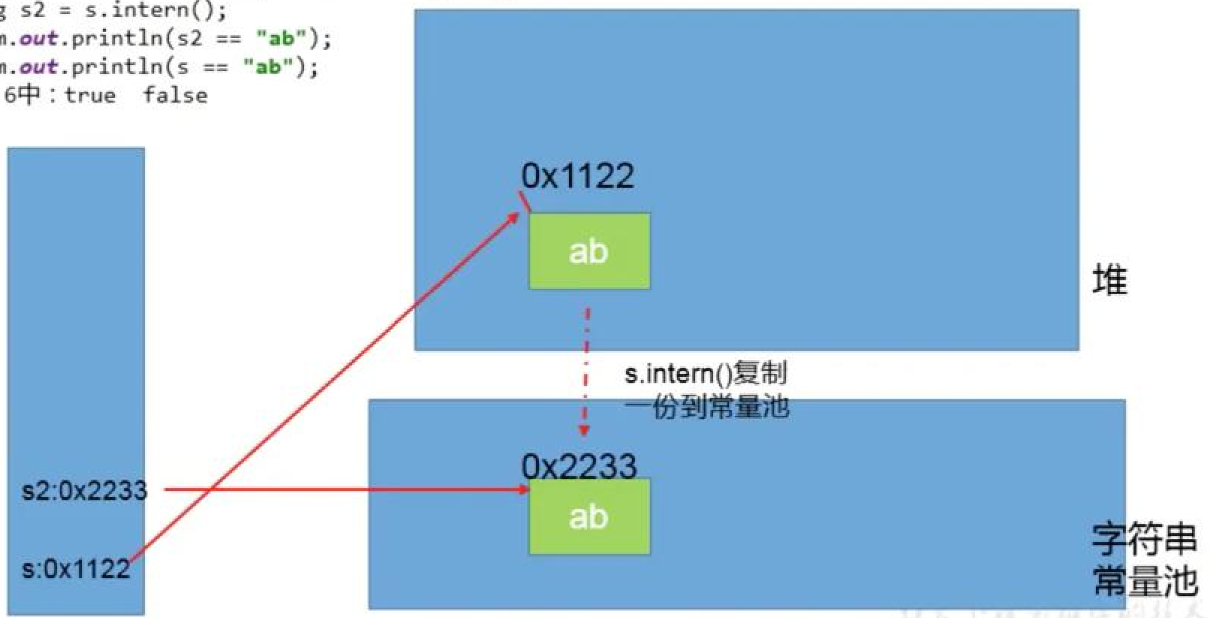
- jdk1.6中，将这个字符串对象尝试放入串池。
  - 如果字符串常量池中有，则并不会放入。返回已有的串池中的对象的地址。
  - 如果没有，会把此对象复制一份，放入串池，并返回串池中的对象地址。
- jdk1.7起，将这个字符串对象尝试放入串池。
  - 如果字符串常量池中有，则并不会放入。返回已有的串池中的对象的地址。
  - 如果没有，则会把对象的引用地址复制一份，放入串池，并返回串池中的 **引用地址。**

练习

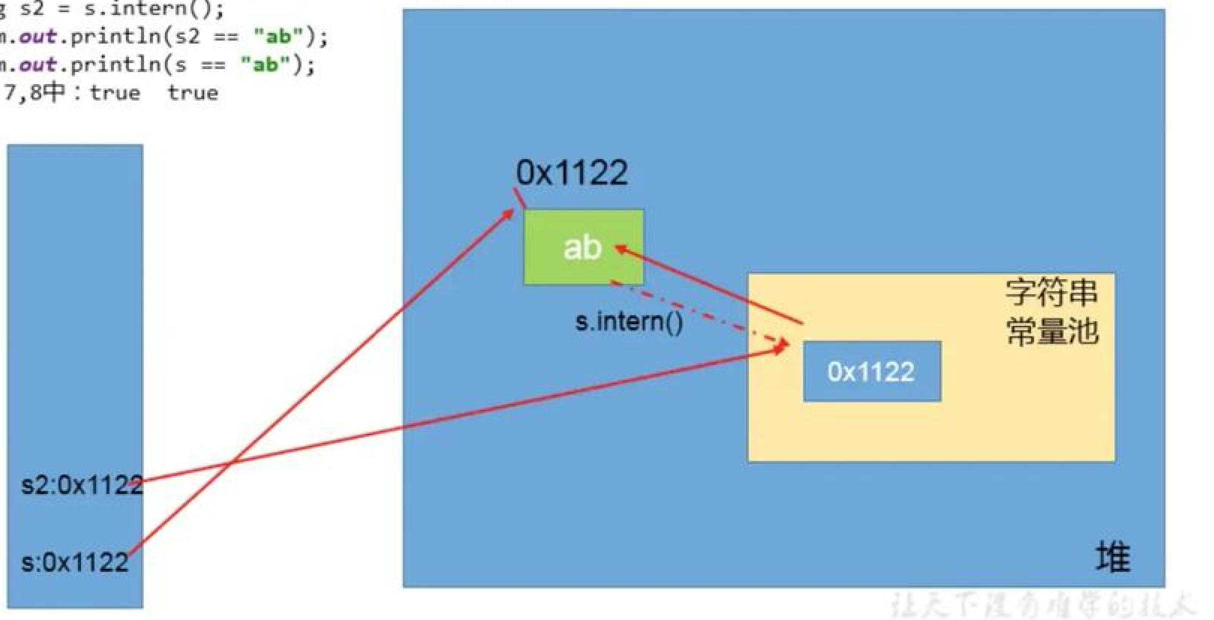
练习1

```
public class StringExer1 {
    public static void main(String[] args) {
        //String x = "ab";
        String s = new String("a") + new String("b");//new String("ab")
        //在上一行代码执行完以后，字符串常量池中并没有"ab"

        String s2 = s.intern();//jdk6中：在串池中创建一个字符串"ab"
                               //jdk8中：串池中没有创建字符串"ab",而是创建一个引用，指向new String("ab"
，将此引用返回

        System.out.println(s2 == "ab");//jdk6:true  jdk8:true
        System.out.println(s == "ab");//jdk6:false  jdk8:true
    }
}
```

jdk6



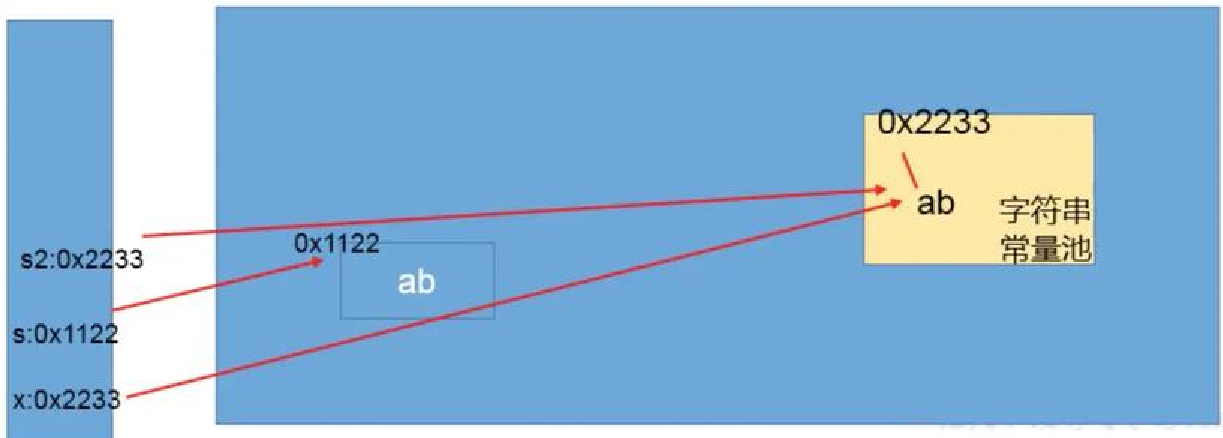
jdk7/8



原文链接：JVM_07 字符串常量池 StringTable

```java
String x = "ab";
String s = new String("a") + new String("b");
String s2 = s.intern();//因为常量池已经有ab,则不会放入
System.out.println(s2 == x);
System.out.println(s == x);
```

在jdk6,7,8中执行都是true false



练习2

```java
public class StringExer2 {
    public static void main(String[] args) {
        String s1 = new String("ab");//执行完以后，会在字符串常量池中会生成"ab"
//        String s1 = new String("a") + new String("b");////执行完以后，不会在字符串常量池中会
成"ab"
        s1.intern();
        String s2 = "ab";
        System.out.println(s1 == s2); //false
    }
}
```

intern()效率测试

大的网站平台，需要内存中存储大量的字符串。比如社交网站，很多人都存储：北京市、海淀区等信息。这时候如果字符串都调用 intern（）方法，就会明显降低内存的大小。

```java
/**
 * 使用intern()测试执行效率：空间使用上
 *
 * 结论：对于程序中大量存在存在的字符串，尤其其中存在很多重复字符串时，使用intern()可以节
内存空间。
 *
 */
public class StringIntern2 {
    static final int MAX_COUNT = 1000 * 10000;
    static final String[] arr = new String[MAX_COUNT];

    public static void main(String[] args) {
        Integer[] data = new Integer[]{1,2,3,4,5,6,7,8,9,10};

        long start = System.currentTimeMillis();
```

```
        for (int i = 0; i < MAX_COUNT; i++) {
//          arr[i] = new String(String.valueOf(data[i % data.length]));
            arr[i] = new String(String.valueOf(data[i % data.length])).intern();

        }
        long end = System.currentTimeMillis();
        System.out.println("花费的时间为： " + (end - start));

        try {
            Thread.sleep(1000000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.gc();
    }
}
```

6.G1中的String去重操作

- 背景：对许多Java应用（有大的也有小的）做的测试得出以下结果：
  - 堆存活数据集合里面String对象占了25%
  - 堆存活数据集合里面重复的String对象有13.5%
  - String对象的平均长度是45
- 许多大规模的Java应用的瓶颈在于内存，测试表明，在这些类型的应用里面，Java堆中存活的数据合差不多258是String对象。更进一步，这里面差不多一半String对象是重复的，重复的意思是说：string1. equals （string2）=true。堆上存在重复的string对象必然是一种内存的浪费。这个项目将在G垃圾收集器中实现自动持续对重复的String对象进行去重，这样就能避免浪费内存。

实现

- 当垃圾收集器工作的时候，会访问堆上存活的对象。对每一个访问的对象都会检查是否是候选的要重的String对象。
- 如果是，把这个对象的一个引用插入到队列中等待后续的处理。一个去重的线程在后台运行，处理个队列。处理队列的一个元素意味着从队列删除这个元素，然后尝试去重它引用的String对象。
- 使用一个hashtable来记录所有的被String对象使用的不重复的char数组。

当去重的时候，会查这个hashtable，来看堆上是否已经存在一个一模一样的char数组。

- 如果存在，String对象会被调整引用那个数组，释放对原来的数组的引用，最终会被垃圾收集器回掉。
- 如果查找失败，char数组会被插入到hashtable，这样以后的时候就可以共享这个数组了。

命令行选项

- □UseStringDeduplication （bool） ：开启String去重，默认是不开启的，需要手动开启。
- □PrintStringDedupl icationStatistics （bool） ：打印详细的去重统计信息，
- □StringDedupl icationAgeThreshold （uintx） ：达到这个年龄的string对象被认为是去重的候对象

原文链接：JVM_07 字符串常量池 StringTable