


第5讲 | String、StringBuffer、StringBuilder有什么区别？

2018-05-15 杨晓峰





第5讲 | String、StringBuffer、StringBuilder有什么区别？

杨晓峰

- 00:00 / 11:59

今天我会聊聊日常使用的字符串，别看它似乎很简单，但其实字符串几乎在所有编程语言里都是个特殊的存在，因为不管是数量还是体积，字符串都是大多数应用中的重要组成。

今天我要问你的问题是，[理解Java的字符串](#)，[String、StringBuffer、StringBuilder有什么区别？](#)

典型回答

String是Java语言非常基础和重要的类，提供了构造和管理字符串的各种基本逻辑。它是典型的Immutable类，被声明成为final class，所有属性也都是final的。也由于它的不可变性，类似拼接、裁剪字符串等动作，都会产生新的String对象。由于字符串操作的普遍性，所以相关操作的效率往往对应用性能有明显影响。

StringBuffer是为解决上面提到拼接产生太多中间对象的问题而提供的一个类，我们可以用append或者add方法，把字符串添加到已有序列的末尾或者指定位置。StringBuffer本质是一个线程安全的可修改字符序列，它保证了线程安全，也随之带来了额外的性能开销，所以除非有线程安全的需要，不然还是推荐使用它的后继者，也就是StringBuilder。

StringBuilder是Java 1.5中新增的，在能力上和StringBuffer没有本质区别，但是它去掉了线程安全的部分，有效减小了开销，是绝大部分情况下进行字符串拼接的首选。

考点分析

几乎所有的应用开发都离不开操作字符串，理解字符串的设计和实现以及相关工具如拼接类的使用，对写出高质量代码是非常有帮助的。关于这个问题，我前面的回答是一个通常的概要性回答，至少你要知道String是Immutable的，字符串操作不当可能会产生大量临时字符串，以及线程安全方面的区别。

如果继续深入，面试官可以从各种不同的角度考察，比如可以：

- 通过String和相关类，考察基本的线程安全设计与实现，各种基础编程实践。
- 考察JVM对象缓存机制的理解以及如何良好地使用。
- 考察JVM优化Java代码的一些技巧。
- String相关类的演进，比如Java 9中实现的巨大变化。
- ...

针对上面这几方面，我会在知识扩展部分与你详细聊聊。

知识扩展

1. 字符串设计和实现考量

我在前面介绍过，String是Immutable类的典型实现，原生的保证了基础线程安全，因为你无法对它内部数据进行任何修改，这种便利甚至体现在拷贝构造函数中，由于不可变，Immutable对象在拷贝时不需要额外复制数据。

我们再来看看StringBuffer实现的一些细节，它的线程安全是通过把各种修改数据的方法都加上synchronized关键字实现的，非常直白。其实，这种简单粗暴的实现方式，非常适合我们常见的线程安全类实现，不必纠结于synchronized性能之类的，有人说“过早优化是万恶之源”，考虑可靠性、正确性和代码可读性才是大多数应用开发最重要的因素。

为了实现修改字符序列的目的，StringBuffer和StringBuilder底层都是利用可修改的（char，JDK 9以后是byte）数组，二者都继承了AbstractStringBuilder，里面包含了基本操作，区别仅在于最终的方法是否加上了synchronized。

另外，这个内部数组应该创建成多大的呢？如果太小，拼接的时候可能要重新创建足够大的数组；如果太大，又会浪费空间。目前的实现是，构建时初始字符串长度加16（这意味着，如果没有构建对象时输入最初的字符串，那么初始值就是16）。如果我们确定拼接会发生非常多次，而且大概是可预计的，那么就可以指定合适的大小，避免很多次扩容的开销。扩容会产生多重开销，因为要抛弃原有数组，创建新的（可以简单认为是倍数）数组，还要进行arraycopy。

前面我讲的这些内容，在具体的代码书写中，应该如何选择呢？

在没有线程安全问题的情况下，全部拼接操作是都应该用StringBuider实现吗？毕竟这样书写的代码，还是要多敲很多字的，可读性也不理想，下面的对比非常明显。

```
String &myBuilder = new
StringBuilder().append("aa").append("bb").append("cc").append
("dd").toString();

String &myConcat = "aa" + "bb" + "cc" + "dd";
```

其实，在通常情况下，没有必要过于担心，要相信Java还是非常智能的。

我们来做个实验，把下面一段代码，利用不同版本的JDK编译，然后再反编译，例如：

```
public class StringConcat {
    public static void main(String[] args) {
        String myStr = "aa" + "bb" + "cc" + "dd";
        System.out.println("My String:" + myStr);
    }
}
```

先编译再反编译，比如使用JDK 9：

```
{JDK9_HOME}/bin/javac StringConcat.java

{JDK9_HOME}/bin/javap -v StringConcat.class
```

JDK 8的输出片段是：

```
6: new          #4          // class java/lang/StringBuilder
9: dup
10: invokespecial #5          // Method java/lang/StringBuilder.<init>:()V
13: ldc          #6           // String My String:
15: invokevirtual #7          // Method java/lang/StringBuilder.append:(Ljava/lang/String;)Ljava/lang/StringBuilder;
18: aload_1
19: invokevirtual #7          // Method java/lang/StringBuilder.append:(Ljava/lang/String;)Ljava/lang/StringBuilder;
22: invokevirtual #8          // Method java/lang/StringBuilder.toString:()Ljava/lang/String;
```

而在JDK 9中，反编译的结果就非常简单了，片段是：

```
7: invokedynamic #4, 0       // InvokeDynamic #0:makeConcatWithConstants:(Ljava/lang/String;)Ljava/lang/String;
```

你可以看到，在JDK 8中，字符串拼接操作会自动被javac转换为StringBuilder操作，而在JDK 9里面则是因为Java 9为了更加统一字符串操作优化，提供了StringConcatFactory，作为一个统一的入口。javac自动生成的代码，虽然未必是最优化的，但普通场景也足够了，你可以酌情选择。

## 2.字符串缓存

我们粗略统计过，把常见应用进行堆转储（Dump Heap），然后分析对象组成，会发现平均25%的对象是字符串，并且其中约半数重复的。如果能避免创建重复字符串，可以有效降低内存消耗和对象创建开销。

String在Java 6以后提供了intern()方法，目的是提示JVM把相应字符串缓存起来，以备重复使用。在我们创建字符串对象并调用intern()方法的时候，如果已经有缓存的字符串，就会返回缓存里的实例，否则将其缓存起来。一般来说，JVM会将所有的类似“abc”这样的文本字符串，或者字符串常量之类缓存起来。

看起来很不错是吧？但实际情况估计会让你大跌眼镜。一般使用Java 6这种历史版本，并不推荐大量使用intern，为什么呢？魔鬼存在于细节中，被缓存的字符串是存在所谓PermGen里的，也就是臭名昭著的“永久代”，这个空间是很有限的，也基本不会被FullGC之外的垃圾收集照顾到。所以，如果使用不当，OOM就会光顾。

在后续版本中，这个缓存被放置在堆中，这样就极大避免了永久代占满的问题，甚至永久代在JDK 8中被MetaSpace（元数据区）替代了。而且，默认缓存大小也在不断地扩大中，从最初的1009，到7u40以后被修改为60013。你可以使用下面的参数直接打印具体数字，可以拿自己的JDK立刻试验一下。

```
-XX:+PrintStringTableStatistics
```

你也可以使用下面的JVM参数手动调整大小，但是绝大部分情况下并不需要调整，除非你确定它的大小已经影响了操作效率。

```
-XX:StringTableSize=N
```

Intern是一种显式地排重机制，但是它也有一定的副作用，因为需要开发者写代码时明确调用，一是不方便，每一个都显式调用是非常麻烦的；另外就是我们很难保证效率，应用开发阶段很难清楚地预计字符串的重复情况，有人认为这是一种污染代码的实践。

幸好在Oracle JDK 8u20之后，推出了一个新的特性，也就是G1 GC下的字符串排重。它是通过将相同数据的字符串指向同一份数据来做到的，是JVM底层的改变，并不需要Java类库做什么修改。

注意这个功能目前是默认关闭的，你需要使用下面参数开启，并且记得指定使用G1 GC：

```
-XX:+UseStringDeduplication
```

前面说到的几个方面，只是Java底层对字符串各种优化的一角，在运行时，字符串的一些基础操作会直接利用JVM内部的Intrinsic机制，往往运行的就是特殊优化的本地代码，而根本就不是Java代码生成的字节码。Intrinsic可以简单理解为，是一种利用native方式hard-coded的逻辑，算是一种特别的内联，很多优化还是需要直接使用特定的CPU指令，具体可以看相关[源码](#)，搜索“string”以查找相关Intrinsic定义。当然，你也可以在启动实验应用时，使用下面参数，了解intrinsic发生的状态。

```
-XX:+PrintCompilation -XX:+UnlockDiagnosticVMOptions -XX:+PrintInlining
//详细输出片段
188   3       3       java.lang.String::charAt (25 bytes)
                        @ 1   java.lang.String::isLatin1 (19 bytes)
                        ...
                        @ 7   java.lang.StringUTF16::getChar (60 bytes) intrinsic
```

可以看出，仅仅是字符串一个实现，就需要Java平台工程师和科学家付出如此大且默默无闻的努力，我们得到的很多便利都是来源于此。

我会在专栏后面的JVM和性能等主题，详细介绍JVM内部优化的一些方法，如果你有兴趣可以再深入学习。即使你不做JVM开发或者暂时还没有使用到特别的性能优化，这些知识也能帮助你增加技术深度。

3. String自身的演化

如果你仔细观察过Java的字符串，在历史版本中，它是使用char数组来存数据的，这样非常直接。但是Java中的char是两个bytes大小，拉丁语系语言的字符，根本就不需要太宽的char，这样无区别的的实现就造成了一定的浪费。密度是编程语言平台永恒的话题，因为归根结底绝大部分任务是要来操作数据的。

其实在Java 6的时候，Oracle JDK就提供了压缩字符串的特性，但是这个特性的实现并不是开源的，而且在实践中也暴露出了一些问题，所以在最新的JDK版本中已经将它移除了。

在Java 9中，我们引入了Compact Strings的设计，对字符串进行了大刀阔斧的改进。将数据存储方式从char数组，改变为一个byte数组加上一个标识编码的所谓coder，并且将相关字符串操作类都进行了修改。另外，所有相关的Intrinsic之类也都进行了重写，以保证没有任何性能损失。

虽然底层实现发生了这么大的改变，但是Java字符串的行为并没有任何大的变化，所以这个特性对于绝大部分应用来说是透明的，绝大部分情况不需要修改已有代码。

当然，在极端情况下，字符串也出现了一些能力退化，比如最大字符串的大小。你可以思考下，原来char数组的实现，字符串的最大长度就是数组本身的长度限制，但是替换成byte数组，同样数组长度下，存储能力是退化了一倍的！还好这是存在于理论中的极限，还没有发现现实应用受此影响。

在通用的性能测试和产品实验中，我们能非常明显地看到紧凑字符串带来的优势，即更小的内存占用、更快的操作速度。

今天我从String、StringBuffer和StringBuilder的主要设计和实现特点开始，分析了字符串缓存的intern机制、非代码侵入性的虚拟机层面排重。Java 9中紧凑字符的改进，并且初步接触了JVM的底层优化机制Intrinsic，从实践的角度，不管是Compact Strings还是底层Intrinsic优化，都说明了使用Java基础类库的优势，它们往往能够得到最大程度、最高质量的优化，而且只要升级JDK版本，就能零成本地享受这些益处。

一课一练

关于今天我们讨论的题目你做到心中有数了吗？限于篇幅有限，还有很多字符相关的问题没有来得及讨论，比如编码相关的问题。可以思考一下，很多字符串操作，比如getBytes()/String(byte[] bytes)等都是隐含着使用平台默认编码，这是一种好的实践吗？是否有利于避免乱码？

请在留言区写写你对这个问题的思考，或者分享一下你在操作字符串时掉过的坑，我会选出经过认真思考的留言，送给你一份学习鼓励金，欢迎你与我一起讨论。

你的朋友是不是也在准备面试呢？你可以“请朋友读”，把今天的题目分享给好友，或许你能帮到他。



Bin

2018-05-16

jdk1.8中，string是标准的不可变类，但其hash值没有用final修饰，其hash值计算是在第一次调用hashCode方法时计算，但方法没有加锁，变量也没用volatile关键字修饰就无法保证其可见性。当有多个线程调用的时候，hash值可能会被计算多次，虽然结果是一样的，但jdk的作者为什么不将其优化一下呢？

作者回复

2018-05-17

这些“优化”在通用场景可能变成持续的成本，volatile read是有明显开销的；如果冲突并不多见，read才是更普遍的，简单的cache是更高效的

公号-Java大后端

2018-05-15

String/StringBuffer/StringBuilder

：

今日	心得																																																												
1 String																																																													
<p>(1) String的创建机理</p> <p>由于String在Java世界中使用过于频繁，Java为了避免在一个系统中产生大量的String对象，引入了字符串常量池。其运行机制是：创建一个字符串时，首先检查池中是否有值相同的字符串对象，如果有则不需要创建直接从池中刚查找到的对象引用；如果没有则新建字符串对象，返回对象引用，并且将新创建的对象放入池中。但是，通过new方法创建的String对象是不检查字符串池的，而是直接在堆区或栈区创建一个新的对象，也不会把对象放入池中。上述原则只适用于通过直接赋值给String对象引用赋值的情况。</p> <p>举例：String str1 = "123"; //通过直接量赋值方式，放入字符串常量池 String str2 = new String("123");//通过new方式赋值方式，不放入字符串常量池</p> <p>注意：String提供了inter()方法。调用该方法时，如果常量池中包括了一个等于此String对象的字符串（由equals方法确定），则返回池中的字符串。否则，将此String对象添加到池中，并且返回此池中对象的引用。</p> <p>(2) String的特性</p> <p>[A] 不可变。是指String对象一旦生成，则不能再对它进行改变。不可变的主要作用在于当一个对象需要被多线程共享，并且访问频繁时，可以省略同步和锁等待的时间，从而大幅度提高系统性能。不可变模式是一个可以提高多线程程序的性能，降低多线程程序复杂度的设计模式。</p> <p>[B] 针对常量池的优化。当2个String对象拥有相同的值时，他们只引用常量池中的同一个拷贝。当同一个字符串反复出现时，这个技术可以大幅度节省内存空间。</p> <p>2 StringBuffer/StringBuilder</p> <p>StringBuffer和StringBuilder都实现了AbstractStringBuilder抽象类，拥有几乎一致对外提供的调用接口；其底层在内存中的存储方式与String相同，都是以一个有序的字符序列（char类型的数组）进行存储，不同点是StringBuffer/StringBuilder对象的值是可以改变的，并且值改变以后，对象引用不会发生改变；两者对象在构造过程中，首先按照默认大小申请一个字符数组，由于会不断加入新数据，当超过默认大小后，会创建一个更大的数组，并将原先的数组内容复制过来，再去弃旧的数组。因此，对于较大对象的扩容会涉及大量的内存复制操作，如果能够预先评估大小，可提升性能。</p> <p>唯一需要注意的是：StringBuffer是线程安全的，但是StringBuilder是线程不安全的。可参看Java标准类库的源代码，StringBuffer类中方法定义前面都会有synchronize关键字。为此，StringBuffer的性能要远低于StringBuilder。</p> <p>3 应用场景</p> <p>[A]在字符串内容不经常发生变化的业务场景优先使用String类。例如：常量声明、少量的字符串拼接操作等。如果有大量的字符串内容拼接，避免使用String与String之间的"+"操作，因为这样会产生大量无用的中间对象，耗费空间且执行效率低下（新建对象、回收对象花费大量时间）。</p> <p>[B]在频繁进行字符串的运算（如拼接、替换、删除等），并且运行在多线程环境下，建议使用StringBuffer，例如XML解析、HTTP参数解析与封装。</p> <p>[C]在频繁进行字符串的运算（如拼接、替换、删除等），并且运行在单线程环境下，建议使用StringBuilder，例如SQL语句拼装、JSON封装等。</p> <p>作者回复</p> <p>很到位</p> <p>2018-05-15</p> <tr><td>Hidden</td><td>2018-05-16</td></tr> <tr><td colspan="2">公司没有技术氛围，项目也只是 功能实现就好，不涉及优化，技术也只是传统技术，想离职，但又怕裸辞后的各种压力</td></tr> <tr><td colspan="2">sea季陪我去看海</td></tr> <tr><td colspan="2">2018-05-16</td></tr> <tr><td colspan="2">作者我有个疑问，String myStr = "aa" + "bb" + "cc" + "dd"; 应该编译的时候就确定了，不会用到StringBuilder。理由是： String myStr = "aa" + "bb" + "cc" + "dd"; String h =aabbccdd Mystr ==h 上机实测返回的是true，如果按照你的说法，应该是返回false才对，因为你说拼接到stringbuilder，那mystr应该是堆地址，h是常量池地址。</td></tr> <tr><td>KongkOng</td><td>2018-05-16</td></tr> <tr><td colspan="2">编译器为什么不把 String myStr = "aa" + "bb" + "cc" + "dd"; 默认优化成 String myStr = "aabbccdd"; 这样不是更聪明嘛</td></tr> <tr><td colspan="2">愉悦在花香的日子里</td></tr> <tr><td colspan="2">2018-05-15</td></tr> <tr><td colspan="2">getBytes和String相关的转换时根据业务需要建议指定编码方式，如果不指定则看看JVM参数里有没有指定file.encoding参数，如果JVM没有指定，那使用的默认编码就是运行的操作系统环境的编码了，那个编码就变得不确定了。常见的编码iso8859-1是单字节编码，UTF-8是变长的编码。</td></tr> <tr><td colspan="2">作者回复</td></tr> <tr><td colspan="2">2018-05-15</td></tr> <tr><td colspan="2">不错，莫依赖于不确定因素</td></tr> <tr><td colspan="2">肖一林</td></tr> <tr><td colspan="2">2018-05-15</td></tr> <tr><td colspan="2">这篇文章写的不错，由浅入深，把来龙去脉写清楚了</td></tr> <tr><td colspan="2">作者回复</td></tr> <tr><td colspan="2">2018-05-15</td></tr> <tr><td colspan="2">谢谢认可</td></tr> <tr><td colspan="2">薛好运</td></tr> <tr><td colspan="2">2018-05-15</td></tr> <tr><td colspan="2">老师，可以讲解这一句话的具体含义吗，谢谢！ 你可以思考下，原来 char 数组的实现，字符串的最大长度就是数组本身的长度限制，但是替换成 byte 数组，同样数组长度下，存储能力是退化了一倍的！还好这是存在于理论中的极限，还没有发现现实应用受此影响。</td></tr> <tr><td colspan="2">作者回复</td></tr> <tr><td colspan="2">2018-05-15</td></tr> <tr><td colspan="2">已回复，一个char两个byte，注意下各个类型宽度</td></tr> <tr><td>Jerry很很</td><td>2018-05-15</td></tr> <tr><td colspan="2">要完全消化一篇文章的所有内容，真得不是一天就能搞定的，可能需要一个月，甚至好几个月。就比如今天的字符串，我觉得这个话题覆盖的面太广：算法角度，易用角度，性能角度；编码传输角度等</td></tr> <tr><td colspan="2">但是好在，我获得见识。接下来，花时间慢慢研究呗，连大师们都花了那么多时间研究，我们多花点时间，很正常嘛💎💎</td></tr> <tr><td colspan="2">一点学习心得，和大家分享</td></tr> <tr><td>Jerry很很</td><td>2018-05-15</td></tr>		Hidden	2018-05-16	公司没有技术氛围，项目也只是 功能实现就好，不涉及优化，技术也只是传统技术，想离职，但又怕裸辞后的各种压力		sea季陪我去看海		2018-05-16		作者我有个疑问，String myStr = "aa" + "bb" + "cc" + "dd"; 应该编译的时候就确定了，不会用到StringBuilder。理由是： String myStr = "aa" + "bb" + "cc" + "dd"; String h =aabbccdd Mystr ==h 上机实测返回的是true，如果按照你的说法，应该是返回false才对，因为你说拼接到stringbuilder，那mystr应该是堆地址，h是常量池地址。		KongkOng	2018-05-16	编译器为什么不把 String myStr = "aa" + "bb" + "cc" + "dd"; 默认优化成 String myStr = "aabbccdd"; 这样不是更聪明嘛		愉悦在花香的日子里		2018-05-15		getBytes和String相关的转换时根据业务需要建议指定编码方式，如果不指定则看看JVM参数里有没有指定file.encoding参数，如果JVM没有指定，那使用的默认编码就是运行的操作系统环境的编码了，那个编码就变得不确定了。常见的编码iso8859-1是单字节编码，UTF-8是变长的编码。		作者回复		2018-05-15		不错，莫依赖于不确定因素		肖一林		2018-05-15		这篇文章写的不错，由浅入深，把来龙去脉写清楚了		作者回复		2018-05-15		谢谢认可		薛好运		2018-05-15		老师，可以讲解这一句话的具体含义吗，谢谢！ 你可以思考下，原来 char 数组的实现，字符串的最大长度就是数组本身的长度限制，但是替换成 byte 数组，同样数组长度下，存储能力是退化了一倍的！还好这是存在于理论中的极限，还没有发现现实应用受此影响。		作者回复		2018-05-15		已回复，一个char两个byte，注意下各个类型宽度		Jerry很很	2018-05-15	要完全消化一篇文章的所有内容，真得不是一天就能搞定的，可能需要一个月，甚至好几个月。就比如今天的字符串，我觉得这个话题覆盖的面太广：算法角度，易用角度，性能角度；编码传输角度等		但是好在，我获得见识。接下来，花时间慢慢研究呗，连大师们都花了那么多时间研究，我们多花点时间，很正常嘛💎💎		一点学习心得，和大家分享		Jerry很很	2018-05-15
Hidden	2018-05-16																																																												
公司没有技术氛围，项目也只是 功能实现就好，不涉及优化，技术也只是传统技术，想离职，但又怕裸辞后的各种压力																																																													
sea季陪我去看海																																																													
2018-05-16																																																													
作者我有个疑问，String myStr = "aa" + "bb" + "cc" + "dd"; 应该编译的时候就确定了，不会用到StringBuilder。理由是： String myStr = "aa" + "bb" + "cc" + "dd"; String h =aabbccdd Mystr ==h 上机实测返回的是true，如果按照你的说法，应该是返回false才对，因为你说拼接到stringbuilder，那mystr应该是堆地址，h是常量池地址。																																																													
KongkOng	2018-05-16																																																												
编译器为什么不把 String myStr = "aa" + "bb" + "cc" + "dd"; 默认优化成 String myStr = "aabbccdd"; 这样不是更聪明嘛																																																													
愉悦在花香的日子里																																																													
2018-05-15																																																													
getBytes和String相关的转换时根据业务需要建议指定编码方式，如果不指定则看看JVM参数里有没有指定file.encoding参数，如果JVM没有指定，那使用的默认编码就是运行的操作系统环境的编码了，那个编码就变得不确定了。常见的编码iso8859-1是单字节编码，UTF-8是变长的编码。																																																													
作者回复																																																													
2018-05-15																																																													
不错，莫依赖于不确定因素																																																													
肖一林																																																													
2018-05-15																																																													
这篇文章写的不错，由浅入深，把来龙去脉写清楚了																																																													
作者回复																																																													
2018-05-15																																																													
谢谢认可																																																													
薛好运																																																													
2018-05-15																																																													
老师，可以讲解这一句话的具体含义吗，谢谢！ 你可以思考下，原来 char 数组的实现，字符串的最大长度就是数组本身的长度限制，但是替换成 byte 数组，同样数组长度下，存储能力是退化了一倍的！还好这是存在于理论中的极限，还没有发现现实应用受此影响。																																																													
作者回复																																																													
2018-05-15																																																													
已回复，一个char两个byte，注意下各个类型宽度																																																													
Jerry很很	2018-05-15																																																												
要完全消化一篇文章的所有内容，真得不是一天就能搞定的，可能需要一个月，甚至好几个月。就比如今天的字符串，我觉得这个话题覆盖的面太广：算法角度，易用角度，性能角度；编码传输角度等																																																													
但是好在，我获得见识。接下来，花时间慢慢研究呗，连大师们都花了那么多时间研究，我们多花点时间，很正常嘛💎💎																																																													
一点学习心得，和大家分享																																																													
Jerry很很	2018-05-15																																																												

特别喜欢这句话：“仅仅是字符串一个实现，就需要 Java 平台工程师和科学家付出如此大且默默无闻的努力，我们得到的很多便利都是来源于此。”

我想说，同学们，写代码的时候记得感恩哦💎💎

对于字符串的研究，我觉得能很好的理解计算机的本质和训练计算机思维，提升自己解决问题的能力。

小小的字符串有着诸多巨人的影子

作者回复

非常感谢

明翼

回答一下上面一个人的问题，问题是\*\*String s3 = new String("12") + new String("34");  
s3.intern();  
String s4 = "1234";  
System.out.println(s3 == s4);//true

求解,为什么在第二段比较中会返回true,从字节码看s3应该就是生成了一个stringbuilder对象完成连接操作后执行了toString, s3不是应该仍然是堆内的对象地址吗?为什么会和常量池中的地址相等?"

我之前也是不明白s3为什么等于s4，查了下资料，说是在jdk1.7之后，如果字符串在堆中有实例，那intern方法就会把这个字符串的引用放在字符串常量池里，所以，String s3 = new String("12") + new String("34");这里在字符串常量池里放了一个字符串"12"，一个字符串"34"，在堆里存放他们的运算结果"1234"，然后把"1234"的引用返回给s3，s3.intern()这段代码运行时，jvm在堆里先到了字符串"1234"，所以就会把他的引用放到字符串常量池里，这个引用和s3相等，String s4 = "1234";这个代码时，会把字符串常量池里"1234"的引用返回给s4，所以s3是等于s4的。

个人理解，如有不对，请指正，谢谢💎💎

echo

String是Immutable，在security、Cache、Thread Safe等方面都有很好的体现。  
Security: 传参的时候我们很多地方使用String参数，可以保证参数不会被改变，比如数据库连接参数url等，从而保证数据库连接安全。  
Cache: 因为创建String前先去Constant Pool里面查看是否已经存在此字符串，如果已经存在，就把该字符串的地址引用赋给字符变量；如果没有，则在Constant Pool创建字符串，并将字符串引用赋给字符变量。所以存在多个引用指向同一个字符串对象，利用缓存有助于提高内存开销。  
Thread Safe: 因为String是immutable，所以它是Automatically thread safe。  
问题：我一直不能很好的理解最后一个体现，到底String是如何体现在thread safe，老师能不能用简短的线程代码给讲讲？非常感激。

Chris

new string ("ghhh").intern ()；会堆到常量池是这个作用吗

轩尼诗。

String s = new String("abc") 创建了几个对象？ 答案1：在字符串常量池中查找有没有“abc”，有则作为参数，就是创建了一个对象；没有则在常量池中创建，然后返回引用，那就是创建了两个对象。 答案2：直接在堆中创建一个新的对象。不检查字符串常量池，也不会把对象放入池中。网上正确答案貌似是两个，求指教到底是哪个！ 第一种可以写成String a = "abc";String s = new String(a) 那么第一种解释说得通，String a = "abc"会在常量池创建"abc"。但是String类的intern()是在字符串常量池中查找该字符串，有就返回，没有就创建，如果第一种解释说得通，那这个方法就废了。

王万云

看大神的文章真的提高太多了，而且还要看评论，评论区也都是高手云集

王建坤

讲师你好，有个疑问：“默认缓存大小也在不断地扩大中，从最初的1009，到7u40以后被修改为60013。”这里的1009 60013是指字符串的个数？还是内存占用？如果是内存占用的话，那单位是什么？

LenX

老师，这章学习到了 Java 8 以后，字符串常量池被移到了堆中，那么，如果通过 String.intern() 产生了大量的字符串常量，JVM 会对它们进行垃圾回收吗？

作者回复

会，具体时机我也没注意

jamie

编译器为什么不把  
String myStr = "aa" + "bb" + "cc" + "dd";  
默认优化成  
String myStr = "aabbccdd";  
这样不是更聪明嘛

这个我反编译试过，已经优化成aabbccdd了

Olm.chenteng

String s1 = new String("do");  
s1.intern();  
String s2 = "do";  
  
System.out.println(s1 == s2);//false  
  
String s3 = new String("12") + new String("34");  
s3.intern();  
String s4 = "1234";  
System.out.println(s3 == s4);//true

求解,为什么在第二段比较中会返回true,从字节码看s3应该就是生成了一个stringbuilder对象完成连接操作后执行了toString, s3不是应该仍然是堆内的对象地址吗?为什么会和常量池中的地址相等?

lo\_

代码荣耀说new的时候不检查字符串常量池.那请问老师  
String str1 = "abc";  
String str2 = new String("abc");  
第二行代码到底创建了几个对象呢.网上说1个.因为abc已经在常量池.但是如果new不检查常量池那是不是应该是两个对象呢？

So Leung	
那想问下，当new String("a")时，不会在常量池创建对象？	2018-05-24
© ®	
String s2=new String("AB")，，如果，常量池中沒有AB.那么会不会去常量池创建，望解答	2018-05-23
作者回复	
new只是创建新的；另外，有没有想过怎么通过一段程序证明？这样更有助于理解	2018-05-24
Miaozhe	
杨老师，问个问题： String str1 = "abc"; String str2 = new String("abc"); 理论上 str1是放入字符串常量池，str2是新增一个对象，新开辟一块地址。 但是通过代码调试，他们的HashCode一样，就说明他们是引用同一个地址。 请帮忙分析一下。	2018-05-18
作者回复	
据我所知，字符串hashCode是取决于内容，而不是地址； 对象是否同一个用==判断； 一定要区分，==，equals，hashCode，搜下或者看看书吧	2018-05-19
debugable	
java9讲字符串内部使用字节数组保存，取一个中文字符串字符个数是不是就不能用length了？	2018-05-15
作者回复	
不用担心，我提到了API行为没变化，包括charAt之类全部都是如此	2018-05-15
曹铮	
思考题里的平台默认编码，平台指的是JVM所在的操作系统，还是指语言平台本身呢？	2018-05-15
作者回复	
环境编码	2018-05-15
囍哈	
char 数组的实现，字符串的最大长度就是数组本身的长度限制，但是替换成 byte 数组，同样数组长度下，存储能力是退化了一倍的！ 怎么理解呢？举个例子呗	2018-05-15
作者回复	
已回复，一个char两个byte，注意下各个类型宽度	2018-05-15
jutsu	
地铁上看起来啦，从来没有string类型的线程问题，受教了	2018-05-15
作者回复	
谢谢	2018-05-15
呆	
String对操作符的覆写是怎样实现的呢，这也算是一类操作符重载吗？	2018-07-17
小潘	
string的intern方法在fastjson中就出现过bug	2018-06-26
风起	
在http传进来的string统一就是utf-8，等出去的话会肯能转一下格式。 然后对于循环外的就用加号，循环内就用stringbuilder。	2018-06-13
过冬	
老师，是退化还是进化？ 替换成 byte 数组，同样数组长度下，存储能力是退化了一倍的！ char变byte，数组长度不变，占用空间更小才对吧？	2018-06-07
作者回复	
退化，占用空间小，最大能表达的信息就少了一半	2018-06-07
aoe	
学习笔记 简单的拼接+就行了，只有相对复杂的，比如需要优化下buffer大小，才有必要考虑。 在 JDK 8 中，字符串拼接操作会自动被 javac 转换为 StringBuilder 操作。 而在 JDK 9 里面则是因为 Java 9 为了更加统一字符串操作优化，提供了 StringConcatFactory，作为一个统一的入口。 javac 自动生成的代码，虽然未必是最优化的，但普通场景也足够了。	2018-06-05
在 Java 9 中，我们引入了 Compact Strings 的设计，对字符串进行了大刀阔斧的改进。将数据存储方式从 char 数组，改变为一个 byte 数组加上一个标识编码的所谓 coder，并且将相关字符串操作类都进行了修改。 在极端情况下，字符串也出现了一些能力退化，比如最大字符串的大小。你可以思考下，原来 char 数组的实现，字符串的最大长度就是数组本身的长度限制，但是替换成 byte 数组，同样数组长度下，存储能力是退化了一倍的！ 还好这是存在于理论中的极限，还没有发现现实应用受此影响。	
在通用的性能测试和产品实验中，我们能非常明显地看到紧凑字符串带来的优势，即更小的内存占用、更快的操作速度。	
helloworld	2018-06-04

因为还没有开始学习JVM相关的知识，一涉及到其相关的内容后就感觉似懂非懂了！马上学习JVM相关的知识，希望老师尽早推出！	
今天也在为演唱会门票努力着	
上面那个问为什么不把"aa"+"bb"+"cc"优化成aabbcc的 据我所知 JDK8中已经实现了这种默认优化	2018-05-29
作者回复	
嗯，会优化，语义上和StringBuilder链一样	2018-05-29
小墨迹	
Jdk1.8 String str = "aa"; str = str + "bb"; 与 String str = "aa"+"bb" 反编译结果不一样，求解，谢谢哒， 什么格式的书写会智能优化	2018-05-29
作者回复	
这的看编译器的策略，具体要问编译器专家了； 也许相关，9里改为StringConcatFactory，一个原因就是便于一致的优化，不然优化逻辑是碎片、脆弱的	
鯨	2018-05-26
我在思考从字符串常量池检索是否已存在字符串时用的什么算法，貌似没有这块性能问题的？	
作者回复	
如果我没错就是哈希表	2018-05-28
So Leung	
经过验证new String时，不会再常量池中创建对象。	2018-05-24
作者回复	
嗯，重要的不是结论，而是如何得到结论	2018-05-28
© ®	
String s1=new String("StringTest"); System.out.println(s1.intern()==s1); //false (JDK 8)  String s1 = new StringBuilder().append("String").append("Test").toString(); System.out.println(s1.intern() == s1);://true (JDK 8)  String s1 = new StringBuilder("StringTest").toString(); system.out.println(s1.intern() == s1); //false (JDK 8) 。。老师，append为什么会造成这个差异	2018-05-24
岁月如歌	
stringbuffer的内容可以变化，但是它是线程安全的，stringbuler是非线程安全的，性能更高	2018-05-22
张高凯	
很不错，深入浅出，涉及到的技术原理都覆盖了，告诉了读者技术方向，想深入的可以继续翻阅更多资料	2018-05-22
作者回复	
谢谢	2018-05-23
df1996	
老师你好，为什么我用dk8试出来aa+bb+cc编译反编译是直接变成了aabbcc，但是我拼接的时候才用stringbuilder	2018-05-21
作者回复	
这个就是示例，具体不同版本、jdk、平台可能有区别， 9的字节码也仅仅是展示了indy模式的输出，实际有非常多模式，字节码也是不同的，只是接触一下，不过没必要都列出来	2018-05-22
刘为红	
你回答的“环境编码”不大明白，具体什么地方配置的	2018-05-17
刘为红	
getBytes()/String(byte[] bytes)使用过，遇到乱码情况，通过使用编码解决，所以这种机制我认为不好，跟平台的编码相关，但我没搞清楚平台的编码是指操作系统的编码？还是java虚拟机的编码	2018-05-17
小沙	
string为啥要设计成final？	2018-05-16
sunlight001	
string new对象的时候不是在 创建两个对象吗，一个放在栈中，一个放在常量池中吗？另外stringbuilder比stringbuffer的性能只有微弱的提高，在千万次的测试中，没有数量级的差别，所以我这程序拼接的时候一般都用stringbuffer，	2018-05-16
兔斯基	
印象中win平台默认是gbk编码，mac、Linux是utf8，前一段时间程序乱码问题就是因为开发用win,部署用linux，我觉得还是统一好，现在我们这边全部统一utf8,最起码这种乱码问题避免了	2018-05-15

你好，能否讲解一下StringConcatFactory的内在原理，最好能在源码这一层，谢谢。	2018-05-15
雪域飞鸿	
String的不可变是指引用不可变吗？既然里面是一个char类型数组，我能改变数组中的某一个元素吗？ 编码中字符集问题必须注意，否则会引起各种坑.....	2018-05-15
xqqsilver	
String q="we";q= "z"+q;是两个不同的对象是吗？	2018-05-15
作者回复	
思考下，怎么判断两个对象是不是相同？	2018-05-16
胖	
javac 自动生成的代码，虽然未必是最优化的，但普通场景也足够了，你可以酌情选择。 所以平常编码，直接用+进行字符拼接，并不需要显式调用StringBuilder？	2018-05-15
作者回复	
简单的拼接+就行了，只有相对复杂的，比如需要优化下buffer大小，才有必要考虑	2018-05-16
王磊	
假如我执行如下操作，并且常量池里确实已缓存过"abc"，那么执行intern()后s会指向常量池的"abc"，堆中的"abc"会被回收，是这样吗？ String s = new String("abc"); s.intern()  这样的话就会多产生一个垃圾。不想产生垃圾的话，就创建时直接指向常量池中的"abc" String s = "abc";	2018-05-15
请老师确实理解是否正确。	
作者回复	
我理解是	2018-05-15
一笑奈何	
char 数组的实现，字符串的最大长度就是数组本身的长度限制，但是替换成 byte 数组，同样数组长度下，存储能力是退化了一倍的！同问。不理解这句话。	2018-05-15
作者回复	
Java一个char是两个byte呀	2018-05-15
雪未央	
char占两个byte,同样长度的char数组和byte数组容量差两倍，不知道这样理解对不对？	2018-05-15
作者回复	
正解	2018-05-15
冬末末末	
用过String最大的坑就是substring方法，在1.6版本导致的内存泄露，老师都没有讲	2018-05-15
作者回复	
谢谢指出	2018-05-15
Leo	
老师，jdk6如果不使用intern方法，字符串常量不是存在永久代的常量池中吗？我的理解是intern方法是减少了永久代中字符串常量的数量，至于效率问题，是因为使用intern方法本身，多做了一步操作，所以会增加性能开销，主要考虑时间开销。	2018-05-15
作者回复	
文字常量那是默认行为，调用intern的不一定都是如此	2018-05-15
DavidWhom佳传	
老师，就这样面试题多讲讲挺好的	2018-05-15
作者回复	
谢谢	2018-05-15
wang_bo	
讲的很好，intern方法之前都没怎么接触过	2018-05-15
作者回复	
谢谢	2018-05-15