

03 | 字符串性能优化不容小觑，百M内存轻松存储几十G数据

2019-05-25 刘超

Java性能调优实战

[进入课程 >](#)



讲述：李良

时长 15:07 大小 13.84M



你好，我是刘超。


从第二个模块开始，我将带你学习 Java 编程的性能优化。今天我们就从最基础的 String 字符串优化讲起。

String 对象是我们使用最频繁的一个对象类型，但它的性能问题却是最容易被忽略的。String 对象作为 Java 语言中重要的数据类型，是内存中占据空间最大的一个对象。**高效地使用字符串，可以提升系统的整体性能。**

接下来我们就从 String 对象的实现、特性以及实际使用中的优化这三个方面入手，深入了解。

在开始之前，我想先问你一个小问题，也是我在招聘时，经常会问到面试者的一道题。虽是老生常谈了，但错误率依然很高，当然也有一些面试者答对了，但能解释清楚答案背后原理的人少之又少。问题如下：

通过三种不同的方式创建三个对象，再依次两两匹配，每组被匹配的两个对象是否相等？代码如下：

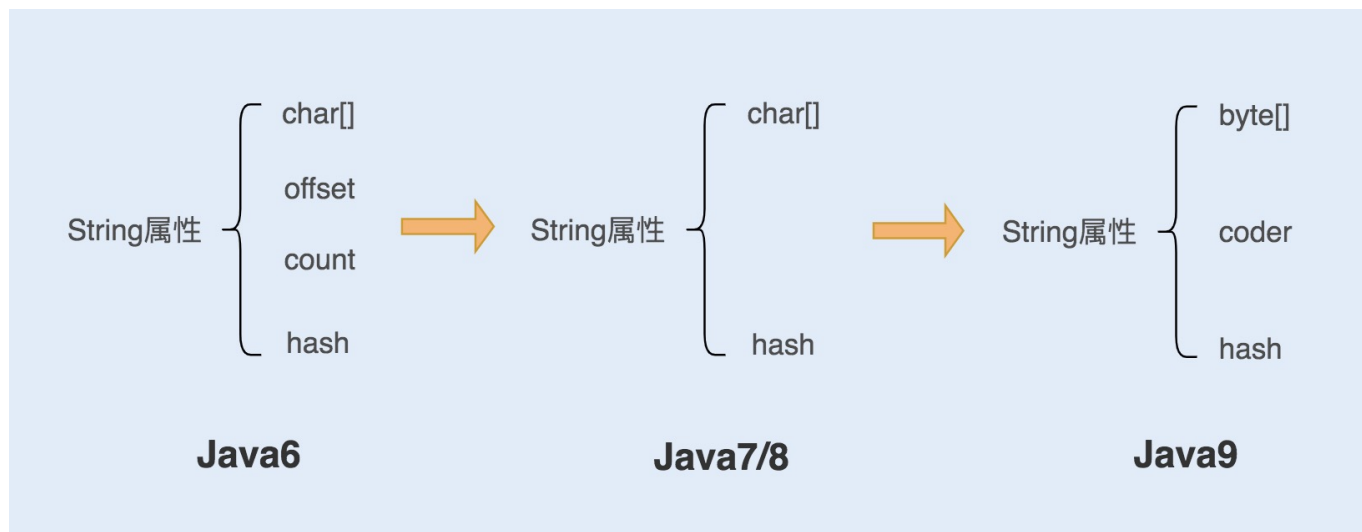
 复制代码

```
1 String str1= "abc";
2 String str2= new String("abc");
3 String str3= str2.intern();
4 assertEquals(str1==str2);
5 assertEquals(str2==str3);
6 assertEquals(str1==str3)
```

你可以先想想答案，以及这样回答的原因。希望通过今天的学习，你能拿到满分。

String 对象是如何实现的？

在 Java 语言中，Sun 公司的工程师们对 String 对象做了大量的优化，来节约内存空间，提升 String 对象在系统中的性能。一起来看看优化过程，如下图所示：



1. 在 Java6 以及之前的版本中，String 对象是对 char 数组进行了封装实现的对象，主要有四个成员变量：char 数组、偏移量 offset、字符数量 count、哈希值 hash。

String 对象是通过 offset 和 count 两个属性来定位 char[] 数组，获取字符串。这么做可以高效、快速地共享数组对象，同时节省内存空间，但这种方式很有可能会导致内存泄漏。

2. 从 Java7 版本开始到 Java8 版本，Java 对 String 类做了一些改变。String 类中不再有 offset 和 count 两个变量了。这样的好处是 String 对象占用的内存稍微少了些，同时，String.substring 方法也不再共享 char[]，从而解决了使用该方法可能导致的内存泄漏问题。

3. 从 Java9 版本开始，工程师将 char[] 字段改为了 byte[] 字段，又维护了一个新的属性 coder，它是一个编码格式的标识。

工程师为什么这样修改呢？

我们知道一个 char 字符占 16 位，2 个字节。这个情况下，存储单字节编码内的字符（占一个字节字符）就显得非常浪费。JDK1.9 的 String 类为了节约内存空间，于是使用了占 8 位，1 个字节的 byte 数组来存放字符串。

而新属性 coder 的作用是，在计算字符串长度或者使用 indexOf () 函数时，我们需要根据这个字段，判断如何计算字符串长度。coder 属性默认有 0 和 1 两个值，0 代表 Latin-1（单字节编码），1 代表 UTF-16。如果 String 判断字符串只包含了 Latin-1，则 coder 属性值为 0，反之则为 1。

String 对象的不可变性

了解了 String 对象的实现后，你有没有发现在实现代码中 String 类被 final 关键字修饰了，而且变量 char 数组也被 final 修饰了。

我们知道类被 final 修饰代表该类不可继承，而 char[] 被 final+private 修饰，代表了 String 对象不可被更改。Java 实现的这个特性叫作 String 对象的不可变性，即 String 对象一旦创建成功，就不能再对它进行改变。

Java 这样做的好处在哪里呢？

第一，保证 String 对象的安全性。假设 String 对象是可变的，那么 String 对象将可能被恶意修改。

第二，保证 hash 属性值不会频繁变更，确保了唯一性，使得类似 HashMap 容器才能实现相应的 key-value 缓存功能。

第三，可以实现字符串常量池。在 Java 中，通常有两种创建字符串对象的方式，一种是通过字符串常量的方式创建，如 `String str = "abc"`；另一种是字符串变量通过 new 形式的创建，如 `String str = new String("abc")`。

当代码中使用第一种方式创建字符串对象时，JVM 首先会检查该对象是否在字符串常量池中，如果在，就返回该对象引用，否则新的字符串将在常量池中被创建。这种方式可以减少同一个值的字符串对象的重复创建，节约内存。

`String str = new String("abc")` 这种方式，首先在编译类文件时，"abc" 常量字符串将会放入到常量结构中，在类加载时，"abc" 将会在常量池中创建；其次，在调用 new 时，JVM 命令将会调用 String 的构造函数，同时引用常量池中的 "abc" 字符串，在堆内存中创建一个 String 对象；最后，str 将引用 String 对象。

这里附上一个你可能会想到的经典反例。

平常编程时，对一个 String 对象 str 赋值 "hello"，然后又让 str 值为 "world"，这个时候 str 的值变成了 "world"。那么 str 值确实改变了，为什么我还说 String 对象不可变呢？

首先，我来解释下什么是对象和对象引用。Java 初学者往往对此存在误区，特别是一些从 PHP 转 Java 的同学。在 Java 中要比较两个对象是否相等，往往是用 `==`，而要判断两个对象的值是否相等，则需要用 `equals` 方法来判断。

这是因为 str 只是 String 对象的引用，并不是对象本身。对象在内存中是一块内存地址，str 则是一个指向该内存地址的引用。所以在刚刚我们说的这个例子中，第一次赋值的时候，创建了一个 "hello" 对象，str 引用指向 "hello" 地址；第二次赋值的时候，又重新创建了一个对象 "world"，str 引用指向了 "world"，但 "hello" 对象依然存在于内存中。


也就是说 str 并不是对象，而只是一个对象引用。真正的对象依然还在内存中，没有被改变。

String 对象的优化

了解了 String 对象的实现原理和特性，接下来我们就结合实际场景，看看如何优化 String 对象的使用，优化的过程中又有哪些需要注意的地方。

1. 如何构建超大字符串？


编程过程中，字符串的拼接很常见。前面我讲过 String 对象是不可变的，如果我们使用 String 对象相加，拼接我们想要的字符串，是不是就会产生多个对象呢？例如以下代码：

 复制代码

```
1 String str= "ab" + "cd" + "ef";
```


分析代码可知：首先会生成 ab 对象，再生成 abcd 对象，最后生成 abcdef 对象，从理论上来说，这段代码是低效的。

但实际运行中，我们发现只有一个对象生成，这是为什么呢？难道我们的理论判断错了？我们再来看编译后的代码，你会发现编译器自动优化了这行代码，如下：

 复制代码

```
1 String str= "abcdef";
```

上面我介绍的是字符串常量的累计，我们再来看看字符串变量的累计又是怎样的呢？

 复制代码

```
1 String str = "abcdef";  
2  
3 for(int i=0; i<1000; i++) {  
4     str = str + i;  
5 }
```

上面的代码编译后，你可以看到编译器同样对这段代码进行了优化。不难发现，Java 在进行字符串的拼接时，偏向使用 StringBuilder，这样可以提高程序的效率。

```
1
2 String str = "abcdef";
3
4 for(int i=0; i<1000; i++) {
5     str = (new StringBuilder(String.valueOf(str))).append(i).toString();
6 }
```

综上所述：即使使用 + 号作为字符串的拼接，也一样可以被编译器优化成 StringBuilder 的方式。但再细致些，你会发现在编译器优化的代码中，每次循环都会生成一个新的 StringBuilder 实例，同样也会降低系统的性能。

所以平时做字符串拼接的时候，我建议你还是要显示地使用 String Builder 来提升系统性能。

如果在多线程编程中，String 对象的拼接涉及到线程安全，你可以使用 StringBuffer。但是要注意，由于 StringBuffer 是线程安全的，涉及到锁竞争，所以从性能上来说，要比 StringBuilder 差一些。


2. 如何使用 String.intern 节省内存？

讲完了构建字符串，我们再来讨论下 String 对象的存储问题。先看一个案例。

Twitter 每次发布消息状态的时候，都会产生一个地址信息，以当时 Twitter 用户的规模预估，服务器需要 32G 的内存来存储地址信息。

```
1 public class Location {
2     private String city;
3     private String region;
4     private String countryCode;
5     private double longitude;
6     private double latitude;
7 }
```

考虑到其中有很多用户在地址信息上是有重合的，比如，国家、省份、城市等，这时就可以将这部分信息单独列出一个类，以减少重复，代码如下：


 复制代码

```
1
2 public class SharedLocation {
3
4     private String city;
5     private String region;
6     private String countryCode;
7 }
8
9 public class Location {
10
11     private SharedLocation sharedLocation;
12     double longitude;
13     double latitude;
14 }
```

通过优化，数据存储大小减到了 20G 左右。但对于内存存储这个数据来说，依然很大，怎么办呢？


这个案例来自一位 Twitter 工程师在 QCon 全球软件开发大会上的演讲，他们想到的解决方法，就是使用 `String.intern` 来节省内存空间，从而优化 `String` 对象的存储。

具体做法就是，在每次赋值的时候使用 `String` 的 `intern` 方法，如果常量池中有相同值，就会重复使用该对象，返回对象引用，这样一开始的对象就可以被回收掉。这种方式可以使重复性非常高的地址信息存储大小从 20G 降到几百兆。

 复制代码


```
1 SharedLocation sharedLocation = new SharedLocation();
2
3 sharedLocation.setCity(messageInfo.getCity().intern());          sharedLocation.setCount
4 sharedLocation.setRegion(messageInfo.getCountryCode().intern());
5
6 Location location = new Location();
7 location.set(sharedLocation);
8 location.set(messageInfo.getLongitude());
9 location.set(messageInfo.getLatitude());
```


为了更好地理解，我们再来通过一个简单的例子，回顾下其中的原理：

 复制代码

```
1 String a =new String("abc").intern();
2 String b = new String("abc").intern();
3
4 if(a==b) {
5     System.out.print("a==b");
6 }
```

输出结果：

 复制代码

```
1 a==b
```

在字符串常量中，默认会将对象放入常量池；在字符串变量中，对象是会创建在堆内存中，同时也会在常量池中创建一个字符串对象，复制到堆内存对象中，并返回堆内存对象引用。

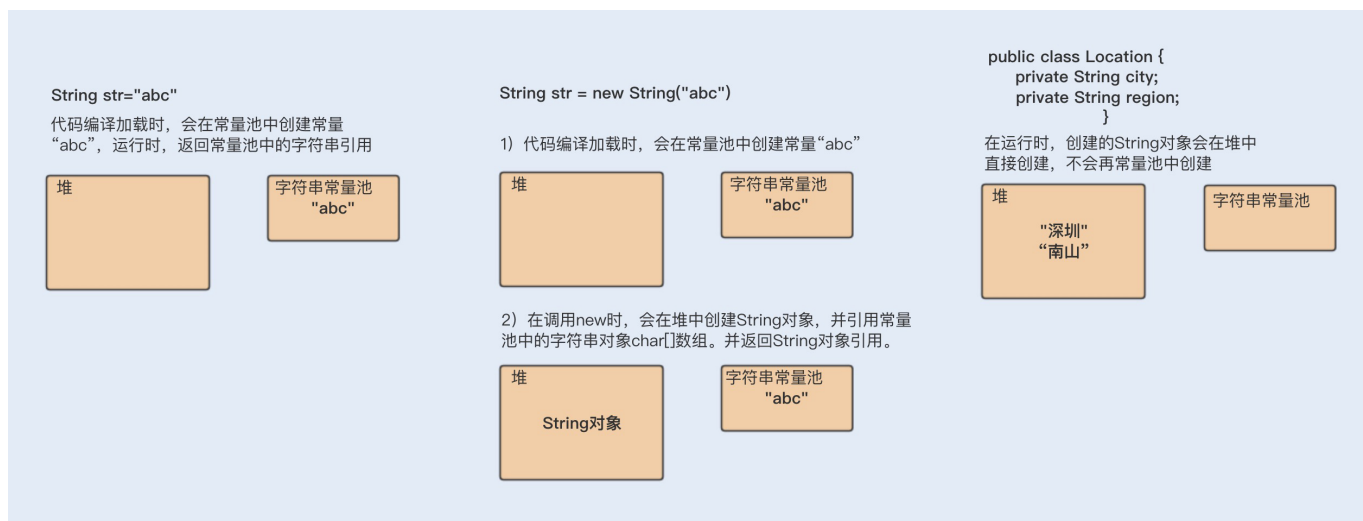
如果调用 intern 方法，会去查看字符串常量池中是否有等于该对象的字符串，如果没有，就在常量池中新增该对象，并返回该对象引用；如果有，就返回常量池中的字符串引用。堆内存中原有的对象由于没有引用指向它，将会通过垃圾回收器回收。

了解了原理，我们再一起看看上边的例子。

在一开始创建 a 变量时，会在堆内存中创建一个对象，同时会在加载类时，在常量池中创建一个字符串对象，在调用 intern 方法之后，会去常量池中查找是否有等于该字符串的对象，有就返回引用。

在创建 b 字符串变量时，也会在堆中创建一个对象，此时常量池中有该字符串对象，就不再创建。调用 intern 方法则会去常量池中判断是否有等于该字符串的对象，发现有等于"abc"字符串的对象，就直接返回引用。而在堆内存中的对象，由于没有引用指向它，将会被垃圾回收。所以 a 和 b 引用的是同一个对象。

下面我用一张图来总结下 String 字符串的创建分配内存地址情况：



使用 intern 方法需要注意的一点是，一定要结合实际场景。因为常量池的实现是类似于一个 HashTable 的实现方式，HashTable 存储的数据越大，遍历的时间复杂度就会增加。如果数据过大，会增加整个字符串常量池的负担。

3. 如何使用字符串的分割方法？

最后我想跟你聊聊字符串的分割，这种方法在编码中也很最常见。Split() 方法使用了正则表达式实现了其强大的分割功能，而正则表达式的性能是非常不稳定的，使用不恰当会引起回溯问题，很可能导致 CPU 居高不下。

所以我们应该慎重使用 Split() 方法，我们可以用 String.indexOf() 方法代替 Split() 方法完成字符串的分割。如果实在无法满足需求，你就在使用 Split() 方法时，对回溯问题加以重视就可以了。

总结

这一讲中，我们认识到做好 String 字符串性能优化，可以提高系统的整体性能。在这个理论基础上，Java 版本在迭代中通过不断地更改成员变量，节约内存空间，对 String 对象进行优化。

我们还特别提到了 String 对象的不可变性，正是这个特性实现了字符串常量池，通过减少同一个值的字符串对象的重复创建，进一步节约内存。

但也是因为这个特性，我们在做长字符串拼接时，需要显示使用 StringBuilder，以提高字符串的拼接性能。最后，在优化方面，我们还可以使用 intern 方法，让变量字符串对象重复使用常量池中相同值的对象，进而节约内存。

最后再分享一个个人观点。那就是千里之堤，溃于蚁穴。日常编程中，我们往往可能就是对一个小小的字符串了解不够深入，使用不够恰当，从而引发线上事故。

比如，在我之前的工作经历中，就曾因为使用正则表达式对字符串进行匹配，导致并发瓶颈，这里也可以将其归纳为字符串使用的性能问题。具体实战分析，我将在 04 讲中为你详解。

思考题

通过今天的学习，你知道文章开头那道面试题的答案了吗？背后的原理是什么？

互动时刻

今天除了思考题，我还想和你做一个简短的交流。

上两讲中，我收到了很多留言，在此非常感谢你的支持。由于前两讲是概述内容，主要是帮你建立对性能调优的整体认识，所以相对来说重理论、偏基础。但我发现，很多同学都有这样迫切的愿望，那就是赶紧学会使用排查工具，监测分析性能，解决当下的一些问题。

我这里特别想分享一点，其实性能调优不仅仅是学会使用排查监测工具，更重要的是掌握背后的调优原理，这样你不仅能够独立解决同一类的性能问题，还能写出高性能代码，所以我希望给你的学习路径是：夯实基础 - 结合实战 - 实现进阶。

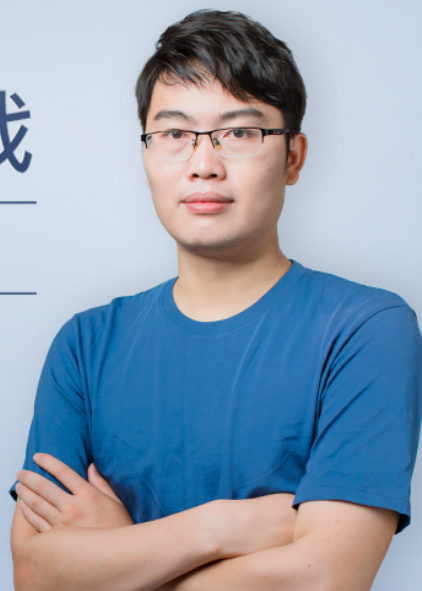
最后，欢迎你积极发言，讨论思考题或是你遇到的性能问题都可以，我会知无不尽。也欢迎你点击“请朋友读”，把今天的内容分享给身边的朋友，邀请他一起讨论。

Java 性能调优实战

覆盖 80% 以上 Java 应用调优场景

刘超

金山软件西山居技术经理



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 02 | 如何制定性能调优策略？

下一篇 04 | 慎重使用正则表达式

精选留言 (51)

写留言



KL3

2019-05-25

20

老师，能解释下，

“String.substring 方法也不再共享 char[]，从而解决了使用该方法可能导致的内存泄漏问题。”

共享char数组可能导致内存泄露问题？

展开

作者回复: 你好 KL3，在Java6中substring方法会调用new string构造函数，此时会复用原来的char数组，而如果我们仅仅是用substring获取一小段字符，而原本string字符串非常大的情况下，substring的对象如果一直被引用，由于substring的里面的char数组仍然指向原字符串，此时string字符串也无法回收，从而导致内存泄露。

试想下，如果有大量这种通过substring获取超大字符串中一小段字符串的操作，会因为内存泄露而导致内存溢出。



失火的夏天

2019-05-25

👍 12

开头题目答案是false false true

str1是建立在常量池中的“abc”，str2是new出来，在堆内存里的，所以str1!=str2，str3是通过str2.intern()出来的，str1在常量池中已经建立了“abc”，这个时候str3是从常量池里取出来的，和str1指向的是同一个对象，自然也就有了str1==str3，str3!=str2了

展开 ▾

作者回复: 这里我纠正下，str3是intern返回的引用，intern而不是创建出来的。

你的答案是正确的!



扫地僧

2019-05-25

👍 8

答案是false,false,true。背后的原理是：

- 1、String str1 = "abc";通过字面量的方式创建，abc存储于字符串常量池中；
- 2、String str2 = new String("abc");通过new对象的方式创建字符串对象，引用地址存放在堆内存中，abc则存放在字符串常量池中；所以str1 == str2?显然是false
- 3、String str3 = str2.intern();由于str2调用了intern()方法，会返回常量池中的数据，地...

展开 ▾

作者回复: 答案非常正确，理解了这个题目基本理解了string的特性了。



快乐的五五...

2019-05-25

👍 8

自学一年居然不知道有String.intern这个方法😞😞

不过从Java8开始（大概）String.split() 传入长度为1字符串的时候并不会使用正则，这种情况还是可以用

展开 ∨

作者回复: 非常感谢Geek的补充, 我在这里也再补充一个小点, split有两种情况不会使用正则表达式:

第一种为传入的参数长度为1, 且不包含 “.\$|()[]{^?*\+\\” regex元字符的情况下, 不会使用正则表达式;

第二种为传入的参数长度为2, 第一个字符是反斜杠, 并且第二个字符不是ASCII数字或ASCII字母的情况下, 不会使用正则表达式。

◀ ▶



风翱

2019-05-25

👍 6

使用 intern 方法需要注意的一点是, 一定要结合实际场景。因为常量池的实现是类似于一个 HashTable 的实现方式, HashTable 存储的数据越大, 遍历的时间复杂度就会增加。如果数据过大, 会增加整个字符串常量池的负担。
像国家地区是有边界的。像其他情况, 怎么把握这个度呢?

作者回复: 如果对空间要求高于时间要求, 且存在大量重复字符串时, 可以考虑使用常量池存储。

如果对查询速度要求很高, 且存储字符串数量很大, 重复率很低的情况下, 不建议存储在常量池中。

具体可以通过模拟测试自己的场景, 对比两种存储方式的性能, 通过数据来给自己答案。

◀ ▶



Eric

2019-05-25

👍 4

对于您文中 “在一开始创建 a 变量时, 会在堆内存中创建一个对象, 同时在常量池中创建一个字符串对象” 这句话 我认为前部分没有问题 分歧点在后面那部分 我觉得abc常量早在运行时常量池就存在了 可以理解使用这个类之前 就已经构造好了运行时常量池 而运行时常量池中就包括 “abc” 常量 至于使用new String(“abc”) 我觉得它应该只会在堆中创建String对象 并将运行时常量池中已经存在的 “abc” 常量的引用作为构造函数的参数...

展开 ∨

作者回复: 你理解的分歧点是对的, 这个构造是在加载类时, 就已经在常量池中构造好常量。

◀ ▶



Zend

2019-05-26

👍 3

“在字符串变量中，对象是会创建在堆内存中，同时也会在常量池中创建一个字符串对象，复制到堆内存对象中，并返回堆内存对象引用。”

比如：

是从常量池中复制到堆内存，这时常量池中字符串与堆内存字符串是完全独立的，内部...
展开 ▾

作者回复: 你好 Zend，具体的复制过程是先将常量池中的字符串压入栈中，在使用string的构造方法时，会拿到栈中的字符串作为构造方法的参数。这里我纠正一点，今天我查看了下这个构造函数，String的构造函数是一个char数组赋值过程，不是new char[]重新创建，所以是引用了常量池中的字符串对象，存在引用关系。



Eric

2019-05-25

👍 3

我在《Java虚拟机规范》里面看到一句话 这句话是当类或接口创建时，它的二进制表示中的常量池表被用来构造运行时常量池 我理解的意思是 类或接口 创建时就根据.class文件的常量池表生成了运行时常量池 执行new String("abc")这行代码应该只会生成一个String对象 并且调用它的构造函数 参数是运行时常量池里面"abc"字符串常量的Reference类型的数据（可以理解为指针吧）怎么会在这行代码执行的时候才会在运行时常量池生...

展开 ▾

作者回复: 如果是需要按照创建顺序来讲，常量“abc”，则会在加载编译时构造常量池时在常量池中创建“abc”字符串对象，而new对象的构造函数是在运行时创建并复制常量池中的“abc”。还有一个运行时常量池，也就是说，在运行时创建的字符串对象，通过intern方法会在运行时常量池中创建字符串对象。



建国

2019-05-25

👍 3

在实际编码中我们应该使用什么方式创建字符串呢？

A.String str= "abcdef";

B.String str= new String("abcdef");

```
C.String str= new String("abcdef"). intern();
```

```
D.String str1=str.intern();
```

展开 ▾

作者回复: 实际编码中, 我们要结合实际场景来选择创建字符串的方式, 例如, 在创建局部变量以及常量时, 我们一般使用A的这种方式; 如果我们要区别一个字符串创建两个不同的对象来使用时, 会选择B; intern一般使用的比较少, 例如我们平时会创建很多一样的字符串的对象时, 且对象会保存在内存中, 我们可以考虑使用intern方法来减少过多重复对象占用内存空间。



晓杰

2019-05-26

👍 2

回答开篇的问题:

str1会在常量池中创建一个对象

str2首先会在堆内存中创建一个对象, 然后在加载类的时候在常量池创建一个字符串对象, 同时复制到堆内存对象中, 并返回堆内存对象的引用

str3会先去常量池中查看存在于该字符串相等的对象, 因为str1已经在常量池创建了一个...

展开 ▾



-W.LI-

2019-05-26

👍 2

老师你说的对!直接把char数组做入参创建的String对象里的value数组地址是不一样的。调用intern()方法后就一样了。是我搞错了, 然后回到第一个问题的后半部分, 我打印输出a==b是false。之前有看到char数组的地址是一样的。这说明new虽然在堆中新建了一个String对象, 但是里面的char数组是复用的。这样做的目的是为了节约char数组的内存开销, 然后String本身就是不可变对象, 复用char数组不会带来问题。问题一:这个char数...

展开 ▾

作者回复: char数组是存放在常量池中, 常量是会在编译时生成字面量, 在类加载时加载到常量池中。

这个存放位置还是重要的, 这就相当于权职划分, 每个位置都有自己的功能和职责。

常量池中的垃圾回收, 也是垃圾回收器完成, 只要没有根引用的对象, 包括类信息等等, 都会在回收期被回收掉。常量池中的常量一般是固定的, 不像对中的对象。



-W.LI-

2019-05-26

👍 2

老师好！第一个问题没有描述清楚。String

`a = " abc" , String b =new String("abc"),String c=new String(new char[] { 'a' , 'b' , 'c' })`。创建的String对象。我debug时发现这三个String对象的value指向的那个char数组地址值都是一样的。他们是复用了一个char数组么?还是工具显示问题?我用的idea。

展开 ∨

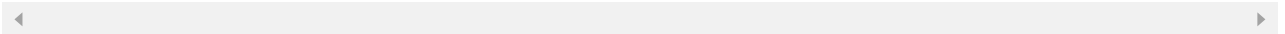
作者回复: 你好 W.LI, 刚我debug了下, a和b的value是同一个地址, 因为a在常量池中创建了"abc", 而new String("abc")时, 发现常量池存在"abc"字符串对象, 不会创建了。这时通过构造函数String(String original)将常量池中的"abc"复制给value, 这里的复制是引用, 不是创建新的char[]数组, 所以是同一个value地址。

而c中的构造函数, 是新开辟了一个char[]数组:

```
public String(char value[]) {  
    this.value = Arrays.copyOf(value, value.length);  
}
```

所以value的地址不一样。

可以再试试, 有问题留言。



QQ怪

2019-05-25

👍 2

老师讲的挺到位的, 挺容易理解, 之前忘记的现在又被老师点出来了, 支持老师 😊



° BugMak...

2019-05-31

👍 1

刘老师您好!"使用 intern 方法需要注意的一点是, 一定要结合实际场景。因为常量池的实现是类似于一个 HashTable 的实现方式, HashTable 存储的数据越大, 遍历的时间复杂度就会增加。如果数据过大, 会增加整个字符串常量池的负担",那这个Twitter 工程师在 QCon 全球软件开发大会上的演讲的那个 intern 方法是如何做到遍历这么多常量池的数据, 同时保证性能的呢?

展开 ∨

作者回复: 你好, 如果我们的数据对查询速度没有这么高要求, 可以考虑使用。



空

2019-05-30

👍 1

老师, java8还有字符串常量池吗, 都整合到堆里面去了吧

展开 ▾

作者回复: 有的, java8字符串常量池是分配到堆中, 并不代表字符串常量池就取消了。



Only now

2019-05-29

👍 1

看了本篇几乎全部留言, 感觉包括老师在内, 对于 "字符串常量池" 和 "常量池", 这两概念用的很混。

对于jdk7 以及之前的jvm版本不再去深究了, 它的字符串常量池存在于方法区, 但是jdk8以后, 它存在于Java堆中, 唯一, 且由java.lang.String类维护, 它和类文件常量池, 运行时常...

展开 ▾

作者回复: 非常感谢only now的总结, 这一讲中没有详细去区分常量池, 而是在强调字符串的使用, 后面我们在JVM中可以再一起研究下常量池。

JVM文献中提到方法区是存在垃圾回收。我们可以通过intern方法来验证这个gc问题, 通过大量请求请求某个接口, 传入参数创建字符串对象, 之后通过intern方法在常量池中生成字符串对象, 之后失去引用, 观察gc情况。



大海

2019-05-26

👍 1

```
String s = new String("abc").intern();
```

既然使用intern也会引用到常量池,那么和 使用intern 和 直接使用 `String s = "abc"`有差别吗

展开 ▾

作者回复: 最终实现达到的结果是一样的, 但过程不一样。我拿这个例子来说明, 在程序运行期间动态创建的字符串对象, 由于这类字符串是在内存中开辟的地址空间存放字符串, 可以使用intern

方法放在常量池中。



业余草

2019-05-25

1

final 标示的一般为常量，按照老师说的 `new String("abc").intern()` 在常量池中也存在 `abc`，`String str1 = "abc"`；通过字面量的方式创建，`abc` 存储于字符串常量池中；是不是说用不用 `final` 都无所谓了？？请帮忙详细解答一下，谢谢！！

作者回复: 正是因为 `final`，字符串才实现了不可变性，`String` 内部的 `value` 已经被 `final` 修饰，所以我们不用再在编码时用 `final` 修饰。



-W.LI-

2019-05-25

1

老师好！jdk8 环境下，观察了下。不管是直接赋值还是 `new` 创建新对象，`String` 对象的 `value` 对应的 `char` 数组地址都是同一个，这么做就是不论字符串多大，重复 `new` 只会多消耗 `string` 对象必须得的 16 字节内存是吗？jdk8 以前常量池在方法区中，属于永久代，GC 不会回收这一部分空间是吗？jdk8 对常量池做了改动，放在了堆中，在堆中会被 GC 回收。我想知道为啥要做这个调整。常量池也是在新生代创建，然后几次 `ygc` 以后进入老年代么？

展开

作者回复: 这位同学 你好，不好意思呀，我没有理解你的第一个问题，你在问 `substring` 在 `java8` 创建新对象的问题吗？

常量池放在堆中，是为了解决之前放在方法区时，由于常量池空间大小有限，存储对象过多导致内存溢出问题。也会存在垃圾回收，但与堆的垃圾回收不一样，这里不会进入老年代，而是直接回收。



Eric

2019-05-25

1

当类或接口创建时 会根据 `.class` 文件里面的常量池表构造运行时常量池 可以认为在调用 `new String("abc")` 很早很早之前 运行时常量池已经有了 `abc` 这个字符串常量吗？这里 `ldc` 只是把对应的字符串常量的 `reference` 类型的数据压栈 作为后面的 `invokespecial` 指令来调

用构造方法的第一个参数？ 理解为ldc指令时在运行时常量池创建abc字符串常量是否合适？ 还是我的理解存在错误？

展开 ∨

作者回复: 你好 Eric，从反编译文件可以看出， 字符串 “abc” 在编译时，已在常量池中创建，这个没问题。如果是变量，这个需要在运行时在常量池中创建。

ldc是JVM的入栈指令，在这里会将常量池中的 “abc” 入栈，invokespecial是JVM调用构造函数的指令，此时会调用String的构造函数， “abc” 作为参数。之后通过出栈指令返回引用给s1。

大概就是这个流程，有疑问欢迎一起探讨。

