# String

## 一、内存占用

### Java对象内存布局

Java对象的内存布局包括：对象头(Header)，实例数据(Instance Data)和补齐填充(Padding)。

对象头：

关于对象头的详细介绍可以参看我的博文：<http://blog.csdn.net/codershamo/article/details/52071996>， 这里只关注其内存占用大小。在64位机器上，默认不开启指针压缩（-XX:-UseCompressedOops）的情况下，对象头占用16bytes，开启指针压缩（-XX:+UseCompressedOops）则占用12bytes。

实例数据：

原生类型（primitive type）的内存占用如下：

|  |  |
| --- | --- |
| **Primitive Type** | **Memory Required(bytes)** |
| byte, boolean | 1 byte |
| short, char | 2 bytes |
| int, float | 4 bytes |
| long, double | 8 bytes |

对象引用（reference）类型在64位机器上，关闭指针压缩时占用8bytes， 开启时占用4bytes。

对齐填充：

Java对象占用空间是8字节对齐的，即所有Java对象占用bytes数必须是8的倍数。例如，一个包含两个属性的对象：int和byte，并不是占用17bytes(12+4+1)，而是占用24bytes（对17bytes进行8字节对齐）

### jdk6

<http://www.importnew.com/21836.html>

对于 String（JDK 6）的成员变量声明如下：

private final char value[];

private final int offset;

private final int count;

private int hash;

先思考一个内存占用的问题：字符串 “Hello World” 会占用多少字节内存？

长度分为以下三部分：

String对象：[12/16 （字符串的对象头） + 4 （偏移） + 4 （偏移长度） + 4 （哈希码） + 4/8 （指向字符数组的引用)，加上字节对齐所需的填充，共为32/40字节]

char[]对象：[12/16 （字符数组的对象头） + 4 （数组长度），加上字节对齐所需的填充，共为 16/24 字节]

字符：11 \* 2 （字符），加上字节对齐所需的填充，共为 24 字节

所以，总长度为32/40+16/24+24=72/88字节。

### jdk7、8

<https://segmentfault.com/a/1190000006933272>

在JDK1.7及以上版本中，String包含2个属性，一个用于存放字符串数据的char[], 一个int类型的hashcode, 部分源代码如下：

public final class String

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

/\*\* The value is used for character storage. \*/

private final char value[];

/\*\* Cache the hash code for the string \*/

private int hash; // Default to 0

}

先思考一个内存占用的问题：字符串 “Hello World” 会占用多少字节内存？

长度分为以下三部分：

String对象：[12/16 （字符串的对象头） + 4 （哈希码） + 4/8 （指向字符数组的引用)，加上字节对齐所需的填充，共为24/32字节]

char[]对象：[12/16 （字符数组的对象头） + 4 （数组长度），加上字节对齐所需的填充，共为 16/24 字节]

字符：11 \* 2 （字符），加上字节对齐所需的填充，共为 24 字节

所以，总长度为24/32+16/24+24=64/80字节。

## split和substring

案例分析，内存溢出

<https://www.ibm.com/developerworks/cn/java/j-lo-optmizestring/>

jdk6和jdk7的效果不同

jdk7的没有内存溢出的问题，没有offset和count

<https://segmentfault.com/a/1190000006933272#articleHeader7>

## 字符串常量池

小于Java6u30版本中，字符串常量池大小不可配置，大于等于Java6u30版本时变得可配置。你需要通过-XX:StringTableSize=N指定，其中N是字符串常量池map的大小。基于性能考虑，N最好是质数。但在Java6中这个参数帮助可能不大，因为你还是受限于PermGen的大小。

小于Java7u40版本中，字符串常量池的默认大小是1009，大于等于Java7u40版本时，字符串常量池的默认大小是60013。超出数量，发生碰撞可能性非常大,性能会急速下降，从Hash的性能很快退化成链表的性能。

在Java7中，Oracle的工程师对“字符串池化”的逻辑作了重大的改变——将字符串常量池移动到了堆中。这意味着你不会再被一块固定大小的内存区域所限制。

在JDK6之前(包含JDK6)，所有的interned strings都存储在PermGen（永久代）中。

除了被显式intern的strings，永久代的字符串常量池还被用来存储程序中使用过的所有String字面量（要注意的是“使用过的”，如果一个类/方法从来没有被加载/调用，定义其中的任何常量都不会被加载）。

java6中这样构建的字符串常量池最大的问题在于它所处的地址——PermGen（永久代）。PermGen内存通常比堆内存小很多，且不可回收。

基于这些限制，你在使用String.intern的时候必须特别小心—— java6最好不要使用Sting.intern缓存任何不受控制的用户输入。这就是为啥在Java6的时候，大部分字符串常量池都是通过手动维护的Maps来实现的原因。

在JDK7之后（包含JDK7），Oracle的工程师对“字符串池化”的逻辑作了重大的改变——将字符串常量池移动到了堆中。像大多数常规对象一样，所有的Strings现在会处于堆中。你受限的是内存空间更大的堆区域。意味着在一开始你可以把字符串常量池设置得更大（根据你应用需求而定）。估算一下你应用中不同strings的数量，将常量池的大小设置为接近strings数量2倍的一个质数（避免可能的碰撞）。这会让String.intern运行在一个常数时间内，并且每个缓存字符串所需内存会很小（在同任务量下，显式使用Java WeakHashMap会产生4-5倍多的内存开销）。

从技术上说，仅仅因为这个原因，你可以重新考虑在java7程序中使用String.intern()。但是还有其他原因。

如果你不确定字符串常量池的使用情况，尝试使用 -XX:+PrintStringTableStatics 虚拟机参数。它将会在你程序结束时打印出你的字符串常量池的使用情况。

字符串常量池中的值可以被GC回收,不管是在永久代还是在堆空间中

是的，JVM字符串常量池中的所有对象在没有被GC roots引用的情况下都可以被回收，这个结论适用于我们讨论的所有Java版本。

这意味着如果你缓存的string逃离了作用域并且失去了引用——它将被移出JVM字符串常量池，并且被gc回收。[1]

http://www.importnew.com/18454.html

## intern

字符串池化（通常也叫作“字符串常量化”），就是把一些标识符（可以理解为变量名）不同但是值相同的String对象用一个共享的String对象替代。你可以通过维护一个Map<String,String>(根据需求可能需要使用soft/weak引用)或者使用JDK提供的String.intern()方法来实现“字符串常量池”。

在Java6的时代，很多标准禁止使用String.intern()，因为如果不加控制地使用String.intern()，很可能导致OOM。Java 7对“字符串常量池”的实现作了非常大的改变。

### JDK6中的String.intern()

其他 String 使用的优化建议

<https://www.ibm.com/developerworks/cn/java/j-lo-optmizestring/>

<http://www.importnew.com/21836.html>

以下三篇文章对intern在jdk6和jdk7里的区别没有解释清楚

~~http://www.importnew.com/14142.html~~

[~~http://www.jianshu.com/p/0d1c003d2ff5~~](http://www.jianshu.com/p/0d1c003d2ff5)

~~http://www.jianshu.com/p/2f209af80f84~~

可以参考下面的文章，对intern解释清楚了

通过反编译深入理解Java String及intern

http://www.importnew.com/21024.html

s.intern()是无效的，因为String是不变对象， String s1 = s.intern()后，这个s1才是个引用。

intern节省内存，但性能会稍微下降的。

执行路径上多次的==，才能抵消常量池HasHMap查找的代价

http://www.importnew.com/22126.html

==当然比equals()快得多，但常量池其实是个HashMap，依然没有神奇的地方，依然要执行HashMap的get操作，所以，一次hashCode() 和至少一次的equals()已经预付了，如果hash冲突，那equals()次数更多。

## 字符串连接

当你连接2或3个String时，使用String.concat()。

如果你要连接多于3个String（不含3）,并且你能够精确预测出最终结果的长度，使用StringBuilder/StringBuffer，并设定初始化容量。

如果你要连接多于3个String（不含3）,并且你不能够精确预测出最终结果的长度，使用StringBundler。

如果你使用StringBundler,并且你能预测出要连接的String数量，使用指定初始化容量的构造函数。

<http://www.blogjava.net/javagrass/archive/2010/01/24/310650.html>

当转化成字符串的时候，应当避免使用”"串进行转化。使用合适的String.valueOf方法或者包装类的toString(value)方法。

尽量使用StringBuilder进行字符串拼接。检查下老旧码，把那些能替换掉的StringBuffer也替换成它。

使用Java 6 update 20引入的-XX:+OptimizeStringConcat选项来提高字符串拼接的性能。在最近的Java7的版本中已经默认打开了，不过在Java 6\_41还是关闭的。

http://www.importnew.com/14595.html

<http://www.importnew.com/20033.html>

## 问题

有人说String是不可变的对象，这是java的安全基石，可是java又推出了StirngBuilder，并且鼓励使用StringBuilder，那是否有必要推出String？

## 引用

[1]http://www.jianshu.com/p/50b085b4920e