# JAVA 一个对象消耗了多少内存

- weizongwei5 - 博客频道 - CSDN.NET

http://blog.csdn.net/weizongwei5/article/details/45000083

请先允许我装逼的说一句:当你写代码达到一定境界的时候，你写一句代码，大概你脑子里已经知道了这句代码执行的时候CPU耗时是0.0000几ms ，内存大概消耗了几个byte。

只学不思的人愚，只思不学的人顿。  学而不思则罔，死而不学则殆。

有本书叫 《深入理解Java虚拟机》  推荐大家读一读  !

**好啦装逼装完了！**

对象内存大小度量

在做内存优化时，需要知道每个对象占用的内存的大小，

一个实例化的对象在内存中需要存储的信息包括：

1. 对象的头部(对象的GC信息，hash值，类定义引用等)
2. 对象的成员变量: 包括基本数据类型和引用。 如成员变量是一个引用, 引用了其他对象，被引用的对象内存另外计算。

如下一个简单的类的定义:

**[java]** [view plain](http://blog.csdn.net/weizongwei5/article/details/45000083) [copy](http://blog.csdn.net/weizongwei5/article/details/45000083)

[print?](http://blog.csdn.net/weizongwei5/article/details/45000083)

1. **class** MyClass {
2. **int** a;
3. Object object;
4. }

实例化一个对象:

**[java]** [view plain](http://blog.csdn.net/weizongwei5/article/details/45000083) [copy](http://blog.csdn.net/weizongwei5/article/details/45000083)

[print?](http://blog.csdn.net/weizongwei5/article/details/45000083)

1. MyClass myClass = **new** MyClass();

对象大小分为:

1. 自身的大小(Shadow heap size)
2. 所引用的对象的大小(Retained heap size)。

myClass实例创建出来之后，在内存中所占的大小就是myClass自身大小(Shadow heap size)。包括类的头部大小以及一个int的大小和一个引用的大小。

myClass 中object 成员变量是一个对象引用，这个被引用的对象也占一定大小。myClass实例所维护的引用的对象所占的大小，称为myClass实例的Retained heap size。

本文讨论的是对象自身的大小，即Shadow heap size。Retained heap size 递归计算即可得。

度量工具

对象大小的计算可用[**Java**](http://lib.csdn.net/base/javase).lang.instrument.Instrumentation 或者 dump内存之后用memory analyzer分析。 这是一份示例代码[java-object-size](https://github.com/liaohuqiu/java-object-size)

基本数据类型大小

基本数据类型大小如下: [From WIKI](http://en.wikibooks.org/wiki/Java_Programming/Primitive_Types)

| **type** | **size(bits)** | **bytes** |
| --- | --- | --- |
| boolean | 8 | 1 |
| byte | 8 | 1 |
| char | 16 | 2 |
| short | 16 | 2 |
| int | 32 | 4 |
| long | 64 | 8 |
| float | 32 | 4 |
| double | 64 | 8 |

引用的大小

在32位的JVM上，一个对象引用占用4个字节；在64位上，占用8个字节。通过 java -d64 -version可确定是否是64位的JVM。

使用8个字节是为了能够管理大于4G的内存，如果你的程序不需要访问大于4G的内存，

可通过-XX:+UseCompressedOops选项，开启指针压缩。从Java 1.6.0\_23起，这个选项默认是开的。可通过jinfo -flag UseCompressedOops <pid>查看。

localhost:~ srain$ jinfo -flag UseCompressedOops 13133

-XX:+UseCompressedOops

对象头部的大小

对象头，结构如下[(来源)](http://mail.openjdk.java.net/pipermail/hotspot-runtime-dev/2008-May/000147.html):

+------------------+------------------+------------------ +---------------.

| mark word | klass pointer | array size (opt) | padding |

+------------------+------------------+-------------------+---------------'

每个对象都有一个mark work头部，以及一个引用，指向类的信息。在32位JVM上，mark word 4个字节，整个头部有8字节大小。

在未开启UseCompressedOops的64位JVM上，对象头有16字节大小。

在开启UseCompressedOops的64位机器上，引用成了4字节，一共12字节。 按照8位对齐，实际占用16字节。

对象的内存布局

1. 每个对象的内存占用按8字节对齐
2. 空对象和类实例成员变量

空对象，指的非inner-class，没有实例属性的类。Object 类或者直接继承Object 没有添加任何实例成员的类。

空对象的不包含任何成员变量，其大小即对象头大小:

* + 在32位JVM上，占用8字节；
  + 在开启UseCompressedOops的64位JVM上，12 + 4 = 16；
  + 在未开启UseCompressedOops的64位JVM上，16 + 4 = 20; 对齐后为24。

1. 对象实例成员重排序

实例成员变量紧随对象头。每个成员变量都尽量使本身的大小在内存中尽量对齐。

比如int按4位对齐，long按8位对齐。为了内存紧凑，实例成员在内存中的排列和声明的顺序可能不一致，实际会按以下顺序排序:

* + doubles and longs
  + ints and floats
  + shorts and chars
  + booleans and bytes
  + references

这样做可尽量节省空间。

如:

**[java]** [view plain](http://blog.csdn.net/weizongwei5/article/details/45000083) [copy](http://blog.csdn.net/weizongwei5/article/details/45000083)

[print?](http://blog.csdn.net/weizongwei5/article/details/45000083)

* + **class** MyClass {
  + **byte** a;
  + **int** c;
  + **boolean** d;
  + **long** e;
  + Object f;
  + }

未重排之前:

**[java]** [view plain](http://blog.csdn.net/weizongwei5/article/details/45000083) [copy](http://blog.csdn.net/weizongwei5/article/details/45000083)

[print?](http://blog.csdn.net/weizongwei5/article/details/45000083)

* 1. 32 bit                    64bit +UseCompressedOops
  3. [HEADER: 12 bytes]  8           [HEADER: 12 bytes] 12
  4. [a:       1 **byte** ]  9           [a:       1 **byte** ] 13
  5. [padding: 3 bytes] 12           [padding: 3 bytes] 16
  6. [c:       4 bytes] 16           [c:       4 bytes] 20
  7. [d:       1 **byte** ] 17           [d:       1 **byte** ] 21
  8. [padding: 7 bytes] 24           [padding: 3 bytes] 24
  9. [e:       8 bytes] 32           [e:       8 bytes] 32
  10. [f:       4 bytes] 36           [f:       4 bytes] 36
  11. [padding: 4 bytes] 40           [padding: 4 bytes] 40

重新排列之后：

**[java]** [view plain](http://blog.csdn.net/weizongwei5/article/details/45000083) [copy](http://blog.csdn.net/weizongwei5/article/details/45000083)

[print?](http://blog.csdn.net/weizongwei5/article/details/45000083)

* 1. 32 bit                      64bit +UseCompressedOops
  3. [HEADER:  8 bytes]  8           [HEADER: 12 bytes] 12
  4. [e:       8 bytes] 16           [e:       8 bytes] 20
  5. [c:       4 bytes] 20           [c:       4 bytes] 24
  6. [a:       1 **byte** ] 21           [a:       1 **byte** ] 25
  7. [d:       1 **byte** ] 22           [d:       1 **byte** ] 26
  8. [padding: 2 bytes] 24           [padding: 2 bytes] 28
  9. [f:       4 bytes] 28           [f:       4 bytes] 32
  10. [padding: 4 bytes] 32

1. 父类和子类的实例成员

父类和子类的成员变量分开存放，先是父类的实例成员。父类实例成员变量结束之后，按4位对齐，随后接着子类实例成员变量。

**[java]** [view plain](http://blog.csdn.net/weizongwei5/article/details/45000083) [copy](http://blog.csdn.net/weizongwei5/article/details/45000083)

[print?](http://blog.csdn.net/weizongwei5/article/details/45000083)

* 1. **class** A {
  2. **byte** a;
  3. }
  5. **class** B **extends** A {
  6. **byte** b;
  7. }

内存结构如下:

**[java]** [view plain](http://blog.csdn.net/weizongwei5/article/details/45000083) [copy](http://blog.csdn.net/weizongwei5/article/details/45000083)

[print?](http://blog.csdn.net/weizongwei5/article/details/45000083)

* 1. 32 bit                  64bit +UseCompressedOops
  3. [HEADER:  8 bytes]  8       [HEADER: 12 bytes] 12
  4. [a:       1 **byte** ]  9       [a:       1 **byte** ] 13
  5. [padding: 3 bytes] 12       [padding: 3 bytes] 16
  6. [b:       1 **byte** ] 13       [b:       1 **byte** ] 17
  7. [padding: 3 bytes] 16       [padding: 7 bytes] 24

如果子类首个成员变量是long或者double等8字节数据类型，而父类结束时没有8位对齐。会把子类的小于8字节的实例成员先排列，直到能8字节对齐。

**[java]** [view plain](http://blog.csdn.net/weizongwei5/article/details/45000083) [copy](http://blog.csdn.net/weizongwei5/article/details/45000083)

[print?](http://blog.csdn.net/weizongwei5/article/details/45000083)

* 1. **class** A {
  2. **byte** a;
  3. }
  5. **class** B **extends** A{
  6. **long** b;
  7. **short** c;
  8. **byte** d;
  9. }

内存结构如下:

**[java]** [view plain](http://blog.csdn.net/weizongwei5/article/details/45000083) [copy](http://blog.csdn.net/weizongwei5/article/details/45000083)

[print?](http://blog.csdn.net/weizongwei5/article/details/45000083)

* 1. 32 bit                  64bit +UseCompressedOops
  3. [HEADER:  8 bytes]  8       [HEADER:  8 bytes] 12
  4. [a:       1 **byte** ]  9       [a:       1 **byte** ] 13
  5. [padding: 3 bytes] 12       [padding: 3 bytes] 16
  6. [c:       2 bytes] 14       [b:       8 bytes] 24
  7. [d:       1 **byte** ] 15       [c:       4 **byte** ] 28
  8. [padding: 1 **byte** ] 16       [d:       1 **byte** ] 29
  9. [b:       8 bytes] 24       [padding: 3 bytes] 32

上面的示例中，在32位的JVM上，B的2个实例成员c, d被提前了。

1. 非静态的内部类，有一个隐藏的对外部类的引用。

数组的内存占用大小

数组也是对象，故有对象的头部，另外数组还有一个记录数组长度的int类型，随后是每一个数组的元素：基本数据类型或者引用。8字节对齐。

* 32 位的机器上

byte[0] 8字节的对象头部，4字节的int长度, 12字节，对齐后是16字节，实际 byte[0] ~ byte[4] 都是16字节。

* 64 位+UseCompressedOops

byte[0] 是16字节大小，byte[1] ~ byte[8] 24字节大小。

* 64 位-UseCompressedOops

byte[0], 16字节头部，4字节的int长度信息，20字节，对齐后 24 字节。byte[0] ~ byte[4] 都是24字节。

字符串大小

| **Field** | **Type** | **64 bit -UseCompressedOops** | **64 bit +UseCompressedOops** | **32 bit** |
| --- | --- | --- | --- | --- |
| HEADER |  | 16 | 12 | 8 |
| value | char[] | 8 | 4 | 4 |
| offset | int | 4 | 4 | 4 |
| count | int | 4 | 4 | 4 |
| hash | int | 4 | 4 | 4 |
| PADDING |  | 4 | 4 | 0 |
| TOTAL |  | 40 | 32 | 24 |

不计算value引用的Retained heap size, 字符串本身就需要 24 ~ 40 字节大小。

在网上搜到了一篇博客讲的非常好：http://yueyemaitian.iteye.com/blog/2033046，里面提供的这个类也非常实用：

**[java]** [view plain](http://blog.csdn.net/weizongwei5/article/details/45000083) [copy](http://blog.csdn.net/weizongwei5/article/details/45000083)

[print?](http://blog.csdn.net/weizongwei5/article/details/45000083)

1. **import** java.lang.instrument.Instrumentation;
2. **import** java.lang.reflect.Array;
3. **import** java.lang.reflect.Field;
4. **import** java.lang.reflect.Modifier;
5. **import** java.util.ArrayDeque;
6. **import** java.util.Deque;
7. **import** java.util.HashSet;
8. **import** java.util.Set;
10. /\*\*
11. \* 对象占用字节大小工具类
12. \*
13. \* @author tianmai.fh
14. \* @date 2014-03-18 11:29
15. \*/
16. **public** **class** SizeOfObject {
17. **static** Instrumentation inst;
19. **public** **static** **void** premain(String args, Instrumentation instP) {
20. inst = instP;
21. }
23. /\*\*
24. \* 直接计算当前对象占用空间大小，包括当前类及超类的基本类型实例字段大小、<br></br>
25. \* 引用类型实例字段引用大小、实例基本类型数组总占用空间、实例引用类型数组引用本身占用空间大小;<br></br>
26. \* 但是不包括超类继承下来的和当前类声明的实例引用字段的对象本身的大小、实例引用数组引用的对象本身的大小 <br></br>
27. \*
28. \* @param obj
29. \* @return
30. \*/
31. **public** **static** **long** sizeOf(Object obj) {
32. **return** inst.getObjectSize(obj);
33. }
35. /\*\*
36. \* 递归计算当前对象占用空间总大小，包括当前类和超类的实例字段大小以及实例字段引用对象大小
37. \*
38. \* @param objP
39. \* @return
40. \* @throws IllegalAccessException
41. \*/
42. **public** **static** **long** fullSizeOf(Object objP) **throws** IllegalAccessException {
43. Set<Object> visited = **new** HashSet<Object>();
44. Deque<Object> toBeQueue = **new** ArrayDeque<Object>();
45. toBeQueue.add(objP);
46. **long** size = 0L;
47. **while** (toBeQueue.size() > 0) {
48. Object obj = toBeQueue.poll();
49. //sizeOf的时候已经计基本类型和引用的长度，包括数组
50. size += skipObject(visited, obj) ? 0L : sizeOf(obj);
51. Class<?> tmpObjClass = obj.getClass();
52. **if** (tmpObjClass.isArray()) {
53. //[I , [F 基本类型名字长度是2
54. **if** (tmpObjClass.getName().length() > 2) {
55. **for** (**int** i = 0, len = Array.getLength(obj); i < len; i++) {
56. Object tmp = Array.get(obj, i);
57. **if** (tmp != **null**) {
58. //非基本类型需要深度遍历其对象
59. toBeQueue.add(Array.get(obj, i));
60. }
61. }
62. }
63. } **else** {
64. **while** (tmpObjClass != **null**) {
65. Field[] fields = tmpObjClass.getDeclaredFields();
66. **for** (Field field : fields) {
67. **if** (Modifier.isStatic(field.getModifiers())   //静态不计
68. || field.getType().isPrimitive()) {    //基本类型不重复计
69. **continue**;
70. }
72. field.setAccessible(**true**);
73. Object fieldValue = field.get(obj);
74. **if** (fieldValue == **null**) {
75. **continue**;
76. }
77. toBeQueue.add(fieldValue);
78. }
79. tmpObjClass = tmpObjClass.getSuperclass();
80. }
81. }
82. }
83. **return** size;
84. }
86. /\*\*
87. \* String.intern的对象不计；计算过的不计，也避免死循环
88. \*
89. \* @param visited
90. \* @param obj
91. \* @return
92. \*/
93. **static** **boolean** skipObject(Set<Object> visited, Object obj) {
94. **if** (obj **instanceof** String && obj == ((String) obj).intern()) {
95. **return** **true**;
96. }
97. **return** visited.contains(obj);
98. }
99. }

参考资料

<http://www.codeinstructions.com/2008/12/java-objects-memory-structure.html>

<http://btoddb-java-sizing.blogspot.com/2012/01/object-sizes.html>

<http://stackoverflow.com/questions/2120437/object-vs-byte0-as-lock>