初探 Java 对象头

作者：Gouzhong1223
原文链接：https://ld246.com/article/1583676127416
来源网站：链滴

<img src="https://ld246.com/images/img-loading.svg" alt="gwr8hdzgncY.jpg" data-src="https://b3logfile.com/file/2020/03/gwr8hdzgncY-ff79bc0d.jpg?imageView2/2/interlace/1/format/jpg">
<h2 id="写在前面">写在前面</h2>
在 Java 程序的运行过程中，会被创建出许许多多的对象，这些对象可能是我们自己写的一个类也可能是 JDK 自带的一些类被我们创建了对象。那么，我们创建出来的对象里面都装了些什么呢？
说到装载的东西，我们首先想到的就是里面会装载我们给对象自定义的一些属性，比如下面这个：
<pre><code class="highlight-chroma">public class Student {
    private String name;
}
</code></pre>
我们在创建了这个对象之后，这个存储在堆上的对象就会肯定会包含一个变量 <code>name</code>，拿出了这个对象属性，里面还有什么东西呢？
经过初步资料查询之后，关于新创建的对象，会保存一下数据：

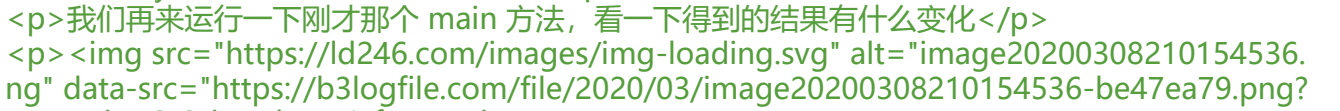
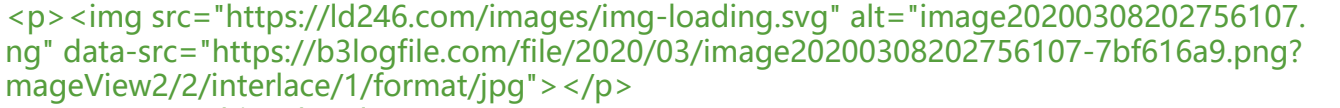
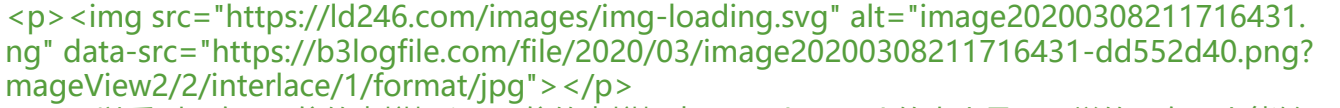
Java 对象头
对象属性
对齐数据填充

<h2 id="什么是数据对齐">什么是数据对齐</h2>
要知道什么是数据对齐，就要先看到一个对象在 Java 中到底是长成什么样子的，这里我们我们用一个 Maven 依赖 jol（Java Object Layout）
<h3 id="查看-Java-对象头">查看 Java 对象头</h3>
<img src="https://ld246.com/images/img-loading.svg" alt="image20200308204353475.png" data-src="https://b3logfile.com/file/2020/03/image20200308204353475-25e0592c.png?imageView2/2/interlace/1/format/jpg">
<pre><code class="highlight-chroma">   <dependency>
        <groupId>org.openjdk.jol</groupId>
        <artifactId>jol-core</artifactId>
        <version>0.10</version>
    </dependency>
</code></pre>
当引入这个以来之后，就可以看到 Java 对象再内存中到底长什么样子，下面来看一下用法：

先看一下我们的测试对象
<pre><code class="highlight-chroma">public class Student {
    private String name;
}
</code></pre>
然后测试代码如下
<pre><code class="highlight-chroma">import com.gouzhong1223.domain.Student;
import org.openj</code></pre>

```
k.jol.info.ClassLayout;
</span></span><span class="highlight-line"><span class="highlight-cl">import org.openj
k.jol.vm.VM;
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">/**
</span></span><span class="highlight-line"><span class="highlight-cl"> * @Author : Gou
hong
</span></span><span class="highlight-line"><span class="highlight-cl"> * @Blog : www.g
uzhong1223.com
</span></span><span class="highlight-line"><span class="highlight-cl"> * @Description :
查看 Java 对象信息
</span></span><span class="highlight-line"><span class="highlight-cl"> * @Date : create
y Gouzhong in 2020-03-02 4:31 下午
</span></span><span class="highlight-line"><span class="highlight-cl"> * @Email : gouzh
ng1223@gmail.com
</span></span><span class="highlight-line"><span class="highlight-cl"> * @Version : 1.0.0
</span></span><span class="highlight-line"><span class="highlight-cl"> */
</span></span><span class="highlight-line"><span class="highlight-cl">public class Demo
 {
</span></span><span class="highlight-line"><span class="highlight-cl">    public static vo
d main(String[] args) {
</span></span><span class="highlight-line"><span class="highlight-cl">        // 创建新对

</span></span><span class="highlight-line"><span class="highlight-cl">        Student stud
nt = new Student();
</span></span><span class="highlight-line"><span class="highlight-cl">        // 查看虚拟
信息
</span></span><span class="highlight-line"><span class="highlight-cl">        System.out.pr
ntln(VM.current().details());
</span></span><span class="highlight-line"><span class="highlight-cl">        // 查看对象
息
</span></span><span class="highlight-line"><span class="highlight-cl">        System.out.pr
ntln(ClassLayout.parseInstance(student).toPrintable());
</span></span><span class="highlight-line"><span class="highlight-cl">    }
</span></span><span class="highlight-line"><span class="highlight-cl">}
</span></span></code></pre>
<p>运行上面的 main 方法，就会得到一下结果：</p>
<p><img src="https://ld246.com/images/img-loading.svg" alt="image20200308205023692.
ng" data-src="https://b3logfile.com/file/2020/03/image20200308205023692-129f55d8.png?
mageView2/2/interlace/1/format/jpg"></p>
<p>红线以上就是我们的虚拟机参数信息，显然我们现在使用的是 64 位的 HotSpot 虚拟机。红线
下就是我们的对象信息，这里可以看到 Object Header 是占用了 12 个 Byte，我们的实例数据 <cod
>name</code> 占用了 4 个 Byte，因为字符串在 Java 中是占用 4 个 Byte 的。但是这里我们没有
到关于数据填充的东西啊，难道 Oracle 的文档是错误的？</p>
<h3 id="为什么没有看到数据对齐-">为什么没有看到数据对齐？</h3>
<p>不急，这里我们把刚才的测试对象升级一下：</p>
<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight
cl">public class Student {
</span></span><span class="highlight-line"><span class="highlight-cl">    private String n
me;
</span></span><span class="highlight-line"><span class="highlight-cl">    private String a
dr;
</span></span><span class="highlight-line"><span class="highlight-cl">}
</span></span></code></pre>
```

按照刚才的逻辑，我们新增了一个字符串字段之后，应该在 16Byte 的基础上增加 4Byte 也就是共占用 20Byte，那么事实真的是这样么？
我们再来运行一下刚才那个 main 方法，看一下得到的结果有什么变化

运行出来的结果居然是 24 Byte，除了新增的 4 Byte，还有一段这样的描述：
<pre><code class="highlight-chroma">20   4   (loss due to the next object alignment)
</code></pre>
意思就是说，虚拟机刚我们在这个对象上补充了 4Byte 的空位置。由于操作系统以及 CPU 的限，我们的程序在运行以及使用的过程中，内存中单个文件的大小最好是 8Byte 的整数倍，按照以往逻，我们这个对象在内存中应该是占有 20Byte 的大小，但是这对于我们的操作系统是非常不变的，所以JVM 就会自动的将对象大小扩容到 8Byte 的整数倍，也就是 24Byte。
这就是数据填充，具体填充多少，这个也是不固定的。
除了数据填充以及对象实例字段，最上面的就是对象头信息。
<h2 id="什么是-Java-对象头">什么是 Java 对象头</h2>
在上面的运行测试中，我们查看了两个不同的对象信息，发现对象头都是占用 12Byte，那这 12 个 Byte 中储存了什么呢？
<h3 id="HotSpot-文档对于对象头的定义">HotSpot 文档对于对象头的定义</h3>
在 HotSpot 的文档中，可以看到，Java 对于对象头是这样定义的：

object header
<blockquote>
<code>Common structure at the beginning of every GC-managed heap object. (Every op points to an object header.) Includes fundamental information about the heap object's layot, type, GC state, synchronization state, and identity hash code. Consists of two words. In arras it is immediately followed by a length field. Note that both Java objects and VM-internal obects have a common object header format.</code>
</blockquote>
上面的意思就是，在 Java 对象头中，储存了堆对象的布局，类型，GC 状态，同步状态和标识希码的基本信息，
这些信息都是储存在一个 <code>Mark Word</code> 里面的，那什么是 Mark Word 呢？</>
<h3 id="关于-Mark-Word">关于 Mark Word</h3>
Mark Word
<blockquote>
<code>The first word of every object header. Usually a set of bitfields including synchronzation state and identity hash code. May also be a pointer (with characteristic low bit encodin) to synchronization related information. During GC, may contain GC state bits.</code>
</blockquote>
每个对象标头的第一个单词。通常，一组位域包括同步状态和标识哈希码。也可以是指向与同步关的信息的指针（具有特征性的低位编码）。在 GC 期间，可能包含 GC 状态位。
打开 HotSpot 的源码，我们可以看到在虚拟机中是如何定义 Mark Word 的

可以看到，在 32 位的虚拟机和 64 位的虚拟机中，Mark Word 的大小是不一样的，在网上能够到写 Mark Word 的文章，基本上都是基于 32 位虚拟机来写的，但是现在已经是 2020 年了，我相已经没有人会用 32 位的电脑了吧，哈哈哈哈。
言归正传，正是因为 32 位的虚拟机和 64 位虚拟机的差异，所以才导致我们在照着代码敲的过中会出现和博客不一样的结果。

<h3 id="对象的状态">对象的状态</h3>
Java 对象有 5 种状态

新创建（无状态）
偏向锁
轻量级所
重量级锁
GC 标记

对应上面 C++ 源码的注释，不难看到，在一个对象是无状态的时候，对应的是下面这种情况：<
p>
<code>unused:25 hash:31 -->| unused:1   age:4   biased_lock:1 lock:2 (normal object
</code>
<h3 id="HashCode-真的存在么-">HashCode 真的存在么？</h3>
回顾一下我们刚刚运行的程序，那个就是一个无状态的对象啊，为什么没有看到 HashCode 呢
难道这个 HotSpot 的源码是假的？不急不急，在探究这个问题之前，我们向来想一想，HashCode
的存在么？？
答案是：不存在！！！！！！
我们来看一下 HashCode 这个方法：

从上面可以看出，HashCode 是一个 native 方法，也就是说这个其实是 C++ 来帮我们实现的
如果我们没有调用这个方法，C++ 怎么知道你要计算 HashCode，所以，如果我们没有为某个对象的
hashCode 方法之前，对象是没有 HashCode 的。
现在来改进一下刚刚那个测试，将代码更改为一下代码：
<pre><code class="highlight-chroma"><span class="highlight
cl">import com.gouzhong1223.domain.Student;
import org.openj
k.jol.info.ClassLayout;
import org.openj
k.jol.vm.VM;

/**
 * @Author : Gou
hong
 * @Blog : www.g
uzhong1223.com
 * @Description :
查看 Java 对象头
 * @Date : create
y Gouzhong in 2020-03-02 4:31 下午
 * @Email : gouzh
ng1223@gmail.com
 * @Since : JDK 1.

 * @ProjectName :
ObjectHeader
 * @Version : 1.0.0
 */
public class Demo
{
    public static vo
d main(String[] args) {

```
</span></span><span class="highlight-line"><span class="highlight-cl">        Student student = new Student();
</span></span><span class="highlight-line"><span class="highlight-cl">        System.out.println(student.hashCode());
</span></span><span class="highlight-line"><span class="highlight-cl">        System.out.println(VM.current().details());
</span></span><span class="highlight-line"><span class="highlight-cl">        System.out.println(ClassLayout.parseInstance(student).toPrintable());
</span></span><span class="highlight-line"><span class="highlight-cl">    }
</span></span><span class="highlight-line"><span class="highlight-cl">}
</span></span></code></pre>
```

在输出对象信息之前，先调用一下这个对象的 hashCode 方法，然后我们再来运行就会得到不样的结果<img src="https://ld246.com/images/img-loading.svg" alt="image20200308213057
02.png" data-src="https://b3logfile.com/file/2020/03/image20200308213057502-132abbad.
ng?imageView2/2/interlace/1/format/jpg">
<h3 id="什么是大端存储">什么是大端存储</h3>
好啦，现在就可以看到这个地方是不为 0 的了，但是为什么这一串数据看起来怪怪的啊？难道是器出 Bug 了？？？　No No No，解释这个之前，这里又要引入一个新的概念：
在 Intel X86 架构的处理器中，我们的计算机是采用大端存储模式的，什么是大端存储模式呢？<
p>
意思就是说，高字节，保存在低地址中，低字节保存在高地址中，我的这一台电脑就是 intel 的PU，所以我们在分析对象头的时候，**应该倒着看！**倒着看那就好理解多了，除去那 25 个未使用，剩下的就是 HashCode，GC 分代年龄，锁状态，锁标记。
<img src="https://ld246.com/images/img-loading.svg" alt="image20200308214256954.
ng" data-src="https://b3logfile.com/file/2020/03/image20200308214256954-fbc432c4.png?i
ageView2/2/interlace/1/format/jpg">
那么问题又来了，为什么除了一个锁标记，还要有一个锁状态呢？
很简单，我们看一下，这个所标记占有 2 个 bit，两个 bit 能够表示的状态，只有 4 种

00
01
10
11

但是我们上面说了，对象有五种状态，这个显然是不能够完全表示的，所以就有一个锁状态来辅标记
比如说新创建的时候就是 0 00
偏向锁的时候就是　1 00
这样就能多表示一种状态了。