学无止境

昵称： duanxz
园龄： 8年5个月
粉丝： 1308
关注： 18
+加关注

< 2019年8月 >

日	一	二	三	四	五	六
28	29	30	31	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31
1	2	3	4	5	6	7

搜索

找找看

谷歌搜索

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签

我的标签

spring(128)
mysql(121)
linux命令(85)
springcloud(64)
monitor(63)
多线程(60)
distributed(55)
juc(55)
springboot(54)
JVM虚拟机(52)
更多

随笔分类

high-concurrence(5)
alibaba(22)
Android(2)
android开源库应用
annotation(7)
Application Server(34)
app-server管理与优化(3)
Architecture(39)
axis2(1)
build(11)
cache(53)
cache-memcache(20)
cache-redis(10)
configserver(1)
configuration-center(2)
consistency(20)
CXF(1)
db操作(3)

博客园 首页 新随笔 联系 订阅 管理

随笔- 2038 评论- 168 文章- 4

java对象在内存中的结构（HotSpot虚拟机）

一、对象的内存布局

HotSpot虚拟机中，对象在内存中存储的布局可以分为三块区域：对象头（Header）、实例数据（Instance Data）和对齐填充（Padding）。

从上面的这张图里面可以看出，对象在内存中的结构主要包含以下几个部分：

- Mark Word(标记字段)：对象的Mark Word部分占4个字节，其内容是一系列的标记位，比如轻量级锁的标记位，偏向锁标记位等等。
- Klass Pointer（Class对象指针）：Class对象指针的大小也是4个字节，其指向的位置是对象对应的Class对象（其对应的元数据对象）的内存地址
- 对象实际数据：这里面包括了对象的所有成员变量，其大小由各个成员变量的大小决定，比如：byte和boolean是1个字节，short和char是2个字节，int和float是4个字节，long和double是8个字节，reference是4个字节
- 对齐：最后一部分是对齐填充的字节，按8个字节填充。

1.1、对象头

1.1.1、Mark Word（标记字段）

HotSpot虚拟机的对象头包括两部分信息，第一部分是**“Mark Word”**，用于存储对象自身的运行时数据， 如哈希码（HashCode）、GC分代年龄、锁状态标志、线程持有的锁、偏向线程ID、偏向时间戳等等，这部分数据的长度在32位和64位的虚拟机（暂 不考虑开启压缩指针的场景）中分别为32个和64个Bits，官方称它为“Mark Word”。对象需要存储的运行时数据很多，其实已经超出了32、64位Bitmap结构所能记录的限度，但是对象头信息是与对象自身定义的数据无关的额 外存储成本，考虑到虚拟机的空间效率，Mark Word被设计成一个非固定的数据结构以便在极小的空间内存储尽量多的信息，它会根据对象的状态复用自己的存储空间。例如在32位的HotSpot虚拟机 中对象未被锁定的状态下，Mark Word的32个Bits空间中的25Bits用于存储对象哈希码（HashCode），4Bits用于存储对象分代年龄，2Bits用于存储锁标志 位，1Bit固定为0，在其他状态（轻量级锁定、重量级锁定、GC标记、可偏向）下对象的存储内容如下表所示。

32位虚拟机			
25Bit	4Bit	1Bit	2Bit
对象的hashCode	对象的分代年龄	是否是偏向锁	锁标志位

但是如果对象是数组类型，则需要三个机器码，因为JVM虚拟机可以通过Java对象的元数据信息确定Java对象的大小，但是无法从数组的元数据来确认数组的大小，所以用一块来记录数组长度。

对象头信息是与对象自身定义的数据无关的额外存储成本，但是考虑到虚拟机的空间效率，Mark Word被设计成一个非固定的数据结构以便在极小的空间内存储尽量多的数据，它会根据对象的状态复用自己的存储空间，也就是说，Mark Word会随着程序的运行发生变化，变化状态如下（32位虚拟机）：

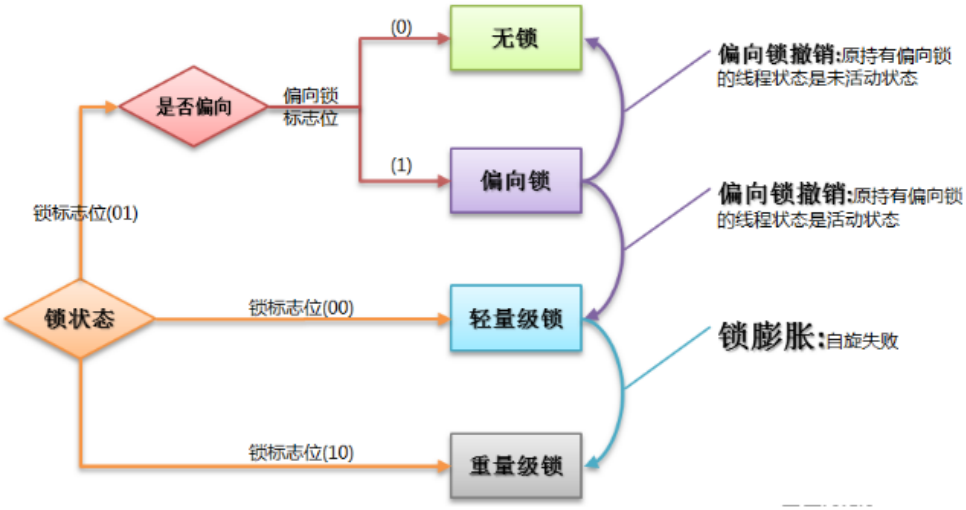
锁状态	25 bit		4bit	1bit	2bit
	23bit	2bit		是否是偏向锁	锁标志位
轻量级锁	指向栈中锁记录的指针				00
重量级锁	指向互斥量（重量级锁）的指针				10
GC标记	空				11
偏向锁	线程ID	Epoch	对象分代年龄	1	01

无锁状态 ： 对象的HashCode + 对象分代年龄 + 状态位001

表1 HotSpot虚拟机对象头Mark Word

存储内容	标志位	状态
对象哈希码、对象分代年龄	01	未锁定
指向锁记录的指针	00	轻量级锁定
指向重量级锁的指针	10	膨胀（重量级锁定）
空，不需要记录信息	11	GC标记
偏向线程ID、偏向时间戳、对象分代年龄	01	可偏向

注意偏向锁、轻量级锁、重量级锁等都是jdk 1.6以后引入的。



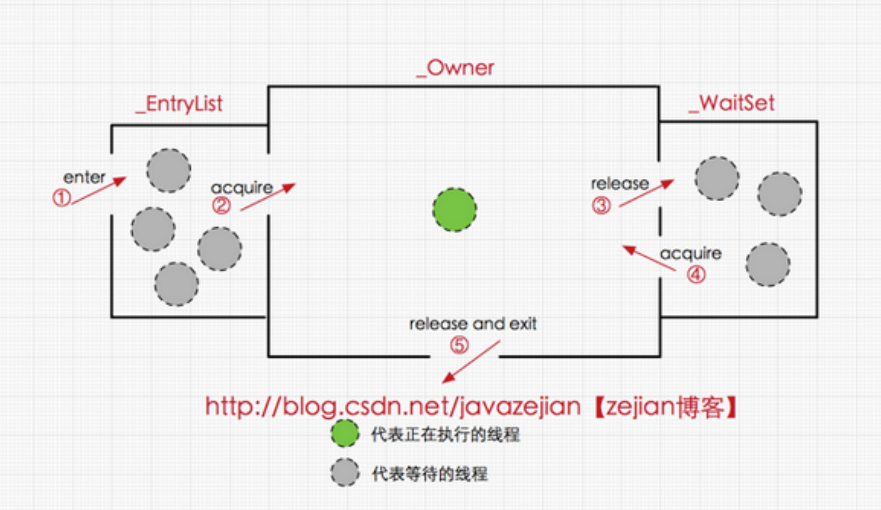
synchronized的细分

其中轻量级锁和偏向锁是Java 6 对 synchronized 锁进行优化后新增加的，稍后我们会简要分析。这里我们主要分析一下重量级锁也就是通常说synchronized的对象锁，锁标识位为10，其中指针指向的是monitor对象（也称为管程或监视器锁）的起始地址。每个对象都存在着一个 **monitor** 与之关联，对象与其 monitor 之间的关系有存在多种实现方式，如monitor可以与对象一起创建销毁或当线程试图获取对象锁时自动生成，但当一个 monitor 被某个线程持有后，它便处于锁定状态。在Java虚拟机(HotSpot)中，monitor是由ObjectMonitor实现的，其主要数据结构如下（位于HotSpot虚拟机源码ObjectMonitor.hpp文件，C++实现的）

```
ObjectMonitor() {
    _header       = NULL;
    _count        = 0; //记录个数
    _waiters      = 0,
    _recursions   = 0;
    _object       = NULL;
```

```
    _owner        = NULL;
    _WaitSet      = NULL; //处于wait状态的线程，会被加入到_WaitSet
    _WaitSetLock  = 0 ;
    _Responsible  = NULL ;
    _succ         = NULL ;
    _cxq          = NULL ;
    FreeNext      = NULL ;
    _EntryList    = NULL ; //处于等待锁block状态的线程，会被加入到该列表
    _SpinFreq     = 0 ;
    _SpinClock    = 0 ;
    OwnerIsThread = 0 ;
  }
```

ObjectMonitor中有两个队列，_WaitSet 和 _EntryList，用来保存ObjectWaiter对象列表(每个等待锁的线程都会被封装成ObjectWaiter对象)，_owner指向持有ObjectMonitor对象的线程，当多个线程同时访问一段同步代码时，首先会进入 _EntryList 集合，当线程获取到对象的monitor 后进入 _Owner 区域并把monitor中的owner变量设置为当前线程同时monitor中的计数器count加1，若线程调用 wait() 方法，将释放当前持有的monitor，owner变量恢复为null，count自减1，同时该线程进入 WaitSe t集合中等待被唤醒。若当前线程执行完毕也将释放monitor(锁)并复位变量的值，以便其他线程进入获取monitor(锁)。如下图所示



由此看来，monitor对象存在于每个Java对象的对象头中(存储的指针的指向)，synchronized锁便是通过这种方式获取锁的，也是为什么Java中任意对象可以作为锁的原因，同时也是notify/notifyAll/wait等方法存在于顶级对象Object中的原因(关于这点稍后还会进行分析)，ok~，有了上述知识基础后，下面我们将进一步分析synchronized在字节码层面的具体语义实现。

对象头的另外一部分是类型指针，即是对象指向它的类的元数据的指针，虚拟机通过这个指针来确定这个对象是哪个类的实例。并不是所有的虚拟机实现都必须在对象数据上保留类型指针，换句话说查找对象的元数据信息并不一定要经过对象本身。另外，如果对象是一个Java数组，那在对象头中还必须有一块用于记录数组长度的数据，因为虚拟机可以通过普通Java对象的元数据信息确定Java对象的大小，但是从数组的元数据中无法确定数组的大小。
以下是HotSpot虚拟机markOop.cpp中的C++代码（注释）片段，它描述了32bits下MarkWord的存储状态：

```
// Bit-format of an object header (most significant first, big endian layout below):
//
//  32 bits:
//  --------
//  hash:25 ------------>| age:4    biased_lock:1 lock:2 (normal object)
//  JavaThread*:23 epoch:2 age:4    biased_lock:1 lock:2 (biased object)
//  size:32 ------------------------------------------>| (CMS free block)
//  PromotedObject*:29 ---------->| promo_bits:3 ----->| (CMS promoted object)
```

1.2、实例数据（Instance Data）

接下来实例数据部分是对象真正存储的有效信息，也既是我们在程序代码里面所定义的各种类型的字段内容，无论是从父类继承下来的，还是在子类中定义的都需要记录下来。 这部分的存储顺序会受到虚拟机分配策略参数（FieldsAllocationStyle）和字段在Java源码中定义顺序的影响。HotSpot虚拟机 默认的分配策略为longs/doubles、ints、shorts/chars、bytes/booleans、oops（Ordinary Object Pointers），从分配策略中可以看出，相同宽度的字段总是被分配到一起。在满足这个前提条件的情况下，在父类中定义的变量会出现

spring原理(5)
spring源码(3)
sql(5)
sqlrecord(9)
sql优化(11)
Sqoop
Storm(15)
struts1(6)
struts2(8)
tcp(7)
temp(20)
testing(29)
thread(18)
threadpool(23)
time-series-database(4)
tomcat(3)
tools(5)
transaction(27)
ubuntu(12)
UML(12)
validator(6)
web(11)
Web-Attacks(19)
webService(7)
work(1)
xml解析(7)
zookeeper(21)
大数据(11)
调试技巧(2)
多线程(61)
分布式事务(5)
工具类库(19)
关注经济(2)
函数式编程(1)
集群-架构(17)
建模(1)
开源包使用(3)
开源组件(11)
面试题(25)
软件质量(12)
设计开发(2)
事件定时器(3)
算法(23)
索引(10)
网络编程(46)
网络通信(30)
项目管理(7)
性能分析及调优(18)
序列化框架(5)
源码(31)
孕妇营养(3)
杂七杂八(2)

随笔档案

2019年8月(9)
2019年7月(8)
2019年6月(1)
2019年5月(3)
2019年4月(6)

在子类之前。如果 CompactFields参数值为true（默认为true），那子类之中较窄的变量也可能会插入到父类变量的空隙之中。

1.3、对齐填充（Padding）

第三部分对齐填充并不是必然存在的，也没有特别的含义，它仅仅起着占位符的作用。由于HotSpot VM的自动内存管理系统要求对象起始地址必须是8字节的整数倍，换句话说就是对象的大小必须是8字节的整数倍。对象头正好是8字节的倍数（1倍或者2倍），因此当对象实例数据部分没有对齐的话，就需要通过对齐填充来补全。

二、对象的创建过程

Java是一门面向对象的编程语言，Java程序运行过程中无时无刻都有对象被创建出来。在语言层面上，创建对象通常（例外：克隆、反序列化）仅仅是一个 new关键字而已，而在虚拟机中，对象（本文中讨论的对象限于普通Java对象，不包括数组和Class对象等）的创建又是怎样一个过程呢？
虚拟机遇到一条new指令时，
1、首先jvm要检查类A是否已经被加载到了内存，即类的符号引用是否已经在常量池中，并且检查这个符号引用代表的类是否已被加载、解析和初始化过的。如果还没有，需要先触发类的加载、解析、初始化。然后在堆上创建对象。

2、为新生对象分配内存。

对象所需内存的大小在类加载完成后便可完全确定，为对象分配空间的任务具体便等同于一块确定大小的内存从Java堆中划分出来，怎么划呢？假设Java堆中内存是绝对规整的，所有用过的内存都被放在一边，空闲的内存被放在另一边，中间放着一个指针作 为分界点的指示器，那所分配内存就仅仅是把那个指针向空闲空间那边挪动一段与对象大小相等的距离，这种分配方式称为“指针碰撞”（Bump The Pointer）。如果Java堆中的内存并不是规整的，已被使用的内存和空闲的内存相互交错，那就没有办法简单的进行指针碰撞了，虚拟机就必须维护一个列表，记录上哪些内存块是可用的，在分配的时候从列表中找到一块足够大的空间划分给对象实例，并更新列表上的记录，这种分配方式称为“空闲列表”（Free List）。选择哪种分配方式由Java堆是否规整决定，而Java堆是否规整又由所采用的垃圾收集器是否带有压缩整理功能决定。因 此在使用Serial、ParNew等带Compact过程的收集器时，系统采用的分配算法是指针碰撞，而使用CMS这种基于Mark-Sweep算法的 收集器时（说明一下，CMS收集器可以通过UseCMSCompactAtFullCollection或CMSFullGCsBeforeCompaction来整理内存），就通常采用空闲列表。
除如何划分可用空间之外，还有另外一个需要考虑的问题是对象创建在虚拟机中是非常频繁的行为，即使是仅仅修改一个指针所指向的位置，在并发情况下也并不是 线程安全的，可能出现正在给对象A分配内存，指针还没来得及修改，对象B又同时使用了原来的指针来分配内存。解决这个问题有两个方案，一种是对分配内存空 间的动作进行同步——实际上虚拟机是采用CAS配上失败重试的方式保证更新操作的原子性；另外一种是把内存分配的动作按照线程划分在不同的空间之中进行， 即每个线程在Java堆中预先分配一小块内存，称为本地线程分配缓冲区，（TLAB ，Thread Local Allocation Buffer），哪个线程要分配内存，就在哪个线程的TLAB上分配，只有TLAB用完，分配新的TLAB时才需要同步锁定。虚拟机是否使用TLAB，可以通过-XX:+/-UseTLAB参数来设定。

3. 完成实例数据部分的初始化工作（初始化为0值）

内存分配完成之后，虚拟机需要将分配到的内存空间都初始化为零值（不包括对象头），如果使用TLAB的话，这一个工作也可以提前至TLAB分配时进行。这 步操作保证了对象的实例字段在Java代码中可以不赋初始值就直接使用，程序能访问到这些字段的数据类型所对应的零值。

4、 完成对象头的填充：如对象自身的运行时数据、类型指针等。

接下来，虚拟机要对对象进行必要的设置，例如这个对象是哪个类的实例、如何才能找到类的元数据信息、对象的哈希码、对象的GC分代年龄等信息。这些信息存放在对象的对象头（Object Header）之中。根据虚拟机当前的运行状态的不同，如是否启用偏向锁等，对象头会有不同的设置方式。

在上面工作都完成之后，在虚拟机的视角来看，一个新的对象已经产生了。但是在Java程序的视角看来，初始化才正式开始，开始调用<init>方法完成初始复制和构造函数，所有的字段都为零值。因此一般来说（由字节码中是否跟随有invokespecial指令所决定），new指令之后会接着就是执 行<init>方法，把对象按照程序员的意愿进行初始化，这样一个真正可用的对象才算完全创建出来。

下面代码是HotSpot虚拟机bytecodeInterpreter.cpp中的代码片段（这个解释器实现很少机会实际使用，大部分平台上都使用模板 解释器；当代码通过JIT编译器执行时差异就更大了。不过这段代码用于了解HotSpot的运作过程是没有什么问题的）。

```
// 确保常量池中存放的是已解释的类
    if (!constants->tag_at(index).is_unresolved_klass()) {
      // 断言确保是klassOop和instanceKlassOop（这部分下一节介绍）
      oop entry = (klassOop) *constants->obj_at_addr(index);
      assert(entry->is_klass(), "Should be resolved klass");
      klassOop k_entry = (klassOop) entry;
      assert(k_entry->klass_part()->oop_is_instance(), "Should be instanceKlass");
      instanceKlass* ik = (instanceKlass*) k_entry->klass_part();
      // 确保对象所属类型已经经过初始化阶段
      if ( ik->is_initialized() && ik->can_be_fastpath_allocated() ) {
        // 取对象长度
        size_t obj_size = ik->size_helper();
```

```
        oop result = NULL;
        // 记录是否需要将对象所有字段置零值
        bool need_zero = !ZeroTLAB;
        // 是否在TLAB中分配对象
        if (UseTLAB) {
          result = (oop) THREAD->tlab().allocate(obj_size);
        }
        if (result == NULL) {
          need_zero = true;
          // 直接在eden中分配对象
    retry:
          HeapWord* compare_to = *Universe::heap()->top_addr();
          HeapWord* new_top = compare_to + obj_size;
          // cmpxchg是x86中的CAS指令，这里是一个C++方法，通过CAS方式分配空间，并发失败的话，转到retry中重试直至成功分配为止
          if (new_top <= *Universe::heap()->end_addr()) {
            if (Atomic::cmpxchg_ptr(new_top, Universe::heap()->top_addr(), compare_to) != compare_to) {
              goto retry;
            }
            result = (oop) compare_to;
          }
        }
        if (result != NULL) {
          // 如果需要，为对象初始化零值
          if (need_zero ) {
            HeapWord* to_zero = (HeapWord*) result + sizeof(oopDesc) / oopSize;
            obj_size -= sizeof(oopDesc) / oopSize;
            if (obj_size > 0 ) {
              memset(to_zero, 0, obj_size * HeapWordSize);
            }
          }
          // 根据是否启用偏向锁，设置对象头信息
          if (UseBiasedLocking) {
            result->set_mark(ik->prototype_header());
          } else {
            result->set_mark(markOopDesc::prototype());
          }
          result->set_klass_gap(0);
          result->set_klass(k_entry);
          // 将对象引用入栈，继续执行下一条指令
          SET_STACK_OBJECT(result, 0);
          UPDATE_PC_AND_TOS_AND_CONTINUE(3, 1);
        }
      }
    }
```

三、对象的访问定位

建立对象是为了使用对象，我们的Java程序需要通过栈上的reference数据来操作堆上的具体对象。由于reference类型在Java虚拟机规范里面只规定了是一个指向对象的引用，并没有定义这个引用应该通过什么种方式去定位、访问到堆中的对象的具体位置，对象访问方式也是取决于虚拟机实现而定的。主流的访问方式有使用句柄和直接指针两种。

如果使用句柄访问的话，Java堆中将会划分出一块内存来作为句柄池，reference中存储的就是对象的句柄地址，而句柄中包含了对象实例数据与类型数据的具体各自的地址信息。如图1所示。

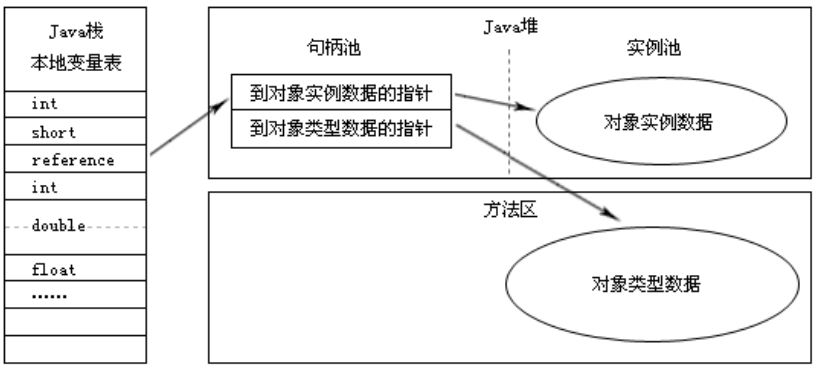


图1 通过句柄访问对象

2013年10月(9)
2013年9月(5)
2013年8月(6)
2013年7月(19)
2013年6月(32)
2013年5月(15)
2013年4月(9)
2013年3月(13)
2013年2月(17)
2013年1月(35)
2012年12月(39)
2012年11月(45)
2012年10月(55)
2012年9月(17)
2012年8月(45)
2012年7月(29)
2012年6月(19)
2012年5月(9)

文章分类

cloud(2)
mysql(1)
web

最新评论

1. Re:bash: chkconfig: command not found
可以
--smart-fox
2. Re:nginx限制请求之一：（ngx_http_limit_conn_module）模块
请问这个 limit_conn_zone 配置以后进行压测，没有效果是因为什么？跟版本有关系吗？
--吴振照
3. Re:深入ThreadLocal之三（ThreadLocal可能引起的内存泄露）
博主，我挖坟来了~
为啥只有key才有弱引用呢？这样的设计莫非是为了避免重复创建大对象的消耗？但key被回收后threadLocal调用get时也会清掉value呀？请解
--seekforwings
4. Re:SpringBoot读取application.properties文件
666666
--规格严格-功夫到家
5. Re:docker常用命令详解
@ 倾杯倒月docker的其它命令能正常使用吗？...
--duanxz
6. Re:HBase之八--(3)：Hbase 布隆过滤器BloomFilter介绍
写得不错
--cq_fuqq
7. Re:docker常用命令详解
你好，我使用since命令的时候，提示：flag provided but not defined: --sinceSee 'docker logs --help'.能告诉我如何解决吗？...
--倾杯倒月
8. Re:linux 中的局部变量、全局变量、shell 变量的总结
非常感谢分享！
--晓风残月1994

如果使用直接指针访问的话，Java堆对象的布局中就必须考虑如何放置访问类型数据的相关信息，reference中存储的直接就是对象地址，如图2所示。

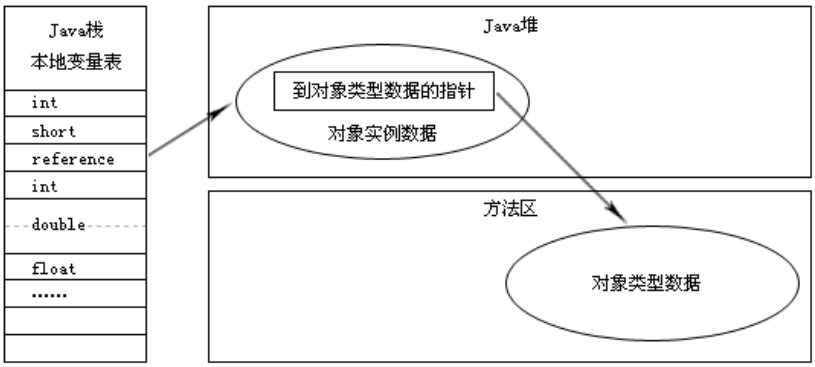


图2 通过直接指针访问对象

这两种对象访问方式各有优势，使用句柄来访问的最大好处就是reference中存储的是稳定句柄地址，在对象被移动（垃圾收集时移动对象是非常普遍的行为）时只会改变句柄中的实例数据指针，而reference本身不需要被修改。

使用直接指针来访问最大的好处就是速度更快，它节省了一次指针定位的时间开销，由于对象访问的在Java中非常频繁，因此这类开销积小成多也是一项非常可观的执行成本。从上一部分讲解的对象内存布局可以看出，就虚拟机HotSpot而言，它是使用第二种方式进行对象访问，但在整个软件开发的范围来看，各种语言、框架中使用句柄来访问的情况也十分常见。

四、示例

在Hotspot JVM中，32位机器下，Integer对象的大小是int的几倍？

我们都知道在Java语言规范已经规定了int的大小是4个字节，那么Integer对象的大小是多少呢？要知道一个对象的大小，那么必须需要知道对象在虚拟机中的结构是怎样的，根据上面的图，那么我们可以得出Integer的对象的结构如下：

Integer只有一个int类型的成员变量value，所以其对象实际数据部分的大小是4个字节，然后再在后面填充4个字节达到8字节的对齐，所以可以得出Integer对象的大小是16个字节。

因此，我们可以得出**Integer**对象的大小是原生的**int**类型的**4**倍。

关于对象的内存结构，需要注意数组的内存结构和普通对象的内存结构稍微不同，因为数据有一个长度length字段，所以在对象头后面还多了一个int类型的length字段，占4个字节，接下来才是数组中的数据，如下图：

9. Re:Redis 发布/订阅机制原理分析
哥，图挂咯，麻烦放一下，小弟谢谢了
--偷偷学习被我发现
10. Re:MyBatis association的两种形式——MyBatis学习笔记之四
照你这么写XML文件头会报错
--同是天涯
11. Re:spring4.0之二：@Configuration的使用
@ 有点懒惰的小青年
applicationContext-configuration.xml这是一个例子，说明Spring支持配置类和XML两种模式混合使用。...
--手机疯
12. Re:spring4.0之二：@Configuration的使用
感谢，写的很详细，有xml的对比！太适合新手了。
--手机疯
13. Re:美的支付-对账系统实现
@ leee123可以。但我的对账需求，需要明确差错单的不同原因：单号对不上，单号对上了，但金额对不上等场景...
--duanxz
14. Re:美的支付-对账系统实现
@ 漂泊一剑客如果对账的数据比较多，mysql可能就用性能问题了...
--duanxz
15. Re:美的支付-对账系统实现
博主，直接用数据库的 关联两张表来进行update，也不错啊，这样的速度也还可以
--漂泊一剑客
16. Re:shell函数（调用、返回值，返回值获取）
重点
2）用$?来接收函数的执行状态，但是$?要紧跟在函数调用处的后面。
--RMS365
17. Re:自己写的基于java Annotation(注解)的数据校验框架
大哥有没有代码可以参考啊
--时光再见
18. Re:kafka之六：为什么Kafka那么快
mark
--lxq159
19. Re:consul之：ACL配置使用
呸，抄袭
--王振耀
20. Re:nginx利用proxy_cache来缓存文件
请问在http下通过proxy_cache配置开启了缓存，怎么在在location下局部关闭?
--轻轻织锦缎
21. Re:RestTemplate实践
@ BigOrang这个是最6的，spring真的很强。哈哈...
--六壬
22. Re:JAVA8 十大新特性详解
学习
--石牌桥网管
23. Re:spring4.0之二：@Configuration的使用
学习啦，谢谢~
--B.J
24. Re:mongoTemplate操作内嵌文档
非常清晰，正好碰到类似问题有幸查到了楼主问题，完美解决。感谢感谢~~
另请问楼主，您是从官方文档里验

关于对象内存布局更多的内容，可以看这篇文章：Java Objects Memory Structure

分类: 性能分析及调优, lock, 多线程, JVM虚拟机

标签: 多线程, lock, synchronized

duanxz
关注 - 18
粉丝 - 1308
+加关注

3 0

« 上一篇： 优化Java堆大小的5个技巧
» 下一篇： 架构师的职责都有哪些?

posted on 2015-11-15 18:45 duanxz 阅读(6028) 评论(2) 编辑 收藏

评论:

#1楼 2018-11-27 00:39 | IQ19
跪赞，MARK
支持(0) 反对(0)

#2楼 2019-04-04 17:26 | damayi
怒赞，收益匪浅
支持(0) 反对(0)

刷新评论 刷新页面 返回顶部

注册用户登录后才能发表评论，请 登录 或 注册， 访问 网站首页。

证出这个知识点的，还是什么其他渠道。谢谢~

--qqer2002

25. Re:性能测试，负载测试，压力测试，容量测试的区别

亲，原谅一个小白的无知，能给说下哪本书写的压力测试，容量测试，负载测试的区别呢

--bowenma

阅读排行榜

1. spring4.0之二：@Configuration的使用(184217)
2. java类中serialversionuid 作用 是什么?举个例子说明(93732)
3. JSONArray数据转换成java List(91699)
4. Maven详解之仓库------本地仓库、远程仓库(66750)
5. springboot中使用@Value读取配置文件(65560)

评论排行榜

1. spring4.0之二：@Configuration的使用(15)
2. UML类图符号 各种关系说明以及举例(7)
3. 美的支付-对账系统实现(6)
4. mina框架详解(5)
5. memcache分布式部署的原理分析(4)

推荐排行榜

1. spring4.0之二：@Configuration的使用(33)
2. UML类图符号 各种关系说明以及举例(19)
3. CAP原则(CAP定理)、BASE理论(13)
4. java类中serialversionuid 作用 是什么?举个例子说明(10)
5. 服务链路追踪(Spring Cloud Sleuth)(7)

相关博文：

· 对象内存结构(hotspot)
· 深入理解JVM——hotspot虚拟机对象探秘
· 深入理解java虚拟机（二）HotSpot Java对象创建，内存布局以及访问方式
· Java虚拟机(二)---HotSpot虚拟机对象探秘
· java 对象创建