**Java 堆和栈、垃圾回收**

2018-09-23 17:31:19 [希尤](https://me.csdn.net/qq_39455116) 阅读数 819更多

分类专栏： [Java](https://blog.csdn.net/qq_39455116/article/category/7464517)

版权声明：本文为博主原创文章，遵循 [CC 4.0 BY-SA](http://creativecommons.org/licenses/by-sa/4.0/" \t "_blank) 版权协议，转载请附上原文出处链接和本声明。

本文链接：<https://blog.csdn.net/qq_39455116/article/details/82822356>

**简单的解释一下垃圾回收**

Java 垃圾回收机制最基本的做法是分代回收。内存中的区域被划分成不同的世代，对象根据其存活的时间被保存在对应世代的区域中

一般的实现是划分成3个世代：年轻、年老和永久。内存的分配是发生在年轻世代中的。当一个对象存活时间足够长的时候，  
它就会被复制到年老世代中。对于不同的世代可以使用不同的垃圾回收算法。  
进行世代划分的出发点是对应用中对象存活时间进行研究之后得出的统计规律。一般来说，一个应用中的大部分对象的存活时间都很短。  
比如局部变量的存活时间就只在方法的执行过程中。基于这一点，对于年轻世代的垃圾回收算法就可以很有针对性. 如何确定一个对象是否可以被回收？

所有新生成的对象首先都是放在年轻代的。年轻代的目标就是尽可能快速的收集掉那些生命周期短的对象  
在年轻代中经历了N（15）次垃圾回收后仍然存活的对象，就会被放到年老代中。 用于存放静态文件，如今Java类、方法等。

**引用计数算法：判断对象的引用数量 任何引用计数为0的对象实例可以被当作垃圾收集**

1. 标记——清除算法  
   1、标记所有需要回收的对象2、标记完成后，清除被标记的对象。
   1. 标记——复制算法
   2. 标记——复制存储算法通过采用双区域交替使用这种方式解决了标记——清除算法中效率低下的问题。
   3. 1.存活的对象将被复制到另外一块区域。2.原先被使用的区域被重置，转为空闲区
      * 1
      * 2
   4. 标记——整理算法

标记-复制算法在对象存活率较高的情况下就要进行较多的复制操作，更重要的是该算法浪费一般的内存空间，为了解决该问题，  
出现了标记——整理算法：  
其标记的过程和“标记-清除”算法一样，而整理的过程则是让所有存活的对象都想另外一端移动，然后直接清理掉端边界以外的内存。

1. 4.分代收集算法
2. 新生代，大多都会死去《《《《《 标记——复制算法
3. 老年代： 标记-清除 标记-复制
   * 1
   * 2

5.增量收集算法  
以上所述的算法，都存在一个缺点：在进行垃圾回首时需要暂停当前应用的执行，  
也就是这时候的垃圾回收线程不能和应用线程同时运行  
这也是增量收集算法的目标，即在不中断应用线程的状态下垃圾回收线程也能进行垃圾回收。

**但是垃圾回收的知识堆内存，栈内存是JVM自动管理的，栈的内存都是随着函数的开始执行和结束自动分配，释放的**

堆是由垃圾回收来负责的，堆的优势是可以动态地分配内存大小，生存期也不必事先告诉编译器，  
因为它是在运行时动态分配内存的，Java的垃圾收集器会自动收走这些不再使用的数据。  
但缺点是，由于要在运行时动态分配内存，存取速度较慢。

在JVM所管理的内存中，堆区是最大的一块，堆区也是Java GC机制所管理的主要内存区域，**堆区由所有线程共享**，在虚拟机启动时创建。堆区的存在是为了存储对象实例

栈的优势是，存取速度比堆要快，而且栈数据可以共享。  
但缺点是，存在栈中的数据大小与生存期必须是确定的，缺乏灵活性。栈中主要存放一些基本类型的变量（,int,  
short, long, byte, float, double, boolean,  
char）和对象句柄。栈有一个很重要的特殊性，就是存在**栈中的数据可以共享**。  
每个线程对应着一个虚拟机栈，因此虚拟机栈也是线程私有的。

栈和线程的关系

栈：线程私有，每个线程都会创建一个虚拟机栈，生命周期与线程相同。每个方法被执行的时候就会创建一个栈帧，用于存储局部变量表，操作数栈，动态链接，方法出口等信息。一个方法执行的过程对应着一个栈帧的入栈到出栈过程。

虚拟机栈的内存大小会直接影响线程创建的数量。  
假定内存中的堆内存的大小不变都是512M，虚拟机栈的大小会直接影响可创建线程数量的大小。虚拟机栈内存越大可创建的数量越小。  
即：JVM内存 = 堆内存 + 线程数量 \* 栈内存

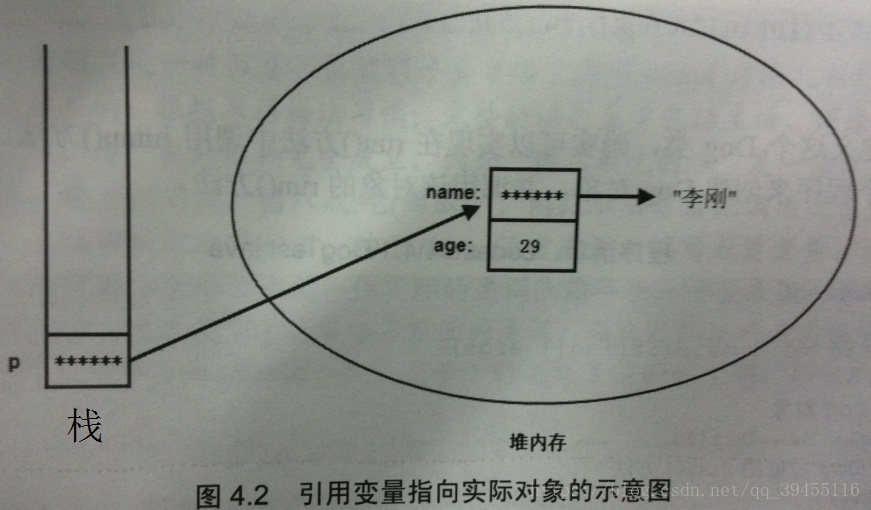
Java运行程序包含：

1. 栈Stack:
2. 保存局部变量的值，包括：
   * 1

1- 保存基本数据类型的值  
2-保存类的实例，即堆区对象的引用  
3-保存加载方法时的祯

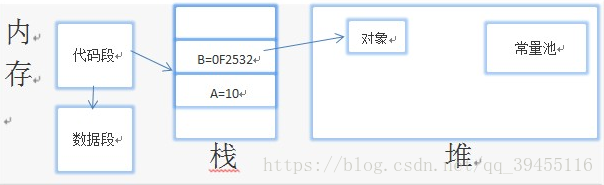
1. 堆Heap
2. 用来存储动态产生的数据
   * 1

1-比如new 出来的对象，注意创建出来的对象只包含各自的成员变量  
不包括成员方法。因为同一个类的对象拥有各自的成员变量，存储在各自的堆中，  
但是他们共享该类的方法，并不是每创建一个对象就把成员方法复制一次。

举例：  
Person p =new Person()  
1- 上面产生了两个东西，一个时变量p ，存放在栈中  
2- 产生了Person对象，存放在堆中  


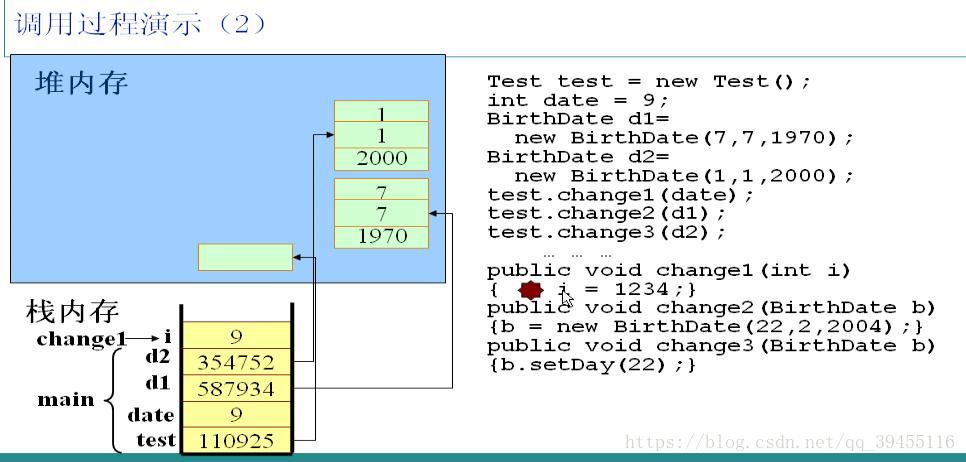
1. 寄存器  
   JVM内部虚拟寄存器，存取速度非常快，程序不可控制。

方法区：Method Area

1. 常量池  
   JVM为每个已加载的类型维护一个常量池，常量池就是这个类型用到的常量的一个有序集合。包括直接常量(基本类型，String)和对其他类型、方法、字段的符号引用(1)。池中的数据和数组一样通过索引访问。由于常量池包含了一个类型所有的对其他类型、方法、字段的符号引用，所以常量池在Java的动态链接中起了核心作用。常量池存在于堆中。
2. 代码段  
   用来存放从硬盘上读取的源程序代码。
3. 数据段  
   用来存放static定义的静态成员。  
   

**总结**  
1.一个Java文件，只要有main入口方法，我们就认为这是一个Java程序，可以单独编译运行。

2.无论是普通类型的变量还是引用类型的变量(俗称实例)，都可以作为局部变量，他们都可以出现在栈中。只不过普通类型的变量在栈中直接保存它所对应的值，而引用类型的变量保存的是一个指向堆区的指针，通过这个指针，就可以找到这个实例在堆区对应的对象。因此，普通类型变量只在栈区占用一块内存，而引用类型变量要在栈区和堆区各占一块内存。



===========================================================================================================================================

1. JVM自动寻找main方法，执行第一句代码，创建一个Test类的实例，在栈中分配一块内存，存放一个指向堆区对象的指针110925。
2. 创建一个int型的变量date，由于是基本类型，直接在栈中存放date对应的值9。
3. 创建两个BirthDate类的实例d1、d2，在栈中分别存放了对应的指针指向各自的对象(对象在堆中)。他们在实例化时调用了有参数的构造方法，因此对象中有自定义初始值。
4. 但是注意有一个change1()方法。  
   调用test对象的change1方法，并且以date为参数。JVM读到这段代码时，检测到i是局部变量，因此会把i放在栈中，并且把date的值赋给i。
5. change1方法执行完毕，立即释放局部变量i所占用的栈空间。
6. 但是如果一个方法必须change2(Person p)需要的是一个对象，而且这个对象是在方法区域内产生的，这时内存情况又该如何分配？  
   change2方法中又实例化了一个Person对象，并且赋给p。在内部执行过程是：在堆区new了一个对象，并且把该对象的指针保存在栈中的p对应空间
7. change2方法执行完毕，立即释放局部引用变量b所占的栈空间，注意只是释放了栈空间，堆空间要等待自动回收。
8. 如果Person p1 =new Person()  
   [p1.name](http://p1.name) =“张三”;  
   test.changName(p1);传入的是一个对象  
   但是：  
   public void changeName (Person bb){  
   bb.setName =“李四”  
   }  
   并没有像6中那样创建一个对象，这时内存该如何分配呢？
9. 调用test实例的change3方法，以实例d2为参数。同理，JVM会在栈中为局部引用变量b分配空间，并且把d2中的指针存放在b中，此时d2和b指向同一个对象。再调用实例b的setDay方法，其实就是调用d2指向的对象的setDay方法。  
   调用实例b的setDay方法会影响d2，因为二者指向的是同一个对象。  
   change3方法执行完毕，立即释放局部引用变量b。

public void change2 (Person p2){

p2 =new Person();

p2.setName("李四");

System.out.println("change2执行："+p2.toString());

}

public void change3 (Person p3){

p3.setName("李四");

System.out.println("change3执行"+p3.toString());

}

public static void main(String[] args) {

Person p =new Person();

p.setName("张三");

System.out.println("1111111111111:"+p.toString());

p.change2(p);

System.out.println("change2之后:"+p.toString());

p.change3(p);

System.out.println("change3执行"+p.toString());

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18

输出结果：

1111111111111:Person{name='张三'}

change2执行：Person{name='李四'}

change2之后:Person{name='张三'}

change3执行Person{name='李四'}

change3执行Person{name='李四'}

* 1
* 2
* 3
* 4
* 5
* 6

**以上就是Java程序运行时内存分配的大致情况。其实也没什么，掌握了思想就很简单了。无非就是两种类型的变量：基本类型和引用类型。二者作为局部变量，都放在栈中，基本类型直接在栈中保存值，引用类型只保存一个指向堆区的指针，真正的对象在堆里。作为参数时基本类型就直接传值，引用类型传指针。**

**小结：**

1.分清什么是实例什么是对象。Class a= new Class();此时a叫实例，而不能说a是对象(new Class()才是对象)。实例在栈中，对象在堆中，操作实例实际上是通过实例的指针间接操作对象。多个实例可以指向同一个对象。

2.栈中的数据和堆中的数据销毁并不是同步的。方法一旦结束，栈中的局部变量立即销毁，但是堆中对象不一定销毁。因为可能有其他变量也指向了这个对象，直到栈中没有变量指向堆中的对象时，它才销毁，而且还不是马上销毁，要等垃圾回收扫描时才可以被销毁。

3.以上的栈、堆、代码段、数据段等等都是相对于应用程序而言的。每一个应用程序都对应唯一的一个JVM实例，每一个JVM实例都有自己的内存区域，互不影响。并且这些内存区域是所有线程共享的。这里提到的栈和堆都是整体上的概念，这些堆栈还可以细分。

4.类的成员变量在不同对象中各不相同，都有自己的存储空间(成员变量在堆中的对象中)。而类的方法却是该类的所有对象共享的，只有一套，对象使用方法的时候方法才被压入栈，方法不使用则不占用内存。

以上分析只涉及了栈和堆，还有一个非常重要的内存区域：常量池，这个地方往往出现一些莫名其妙的问题。常量池是干嘛的上边已经说明了，也没必要理解多么深刻，只要记住它维护了一个已加载类的常量就可以了。接下来结合一些例子说明常量池的特性。

预备知识：

基本类型和基本类型的包装类。

基本类型有：byte、short、char、int、long、boolean。

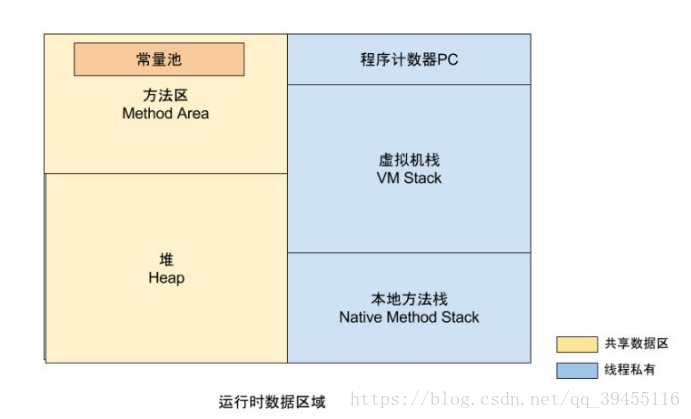
基本类型的包装类分别是：Byte、Short、Character、Integer、Long、Boolean。

注意区分大小写。

二者的区别是：基本类型体现在程序中是普通变量，基本类型的包装类是类，体现在程序中是引用变量。

因此二者在内存中的存储位置不同：基本类型存储在栈中，而基本类型包装类存储在堆中。上边提到的这些包装类都实现了常量池技术，另外两种浮点数类型的包装类则没有实现。另外，String类型也实现了常量池技术。  
什么是成员变量？成员方法？

JVM 详解



堆栈上面分析过了

**方法区：**

（1）被所有线程共享区域，用于存放已被虚拟机加载的类信息，常量，静态变量等数据。  
被Java虚拟机描述为堆的一个逻辑部分。  
习惯是也叫它永久代（仅仅是因为HotSpot虚拟机选择把GC分代收集扩展至方法区）；  
（2）垃圾回收很少光顾这个区域，不过也是需要回收的，主要针对常量池回收，类型卸载。  
（3）会有异常OutOfMemoneyError；线程私有区域

static 修饰的静态变量，什么时候被回收？

在不同的类和包中都可以使用.  
类被加载的时候，静态变量被分配内存，并且在虚拟机中单独占用内存，静态变量在类被卸载的时候才会被销毁，而类只有在进程结束的时候才会被卸载，也就是说被static修饰的静态变量只有在进程被销毁的时候才会被回收

static一般用于公具类，所有人都可以访问，final是不能更改的意思

private static final String configFileName = “fastDFS/client.properties”;

**程序计数器：**

（1）当前线程所执行的字节码指令的行号指示器，如分支、跳转、循环、异常处理、线程恢复都依赖程序计数器实现；  
（2）Java多线程是通过线程轮流切换并分配CPU时间片来执行的，为了线程切换后能恢复到正确的位置，  
所以每个线程都有一个单独的程序计数器，所以程序计数器是私有的； （3）Jvm没有规定OutOfMemory的区块；

**Java虚拟机栈：**

（1）为执行Java方法服务‘  
（2）当线程创建的时候，为线程分配一块内存区域，在线程执行的过程中，  
每个方法的执行都会创建一个栈帧，用于存放局部变量表、操作栈、动态链接，方法出口等。  
每个方法从被调用，直到被执行完。对应着一个栈帧在虚拟机中从入栈到出栈的过程； （3）会有两种异常StackOverFlowError和  
OutOfMemoneyError。 当线程请求栈深度大于虚拟机所允许的深度就会抛出StackOverFlowError错误；  
虚拟机栈动态扩展，当扩展无法申请到足够的内存空间时候，抛出OutOfMemoneyError； （4）它是线程私有的，生命周期与线程相同；

局部变量表存放了编译期可知的各种基本数据类型  
（boolean、byte、char、short、int、float、long、double）、对象引用

**本地方法栈：**

（1）与java虚拟机栈所发挥的作用非常相似，它们之间的区别在于java虚拟机栈执行java方法服务的， 本地方法栈是执行本地方法服务的

**运行时常量池**

运行时常量池（Runtime Constant Pool）是方法区的一部分。 Class 文件中除了有

类的版本、字段、方法、接口等描述等信息外，还有一项信息是常量池（Constant Pool

Table）， 用于存放编译期生成的各种字面量和符号引用，这部分内容将在类加载后存放

到方法区的运行时常量池中。