Java和C++之间有一堵由内存动态分配和垃圾回收机制构成的围墙

2.1运行时数据区域。

JVM在执行Java程序的过程中会把它所管理的内存划分成若干个不同的区域。 这些区域都有各自的用途， 以及创建和销毁时间， 有些区域随着JVM进程的启动而存在， 有些则依赖用户线程的启动和结束而建立和销毁，主要包含： 方法区， 堆， 程序计数器， VM 栈， 本地方法栈，常量池， 直接内存

私有部分

1） 程序计数器： Program counter register 是一块比较小的内存区域， 可以就看成是当前字节码的行号指示器， 字节码解释器就是通过修改它的值， 来实现分支，循环， 跳转， 异常处理， 线程恢复等。。。如果执行的是Java方法， 它则记录虚拟机字节码的地址； 如果执行的是native方法(调用一个非Java语言的方)， 则它的值为空（Undefined）.

2） Java虚拟机栈。 其生命周期与线程同步， 也可以允许动态扩展内存，其描述的是Java方法的执行过程：每个方法在执行时都会创立一个栈帧（stack frame）用以存储局部变量表（在编译期间就已经确定大小， 存放各种基本类型int，long...，以及对象引用） ，操作数栈， 动态链接， 方法出口。每个方法从调用到执行完毕的过程， 就对应着一个栈帧在虚拟机栈中的入栈出栈过程。

3）本地方法栈：与2）类似， 只是存储的是natvie方法的数据， 具体的虚拟机可以具体的去实现， 甚至可以和2）结合为栈

所有线程共有部分

4）Java堆：这是JVM管理的最大一块内存了， 被所有线程共享，在JVM启动时创建， 存放几乎所有的对象实例和数组。 这里也是垃圾回收的主要区域， 有时被戏称为GC 堆。

从垃圾回收角度来看， 这一段可以分作新生代， 老生代： Eden空间， from survive空间， to survive空间；从内存分配角度看， 这里还可以划分出多个线程能私有的分配缓存区， TLAB， 只是在逻辑上是连续的空间

5） 方法区：用以存储已经被JVM 加载的类信息， 常量， 静态变量， JIT即时编译器编译的代码等数据。这个区域我们常常用永久代来实现， 只是为了使GC分代收集能扩展至此， 或是根本不适用GC都可以

6） 运行时常量池：属于方法5）的一部分，存放。class文件中的符号引用（类的版本方法接口等），还有翻译出来的直接引用； 运行时产生的常量也可以存放至此（String的intern（）方法），具备有动态性

7）直接内存。在JDK1.4后引入了新的NIO类， 可以使得Native函数直接分配堆内存， 这个内存DirectByteBuffer对象作为这块内存的使用操作， 避免Java堆和Native堆的来回

copy

2.2 HotS虚拟机对象探秘

2.1中我们了解了java各部分内存存放什么， 现在问问各类数据区域是如何创建， 布局，如何访问的？？？

（1） 对象的创建

在语言层次上，就是一个new 关键字， 但是虚拟机怎么操作的?

1.首先检查这个指令的参数是否能定位到常量池的一个类的符号引用， 再检查这个类是否已经被加载， 解析，初始化，如果没有， 则启动类的加载过程

2.JVM 为新生的对象分配内存。对象的内存大小在编译时即可确定， 等同于把堆上的一块确定大小的内存从堆上分配出去（可以使用指针碰撞法， 空闲列表法）。这个过程可能需要加锁同步（或者是用 TLAB），所有定义数据初始为零值

3.对对象头进行设置。比如是哪一个类的实例， GC分代年龄， 对象的哈希码， 如何找到类的元数据信息。。。

4.对象的初始化。<init>方法要执行， 把对象按照程序员的意愿初始化，这样才最终产生了可用的一个整整对象

（2）对象的内存布局

对象在内存中的存储布局主要分3部分

1. 对象头 Header。定义为与对象本身无关的额外内存， 分为

a. Mark Word。用于存储对象自身运行时候的数据， 包括 GC分代年龄， 运行时数据，哈希码， 持有的锁和锁的标志， 偏向线程ID ，时间戳,数组大小。。。非固定大小

b. 类型指针，指向类元数据， 表明属于那一个类的实例，并非都有

2.实例数据。对象真正存储的有效信息，也是在代码中所定义的各种类型的字段内容。自己的定义部分，也包括从父类继承的。其储存顺序不定， HotSpot虚拟机默认longs/doubles , int , shorts/chars， bytes/boolean , oops(对象指针)，父类 -->子类

3）填充字段。 对象的分配内存起始地址为 8 * n，故需要

（2）对象的访问定位

创建对象是为了使用对象，我们的Java程序需要栈上的reference数据来确定堆上的具体对象。

a.句柄使用。对象改变时， 只用改动句柄， reference不须改变，即稳定的句柄地址

Java在堆上划分出一块内存， 句柄池， reference记载对象的句柄地址， 句柄中包含了对象实例数据与类型数据的地址

b.直接指针。 reference直接存储的对象地址， 一次定位， 速度快