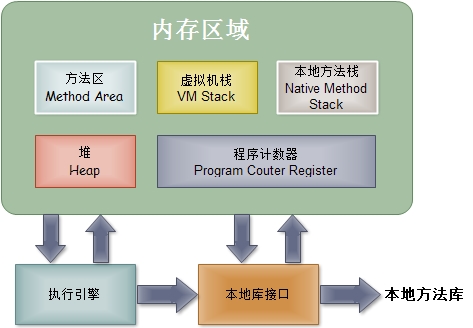
Java堆外内存

Java语言使用Java虚拟机屏蔽了与具体平台相关的信息，使得Java语言编译的程序之需要生成在JVM上运行的目标代码（bytecode），就可以在多种平台上不加修改的运行。在JDK中，编译器将Java文件编译成class文件，然后将class文件输入到jvm中，JVM会加载并执行类文件，运行时数据区分为5个主要组件：

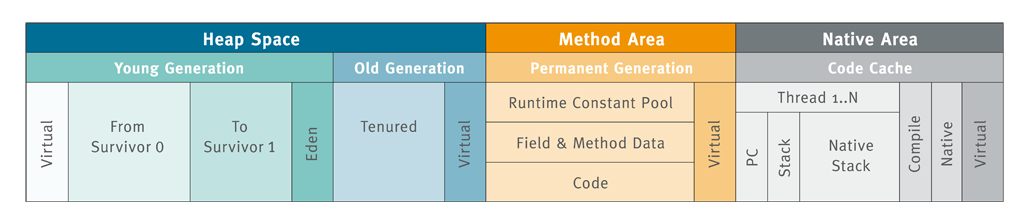


1. 方法区， 所有的类级别的数据都存储在这里，包括静态变量，每个JVM只有一个方法区，并且它是共享资源。类加载器将数据加载到这个数据区中。
2. 堆区域，所有对象及其对应的实例便利和数组都存储在这里，每个JVM只有一个堆区域，由于方法和堆区域共享多个线程的内存，因此所存储的数据非线程安全。
3. 堆栈区，JVM为每个线程都会创建一个单独的运行时栈，对于每个方法调用，将在堆栈存储器中产生一个条目，成为堆栈帧。所有局部变量将在堆栈内存中创建，堆栈区域是线程安全的，不是共享资源。堆栈帧分为三个子元素：

* 局部变量数组，与方法相关，涉及局部变量以及将在此存储的相应值得多少
* 操作数堆栈，如果需要执行任何中间操作，那么操作数堆栈充当工作空间来执行操作
* 帧数据，对应于方法的所有符号存储在此处，在任何异常的情况下，捕捉块信息都被保持在帧数据中。

1. PC寄存器，每个线程都有单独的PC寄存器，用于保存当前执行的指令地址，一旦指令执行，PC寄存器将更新到下一条指令
2. 本地方法堆栈，本地方法堆栈保存本地方法信息，对于每个线程，将创建一个单独的本地方法堆栈

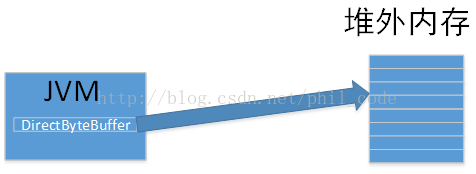
对于JVM，方法区和堆区域是内存分配和管理的主要区域，内存模型如下图：



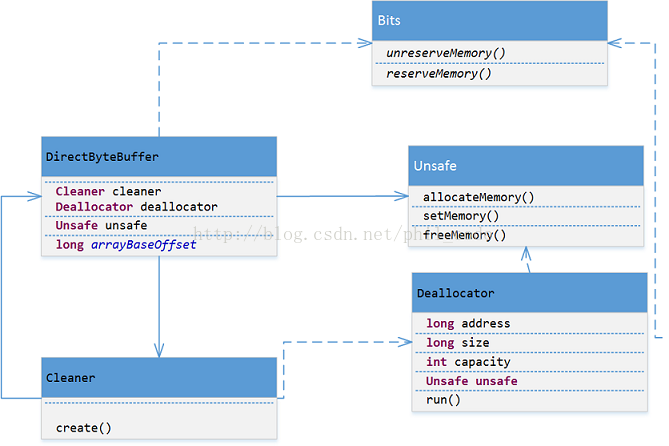
堆区域是运行时数据区域，所有类实例和数组的内存都从此处分配，其在虚拟机启动时创建。但是JVM还可以在堆之外管理一块内存区域，成为堆外内存：

* 堆内存，由JVM的GC自动回收内存，堆的大小可以固定，也可以扩大或者缩小
* 堆外内存，不受JVM控制的内存，直接在堆外分配的内存来存储数据，程序通过JNI直接将数据读写到堆外内存中

其中堆外内存的典型使用类为DirectByteBuffer，其使用方式如下图：



其源码包含这些类：



* Bits，记录申请堆外内存的大小和创建unSafe对象
* Unsafe，申请堆外内存和销毁堆外内存
* Deallocator，堆外内存回收线程
* Cleaner，堆外内存的回收

DirectByteBuffer的使用方法如下：

*ByteBuffer.allocateDirect(int capacity);*

这块区域默认是64MB，可以通过参数-XX:MaxDirectMemorySize控制的。

https://examples.javacodegeeks.com/core-java/nio/bytebuffer/java-direct-bytebuffer-example/