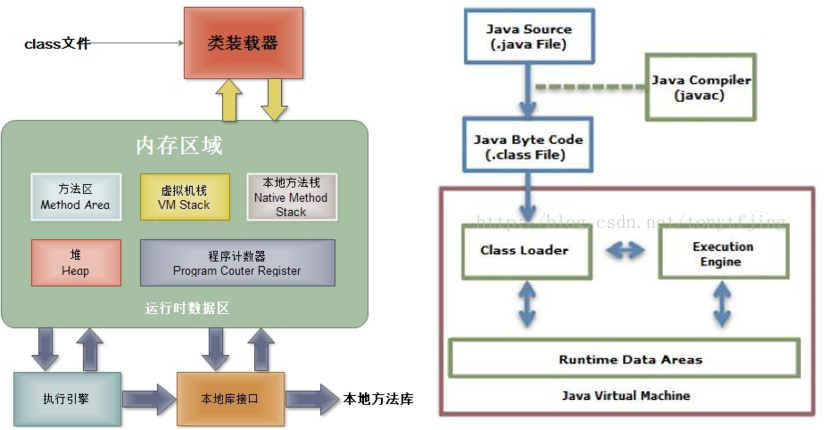
JVM结构学习

# JVM结构

根据《java虚拟机规范》规定，JVM的基本结构一般如下图所示：



# JVM的组成：

从左图可知，JVM主要包括四个部分：

## 类加载器（ClassLoader）:

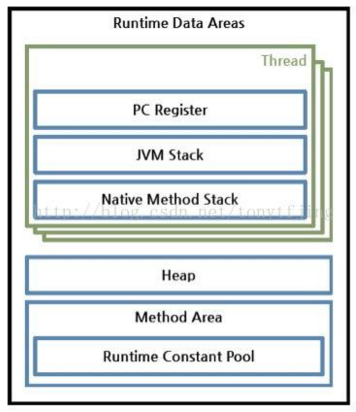
在JVM启动时或者在类运行时将需要的class加载到JVM中。（右图表示了从java源文件到JVM的整个过程，可配合理解。 关于类的加载机制，可以参考http://blog.csdn.net/tonytfjing/article/details/47212291）

## 执行引擎：Execution Engine

负责执行class文件中包含的字节码指令（执行引擎的工作机制，这里也不细说了，这里主要介绍JVM结构）；

## 内存区（也叫运行时数据区）：

是在JVM运行的时候操作**所分配的内存区**。运行时内存区主要可以划分为5个区域，如图：



### 方法区(Method Area)：

用于存储类结构信息的地方，包括常量池、静态变量、构造函数等。虽然JVM规范把方法区描述为堆的一个逻辑部分， 但它却有个别名non-heap（非堆），所以大家不要搞混淆了。方法区还包含一个运行时常量池。

### java堆(Heap)：

存储java实例或者对象的地方。这块是GC的主要区域（后面解释）。从存储的内容我们可以很容易知道，方法区和堆是被所有java线程共享的。

### java栈(Stack)：

java栈总是和线程关联在一起，每当创建一个线程时，JVM就会为这个线程创建一个对应的java栈。在这个java栈中又会包含多个栈帧，每运行一个方法就创建一个栈帧，用于存储局部变量表、操作栈、方法返回值等。每一个方法从调用直至执行完成的过程，就对应一个栈帧在java栈中入栈到出栈的过程。所以java栈是现成私有的。

### 程序计数器(PC Register)：

用于保存当前线程执行的内存地址。由于JVM程序是多线程执行的（线程轮流切换），所以为了保证线程切换回来后，还能恢复到原先状态，就需要一个独立的计数器，记录之前中断的地方，可见程序计数器也是线程私有的。

### 本地方法栈(Native Method Stack)：

和java栈的作用差不多，只不过是为**JVM使用到的native方法服务的**。

## 本地方法接口：

主要是调用C或C++实现的本地方法及返回结果。

# 内存分配

我觉得了解垃圾回收之前，得先了解**JVM是怎么分配内存**的，然后识别**哪些内存是垃圾需要回收**，最后才是**用什么方式回收**。

Java的内存分配原理与C/C++不同，**C/C++每次申请内存时都要malloc进行系统调用，而系统调用发生在内核空间，每次都要中断进行切换，这需要一定的开销，而Java虚拟机是先一次性分配一块较大的空间，然后每次new时都在该空间上进行分配和释放，减少了系统调用的次数，节省了一定的开销，这有点类似于内存池的概念**；二是**有了这块空间过后，如何进行分配和回收就跟GC机制有关了**。

java一般内存申请有两种：静态内存和动态内存。很容易理解，编译时就能够确定的内存就是静态内存，即内存是固定的，系统一次性分配，比如int类型变量；动态内存分配就是在程序执行时才知道要分配的存储空间大小，比如java对象的内存空间。根据上面我们知道，java栈、程序计数器、本地方法栈都是线程私有的，线程生就生，线程灭就灭，栈中的栈帧随着方法的结束也会撤销，内存自然就跟着回收了。所以这几个区域的内存分配与回收是确定的，我们不需要管的。但是java堆和方法区则不一样，我们只有在程序运行期间才知道会创建哪些对象，所以这部分内存的分配和回收都是动态的。一般我们所说的垃圾回收也是针对的这一部分。

总之Stack的内存管理是顺序分配的，而且定长，不存在内存回收问题；而Heap 则是为java对象的实例随机分配内存，不定长度，所以存在内存分配和回收的问题；

# GC：Garbage Collection垃圾回收

Garbage Collection