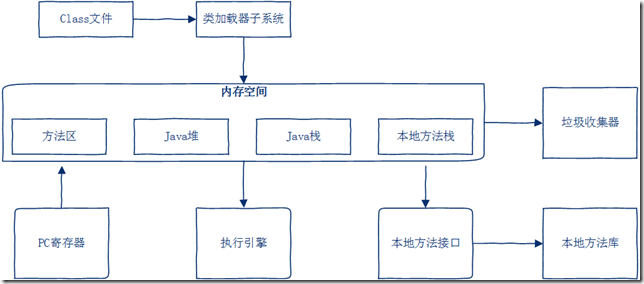
JVM的内部结构：



**方法区**是各个线程共享的区域，存放类信息、常量、静态变量。

**Java堆**也是线程共享的区域，我们的类的实例就放在这个区域，可以想象你的一个系统会产生很多实例，因此java堆的空间也是最大的。如果java堆空间不足了，程序会抛出OutOfMemoryError异常。

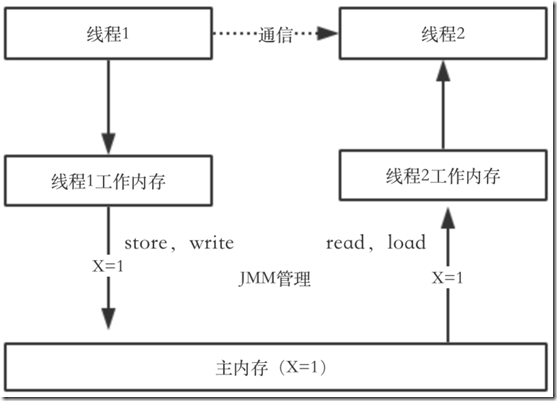
**Java栈**是每个线程私有的区域，它的生命周期与线程相同，一个线程对应一个java栈，每执行一个方法就会往栈中压入一个元素，这个元素叫“栈帧”，而栈帧中包括了方法中的局部变量、用于存放中间状态值的操作栈，这里面有很多细节，我们以后再讲。如果java栈空间不足了，程序会抛出StackOverflowError异常，想一想什么情况下会容易产生这个错误，对，递归，递归如果深度很深，就会执行大量的方法，方法越多java栈的占用空间越大。

**本地方法栈**角色和java栈类似，只不过它是用来表示执行本地方法的，本地方法栈存放的方法调用本地方法接口，最终调用本地方法库，实现与操作系统、硬件交互的目的。

**PC寄存器**，说到这里我们的类已经加载了，实例对象、方法、静态变量都去了自己改去的地方，那么问题来了，程序该怎么执行，哪个方法先执行，哪个方法后执行，这些指令执行的顺序就是PC寄存器在管，它的作用就是控制程序指令的执行顺序。

**执行引擎**当然就是根据PC寄存器调配的指令顺序，依次执行程序指令。

线程间的通信



当线程1需要将一个更新后的变量值刷新到主内存中时，需要经过两个步骤：

1、 工作内存执行store操作；

2、 主内存执行write操作；

完成这两步即可将工作内存中的变量值刷新到主内存，即线程1工作内存和主内存的变量值保持一致；

当线程2需要从主内存中读取变量的最新值时，同样需要经过两个步骤：

1、主内存执行read操作，将变量值从主内存中读取出来；

2、工作内存执行load操作，将读取出来的变量值更新到本地内存的副本；

完成这两步，线程2的变量和主内存的变量值就保持一致了。

关键字volatile

Volatile保证两件事：

1、 线程1工作内存中的变量更新会强制立即写入到主内存；

2、 线程2工作内存中的变量会强制立即失效，这使得线程2必须去主内存中获取最新的变量值。

指令重排序

IMG_256

在这个流程中第一步属于编译器重排查，编译器重排序会按JMM的规范严格进行，换言之编译器重排序一般不会对程序的正确逻辑造成影响。第二、三步属于处理器重排序，处理器重排序JMM就不好管了，怎么办呢？它会要求java编译器在生成指令时加入内存屏障，内存屏障是什么？你可以理解为一个不透风的保护罩，把不能重排序的java指令保护起来，那么处理器在遇到内存屏障保护的指令时就不会对它进行重排序了。