### volatile关键字

确保本条指令不会因编译器的优化而省略，且要求使用时每次都重新读取、写入这个变量的值而不是使用保存在寄存器里的备份。

（为了提高存取速度，编译器优化有时会先把变量读取到一个寄存器中，以后再取变量值时，就直接从寄存器中取值。而当变量在别的线程改变了值，该寄存器的值不会相应的改变）

Java语言规范中说：为了获得最佳速度，允许线程保存共享成员变量的私有拷贝，而且只当线程进入或离开同步代码块时才与共享成员变量的原始值对比。所以建议在两个或更多的线程访问的成员变量上使用volatile，当要访问的变量已在synchronized代码块中时或为常量时，不必使用。

常用在

1. 并行设备的硬件寄存器，如：状态寄存器
2. 一个中断服务子程序中会访问到的非自动变量
3. 多线程应用中被几个任务共享的变量

volatile具有synchronized的可见性特征，但是不具备原子特性。在某些情况下，如果读操作远远大于写操作，volatile变量还可以提供优于锁的性能优势。

注意使用条件：

1. 对变量的写操作不依赖于当前值，如x++
2. 该变量没有包含在具有其他变量的不变式中

### Session

会话控制、时域。

指一个终端用户与交互系统进行通信的时间间隔，通常指从注册进入系统到注销退出系统之间所经过的时间，如果需要的话，可能还有一定的操作空间。

工作原理：

1. 当一个session第一次被启动时，一个唯一的标志碑存储在本地的cookie中；
2. 首先使用session\_start()函数，PHP从session仓库中加载已经存储的session变量；
3. 当执行PHP脚本时，通过使用session\_register()函数注册session变量；
4. 当PHP脚本执行结束时，未被销毁的session变量会被自动保存在本地一定路径下的session库中，这个路径可以通过php.ini文件中的session.save\_path指定，下次浏览网页时可以加载使用。

### JIT Compiler

最早的Java建置方案是由interpreter（解释器），将每个Java指令都转译成对等的微处理器指令，并根据转译后的指令先后次序依序执行，由于一个Java指令可能被转译成十几个或数十个微处理器指令，这种模式的执行速度相当缓慢。

JIT编译器，当Java执行runtime环境时，每遇到一个新的类，JIT编译器就在此时对这个类进行编译，经过编译后的程序被优化成二进制，执行速度非常快，花费少许的编译时间来节省稍后相当长的执行时间。

Dynamic Compiler，动态编译器仅针对较常被执行的程序码进行编译，其余部分扔使用转译程序来执行。

### Native

Java的不足除了体现在运行速度上要比传统的C++慢许多之外，Java无法直接访问到操作系统底层（比如系统硬件），为此Java使用native方法来扩展Java程序的功能。

实现步骤：

1. 在Java中声明native()方法，然后编译；
2. 用javah产生一个.h文件；
3. 写一个.cpp文件实现native导出方法，其中需要包含第二部产生的.h文件（注意其中又包含了JDK代的jni.h文件）；
4. 将第三步的.cpp文件编译成动态链接库文件；
5. 在Java中用System.loadLibrary()方法加载第四步产生的动态链接库文件，这个native()方法就可以在Java中被访问了。

适用情况：

1. 为了使用底层的主机平台的某个特性，而这个特性不能通过Java API访问
2. 为了访问一个老的系统或者使用一个已有的库，而这个系统或这个库不是用Java编写的；
3. 为了加快程序的性能，而将一段时间敏感的代码作为本地方法实现。