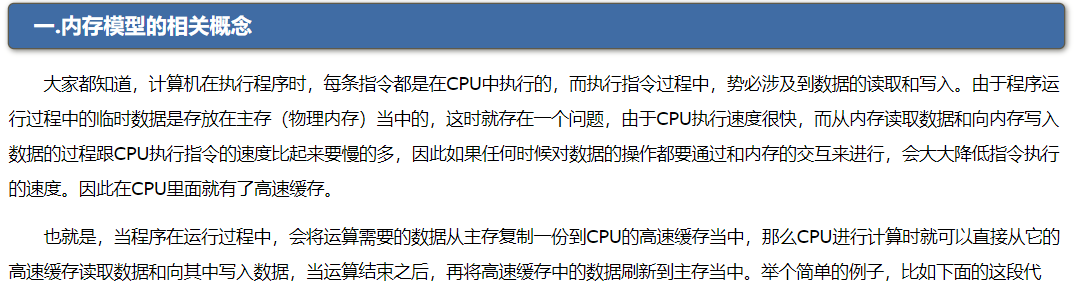
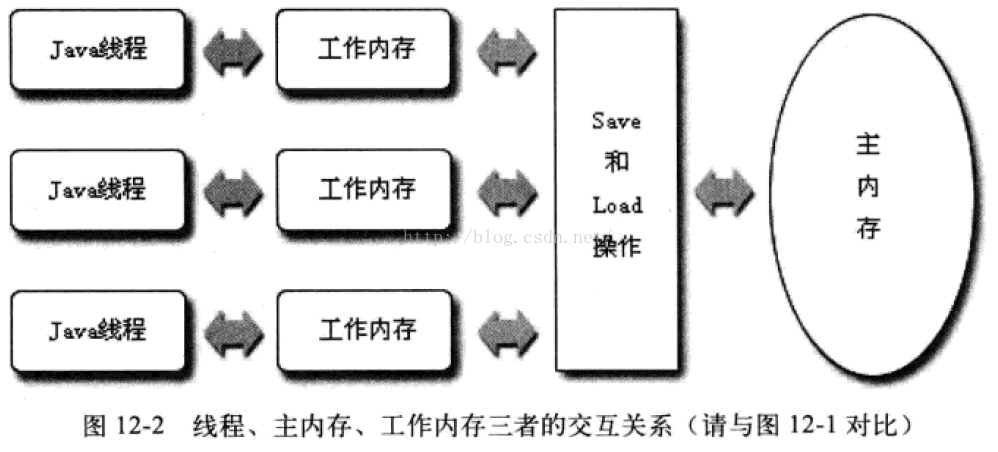
线程内存模型



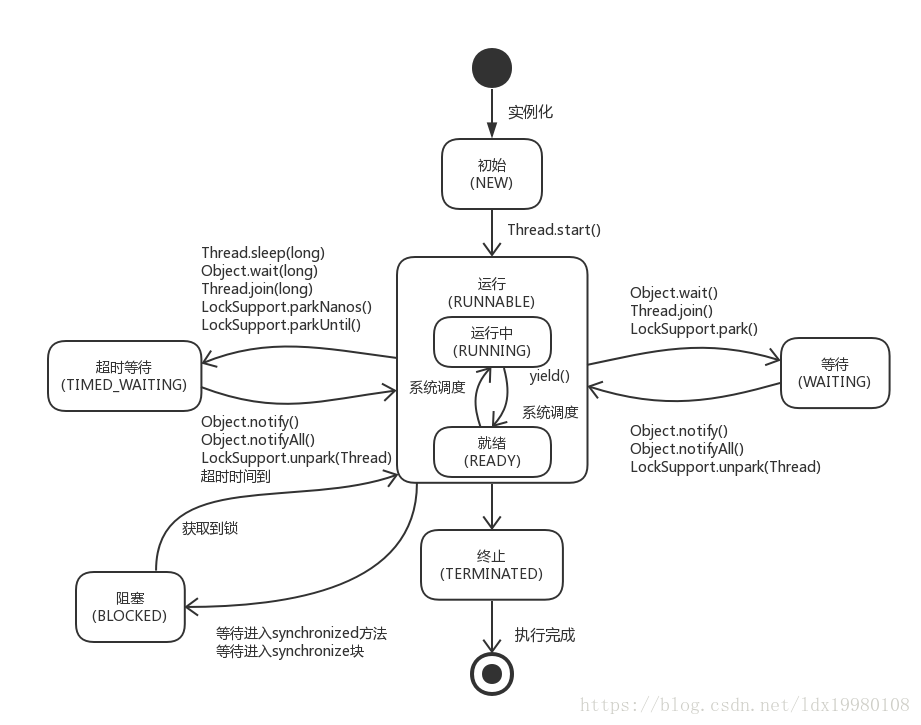


Java所有的变量都存在主内存中，此处的主内存仅仅是虚拟机内存的一部分，而虚拟机内存也仅仅是计算机物理内存的一部分（为虚拟机进程分配的那一部分）。

Java内存模型分为主内存，和工作内存。主内存是所有的线程所共享的，工作内存是每个线程自己有一个，不是共享的

* 每条线程还有自己的工作内存，线程的工作内存中保存了此线程使用到的数据,数据是主内存的拷贝
* 线程对变量的所有操作（读取、赋值），都必须在工作内存中进行，而不能直接读写主内存中的变量，操作结束再将工作内存的数据刷新到主存
* 不同线程之间也无法直接访问对方工作内存中的变量，线程间变量值的传递均需要通过主内存来完成

线程6种状态



**初始态:**

* + **创建一个线程对象但未使用start()方法，该线程对象处于初始态**

**运行态：RUNNABLE**

在Java中，运行态包括就绪态 和 运行态

**就绪态**

* **该状态下的线程已经获得执行所需的所有资源，只要CPU分配执行权就能运行**
* **所有就绪态的线程存放在就绪队列中。**

**运行态**

* **获得CPU执行权，正在执行的线程**
* **由于一个CPU同一时刻只能执行一条线程，因此每个CPU每个时刻只有一条运行态的线程**

**阻塞状态:**

当一条正在执行的线程请求某一资源失败时，就会进入阻塞态

PS：锁、IO、Socket等都资源

* 在Java中，阻塞态专指请求锁失败时进入的状态
* 由一个阻塞队列存放所有阻塞态的线程
* 处于阻塞态的线程会不断请求资源，一旦请求成功，就会进入就绪队列，等待执行
* **等待态**
* 当前线程（已获得锁）中调用wait（）、join（）函数时，当前线程就会进入等待态, 进入等待态的线程会释放资源（如：锁）
* 等待队列存放所有等待态的线程
* 线程处于等待态表示它需要等待其他线程的唤醒才能继续运行
* **超时等待态**
* 当前线程（已获得锁）中调用带时间参数的wait(long milli)方法
* 与等待态的区别：到了超时时间后自动进入阻塞队列，开始竞争锁
* **死亡状态:**
* **线程执行结束后的状态**

**线程的优先级**

**每一个 Java 线程都有一个优先级，这样有助于操作系统确定线程的调度顺序**

**Java 线程的优先级是一个整数**

**其取值范围是 1 （Thread.MIN\_PRIORITY ） - 10 （Thread.MAX\_PRIORITY ）。**

**默认情况下，每一个线程都会分配一个优先级 NORM\_PRIORITY（5）。**

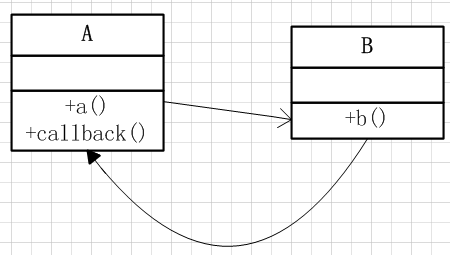
具有较高优先级的线程对程序更重要，并且应该在低优先级的线程之前分配处理器资源。但是，线程优先级不能保证线程执行的顺序，依赖于平台。

线程的优先级有继承关系，比如A线程中创建了B线程，那么B将和A具有相同的优先级

**启动一个线程是调用run()还是start()方法？**

启动一个线程是调用start()方法，使线程处于可运行状态，它可以被JVM 调度并执行，但并不是会立即运行。run()方法是线程启动后要进行回调（callback）的方法

run（）方法是多线程程序的一个约定。所有的多线程代码都在run方法里面



* 类A的a()方法调用类B的b()方法
* 类B的b()方法执行完毕主动调用类A的callback()方法

**Main**

**程序启动运行main时候，java虚拟机启动一个进程，主线程main在main()调用时候被创建**

**在java中，每次程序运行至少启动2个线程。一个是main线程，一个是垃圾收集线程**

(用户线程(系统的工作线程)和守护线程的区别在于，是否依赖于主线程结束而结束)

当一个Java应用内只有守护线程时，虚拟机会主动退出

**多线程特点**

1. **多线程程序是乱序执行。因此，只有乱序执行的代码才有必要设计为多线程**
2. **java中所有的线程都是同时启动的，哪个先执行，完全看谁先得到CPU的资源**
3. **因为java编译器在编译成字节码的时候，会对代码进行一个重排序，也就是说，编译器会根据实际情况对代码进行一个合理的排序，编译前代码写在前面，在编译后的字节码不一定排在前面** test1写在test2前面，test2却先运行

**interrupt():不是中断某个线程！只是改变中断状态(false->true)，不会终止一个正在运行的线程，中断一个线程只是为了引起该线程的注意**

* **如果线程被Object.wait, Thread.join和Thread.sleep三种方法之一阻塞，此时调用该线程的interrupt()方法，该线程将抛出一个 InterruptedException中断异常，然后线程的中断标志位会由true重置为false，线程为了处理异常从而提早退出被阻塞状态，重新处于就绪状态**
* **如果线程没有被阻塞，这时调用 interrupt()将不起作用**