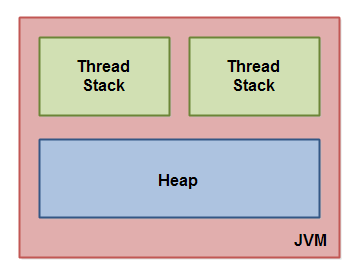
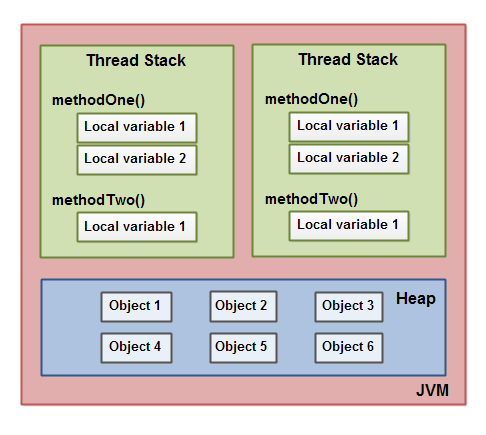
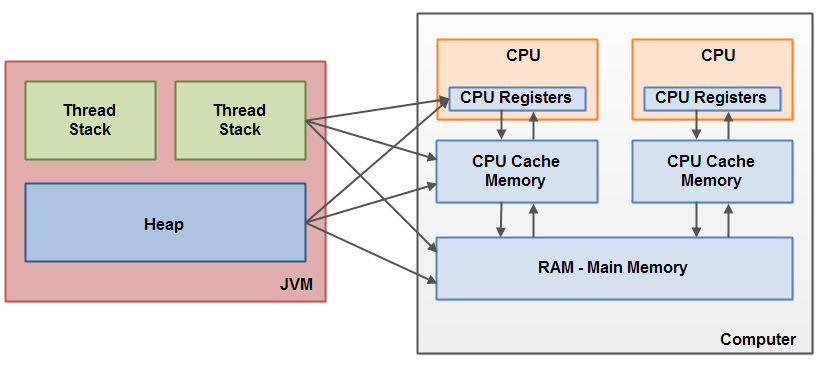
内存模型：





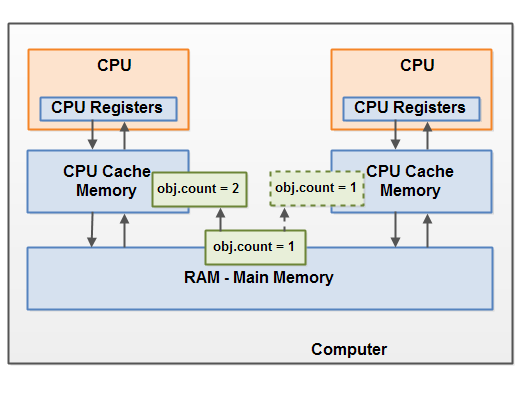


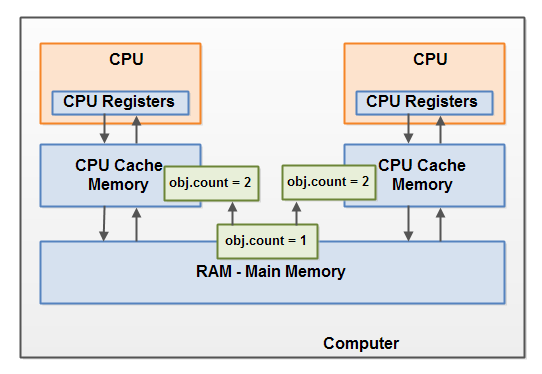




两个问题：1）线程更新共享变量的可见域

2）读写共享变量的竞争条件





1. synchronized
2. public **synchronized** void add(int value){
3. this.count += value;
4. }

public **static synchronized** void add(int value){

count += value;

}

public class MyClass {

public **synchronized** void log1(String msg1, String msg2){

log.writeln(msg1);

log.writeln(msg2);

}

public void log2(String msg1, String msg2){

**synchronized(this){**

log.writeln(msg1);

log.writeln(msg2);

**}**

}

}

public class MyClass {

public static synchronized void log1(String msg1, String msg2){

log.writeln(msg1);

log.writeln(msg2);

}

public static void log2(String msg1, String msg2){

synchronized(MyClass.class){

log.writeln(msg1);

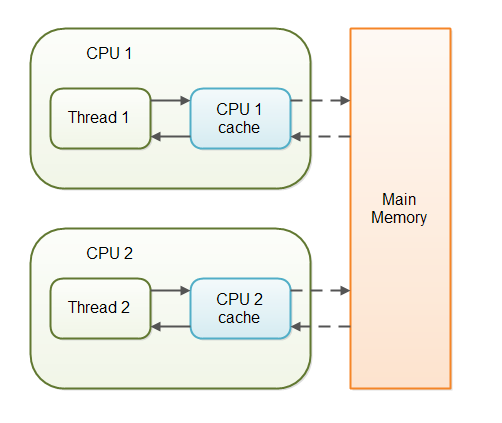
log.writeln(msg2);

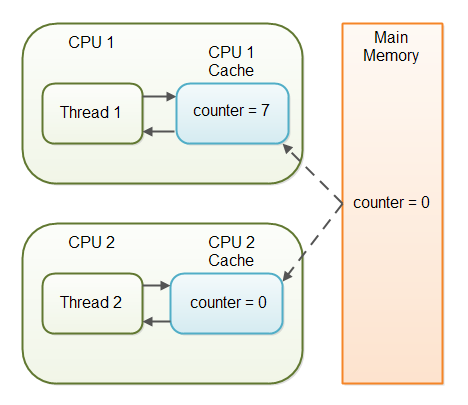
}

}

}

2.volatile 轻量级的synchronized：保证了共享变量的可见性。Volatile变量修饰符如果使用**恰当**的话，它比synchronized的**使用和执行成本会更低**，因为它不会引起线程上下文的切换和调度。





Happens-before

4.ThreadLocal

代码

5.lock

响应中断，可以设置时长，可以在单独的方法中分别调用lock()和unlock()

Lock lock = new ReentrantLock();

lock.lock();

//critical section

lock.unlock();

Semaphore semaphore = new Semaphore(1);

//critical section

semaphore.acquire();

...

semaphore.release();

CountDownLatch CyclicBarrier

6. executor

ExecutorService executorService = Executors.newFixedThreadPool(10);

executorService.execute(new Runnable() {

public void run() {

System.out.println("Asynchronous task");

}

});

executorService.shutdown();