

同步和异步

多线程并发时，多个线程同时请求同一个资源，必然导致此资源的数据不安全，A线程修改了B线程的处理的数据，而B线程又修改了A线程处理的数理。显然这是由于全局资源造成的，有时为了解决此问题，优先考虑使用局部变量，退而求其次使用同步代码块，出于这样的安全考虑就必须牺牲系统处理性能，加在多线程并发时资源挣夺最激烈的地方，这就实现了线程的同步机制

同步：A线程要请求某个资源，但是此资源正在被B线程使用中，因为同步机制存在，A线程请求不到，怎么办，A线程只能等待下去

异步：A线程要请求某个资源，但是此资源正在被B线程使用中，因为没有同步机制存在，A线程仍然请求的到，A线程无需等待

显然，同步最最安全，最保险的。而异步不安全，容易导致死锁，这样一个线程死掉就会导致整个进程崩溃，但没有同步机制的存在，性能会有所提升

1、 什么时候必须同步？什么叫同步？如何同步？

要跨线程维护正确的可见性，只要在几个线程之间共享非 final 变量，就必须使用synchronized（或 volatile）以确保一个线程可以看见另一个线程做的更改。为了在线程之间进行可靠的通信，也为了互斥访问，同步是必须的。这归因于java语言规范的内存模型，它规定了：一个线程所做的变化何时以及如何变成对其它线程可见。

因为多线程将异步行为引进程序，所以在需要同步时，必须有一种方法强制进行。

例如：如果2个线程想要通信并且要共享一个复杂的数据结构，如链表，此时需要确保它们互不冲突，也就是必须阻止B线程在A线程读数据的过程中向链表里面写数据（A获得了锁，B必须等A释放了该锁）。

为了达到这个目的，java在一个旧的的进程同步模型——监控器（Monitor）的基础上实现了一个巧妙的方案：监控器是一个控制机制，可以认为是一个很小的、只能容纳一个线程的盒子，一旦一个线程进入监控器，其它的线程必须等待，直到那个线程退出监控为止。通过这种方式，一个监控器可以保证共享资源在同一时刻只可被一个线程使用。这种方式称之为同步。（一旦一个线程进入一个实例的任何同步方法，别的线程将不能进入该同一实例的其它同步方法，但是该实例的非同步方法仍然能够被调用）。

错误的理解：同步嘛，就是几个线程可以同时进行访问。

同步和多线程关系：没多线程环境就不需要同步;有多线程环境也不一定需要同步。锁提供了两种主要特性：互斥（mutual exclusion）和可见性（visibility）。互斥即一次只允许一个线程持有某个特定的锁，因此可使用该特性实现对共享数据的协调访问协议，这样，一次就只有一个线程能够使用该共享数据。

可见性要更加复杂一些，它必须确保释放锁之前对共享数据做出的更改对于随后获得该锁的另一个线程是可见的 —— 如果没有同步机制提供的这种可见性保证，线程看到的共享变量可能是修改前的值或不一致的值，这将引发许多严重问题

小结：为了防止多个线程并发对同一数据的修改，所以需要同步，否则会造成数据不一致（就是所谓的：线程安全。如java集合框架中Hashtable和Vector是线程安全的。我们的大部分程序都不是线程安全的，因为没有进行同步，而且我们没有必要，因为大部分情况根本没有多线程环境）。

怎样理解阻塞非阻塞与同步异步的区别？

“阻塞”与"非阻塞"与"同步"与“异步"不能简单的从字面理解，提供一个从分布式系统角度的回答。  
1.同步与异步  
同步和异步关注的是消息通信机制 (synchronous communication/ asynchronous communication)  
所谓同步，就是在发出一个\*调用\*时，在没有得到结果之前，该\*调用\*就不返回。但是一旦调用返回，就得到返回值了。  
换句话说，就是由\*调用者\*主动等待这个\*调用\*的结果。

而异步则是相反，\*调用\*在发出之后，这个调用就直接返回了，所以没有返回结果。换句话说，当一个异步过程调用发出后，调用者不会立刻得到结果。而是在\*调用\*发出后，\*被调用者\*通过状态、通知来通知调用者，或通过回调函数处理这个调用。

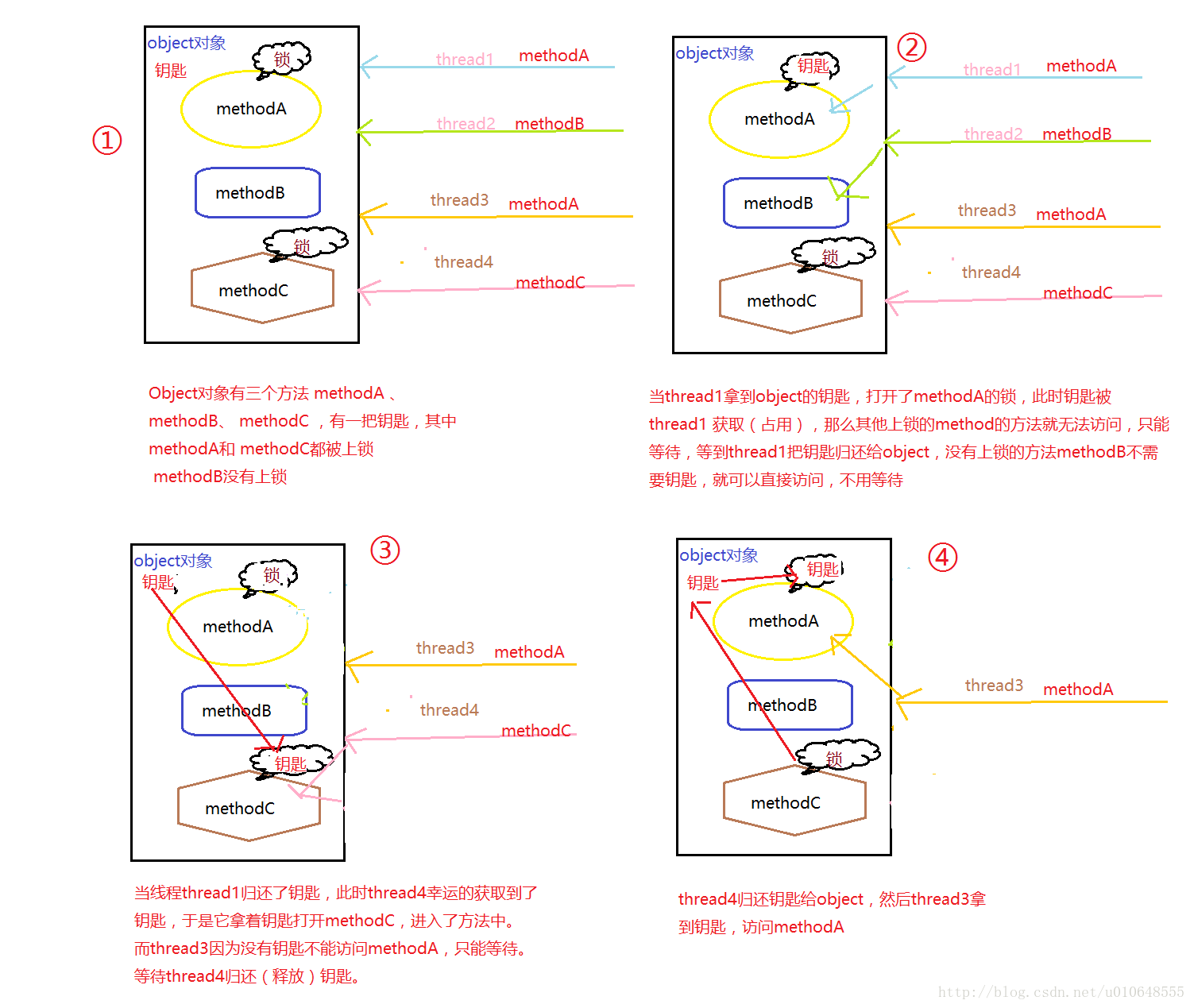
典型的异步编程模型比如Node.js

举个通俗的例子：  
你打电话问书店老板有没有《分布式系统》这本书，如果是同步通信机制，书店老板会说，你稍等，”我查一下"，然后开始查啊查，等查好了（可能是5秒，也可能是一天）告诉你结果（返回结果）。  
而异步通信机制，书店老板直接告诉你我查一下啊，查好了打电话给你，然后直接挂电话了（不返回结果）。然后查好了，他会主动打电话给你。在这里老板通过“回电”这种方式来回调。

2. 阻塞与非阻塞  
阻塞和非阻塞关注的是程序在等待调用结果（消息，返回值）时的状态.

阻塞调用是指调用结果返回之前，当前线程会被挂起。调用线程只有在得到结果之后才会返回。  
非阻塞调用指在不能立刻得到结果之前，该调用不会阻塞当前线程。

还是上面的例子，  
你打电话问书店老板有没有《分布式系统》这本书，你如果是阻塞式调用，你会一直把自己“挂起”，直到得到这本书有没有的结果，如果是非阻塞式调用，你不管老板有没有告诉你，你自己先一边去玩了， 当然你也要偶尔过几分钟check一下老板有没有返回结果。  
在这里阻塞与非阻塞与是否同步异步无关。跟老板通过什么方式回答你结果无关。



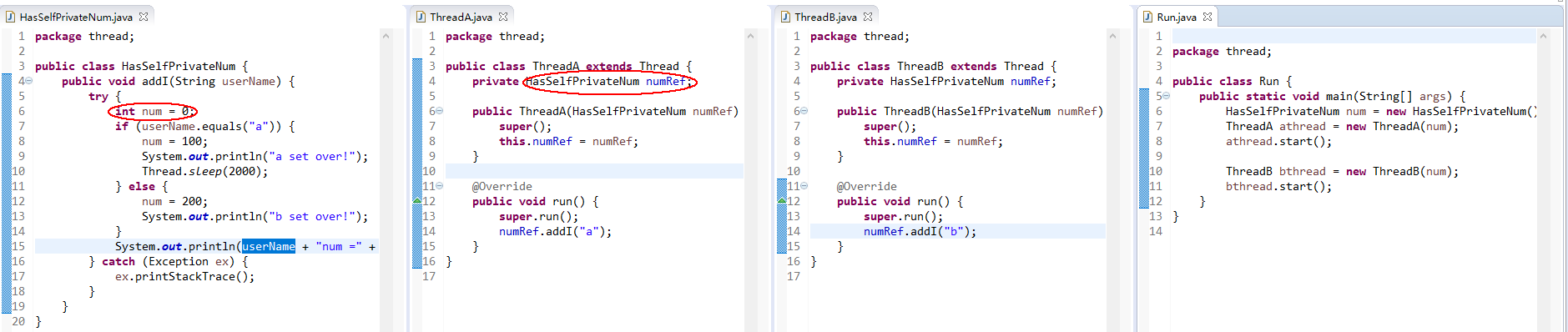
第二章:对象及变量的并发访问

1>”线程安全”与”非线程安全”?

答:”非线程安全”其实会在多个线程对同一个对象中的实例变量进行并发访问时发生,产生的后果就是”脏读”,也就是取到的数据其实是被更改过的.

而”线程安全”就是以获得的实例变量的值是经过同步处理的,不会出现脏读的现象.

2.1.1方法内的变量为线程安全

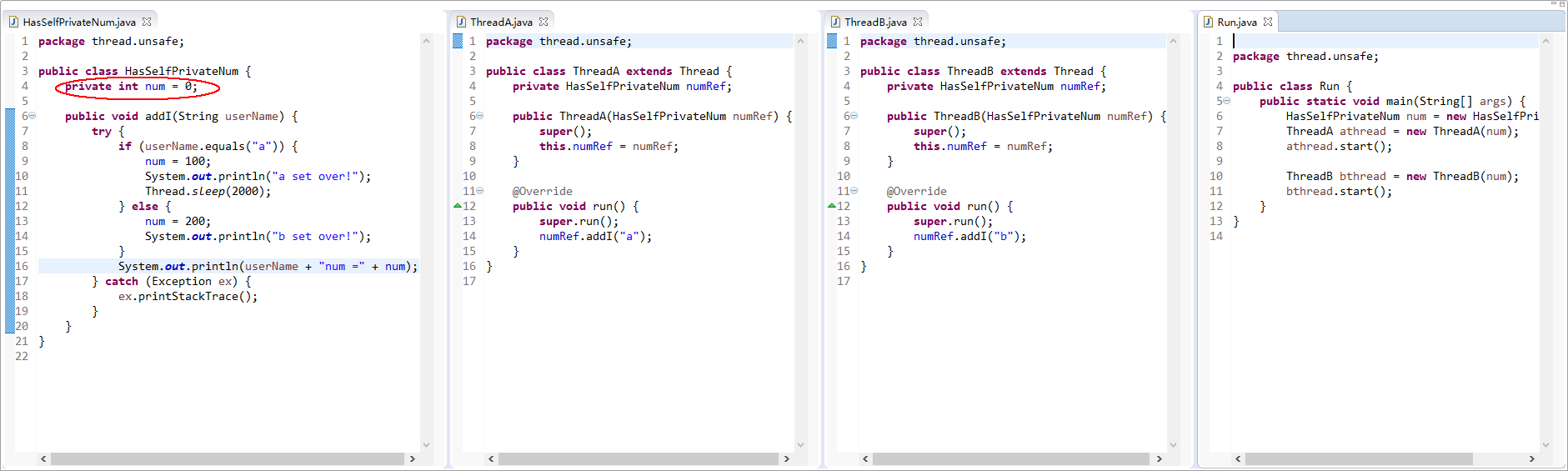




Num为方法内私有变量!

2.1.2实例变量非线程安全

如果多个线程共同访问1个对象中的实例变量,则有可能出现”非线程安全”问题.





Num为共享变量.

如何解决?

在方法addI前加个synchronized即可.

实验结论:在两个线程访问同一个对象中的同步方法时一定是线程安全的.

\*关键字synchronized取得的锁都是对象锁,而不是把一段代码或方法(函数)当做锁.