委托：将方法作为参数传递，避免大量的if-else判断。

事件：对委托进行了封装，防止调用委托方随意对委托进行修改。

Observer设计模式：（为了定义对象间一对多的依赖关系，以便于当监视对象状态改变时，

监视者被自动告知并更新）

Subject：监视对象，包含着其他对象感兴趣的内容；

Observer：监视者，监视Subject

事件发生顺序：注册-》状态改变-》自动调用

反射：可访问一个正在运行的程序的元数据。

异步编程：

代码不需要按照编写时的顺序执行，异步调用某方法时，该方法会立即返回，

然后再执行方法内部的语句。

11-30

如果程序定义了一个不会完成的前台线程，主程序并不会正常结束。(读注:之前调用组卷，异常后系统无法退出，不知道是否是这个原因)

lock：如果锁定了一个对象，则需要访问该对象的所有其它线程会处于阻塞状态。是Monitor类的语法糖。

12-3

线程内的代码使用try catch捕获异常。

12-6

线程池的用途是执行运行时间短的操作。原因是使用了一定数量的可用工作线程，如果执行长时间计算密集型的操作，则会造成线程不够用的情况，降低性能。

线程池的线程都是后台线程，所以当前台线程(包括主程序线程)完成后，所有后台线程会停止操作。

12-7

使用线程池，为操作系统节省了内存和线程数，但是付出了更长的执行时间。

12-16

异步编程模型(APM Asynchronous Programming Model)

基于事件的异步模式 (EAP Event-based Asynchronous Pattern)

基于任务的异步编程，任务并行库（TPL），(TAP Task-based Asynchronous Pattern)

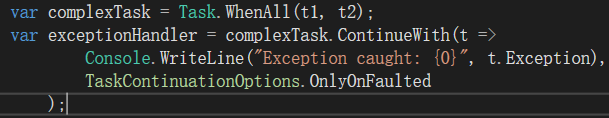
当在调用task.Result时，调用线程会被阻塞，直到返回结果，只要不调用.Result，则不会阻塞

调用.Result会将异常传给当前线程(调用线程)

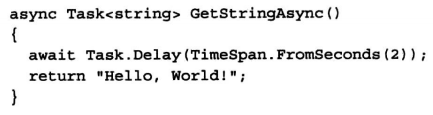
.Result //会封装异常为AggregateException

.GetAwaiter().GetResult() //不会封装异常，因为TPL基础设施会提取异常

//多个任务抛异常的情况，会封装异常为AggregateException



async await



在执行完await调用的代码行后，该方法会立即返回。如果是同步执行，执行线程会阻塞两秒然后返回结果。这里执行完await操作后，立即将工作线程放回线程池的过程中，我们会异步等待。2秒后，我们又一次从线程池中得到工作线程并继续运行其中剩余的异步方法。这允许我们在等待2秒时重用工作线程做些其他事。

任何lambda表达式的类型都不能通过lambda自身来推断，所以需要显示向C#编译器指定它的类型。

**Thread.Sleep vs. Task.Delay**

Thread.Sleep会阻塞当前调用的线程，如在UI线程中调用，则在时间耗尽之前，无法对其abort，UI线程无法响应任何操作；Task.Delay创建一个task，内部启动一个定时器(因此可以取消)，不会阻塞调用线程。

详情：

<https://social.technet.microsoft.com/wiki/contents/articles/21177.visual-c-thread-sleep-vs-task-delay.aspx>

12-23

await关键字 或者task.Wait()，并不是说代码执行到这句才执行，在新建任务后，任务就开始执行了，只是在到await或wait时，必须等待任务完成。

12-24

数据并行：复杂运算，例如对数据块进行并行运算，然后聚合计算结果。

结构并行：较简单的运算，程序员可以不关心如何实现并行，将细节交给PFX库。

Parallel静态类：

Invoke方法：会阻塞其他线程，直到所有的任务都被完成。

For和ForEach方法

**02-28 .net垃圾回收**

CLR垃圾回收器根据所占空间大小划分对象，阈值为85000byte。

小对象：.net1.0和.net2.0中，大小 < 85000byte。

大对象：.net1.0和.net2.0中，大小 >= 85000byte。

大对象，会被分配到大对象堆(LOH)中，小对象会被分配到小对象堆(SOH)。

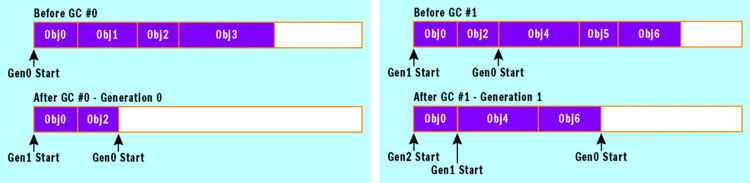
.net GC按照“代”回收，程序中的对象分为0代、1代、2代，0代最年轻，2代存活时间最长。GC按代回收垃圾也是处于性能考虑，通常的对象会在0代时被回收。

从代的角度看，大对象属于2代对象，因为只有在2代回收时才会处理大对象。当某代垃圾回收执行时，会同时执行更年轻代的垃圾回收。

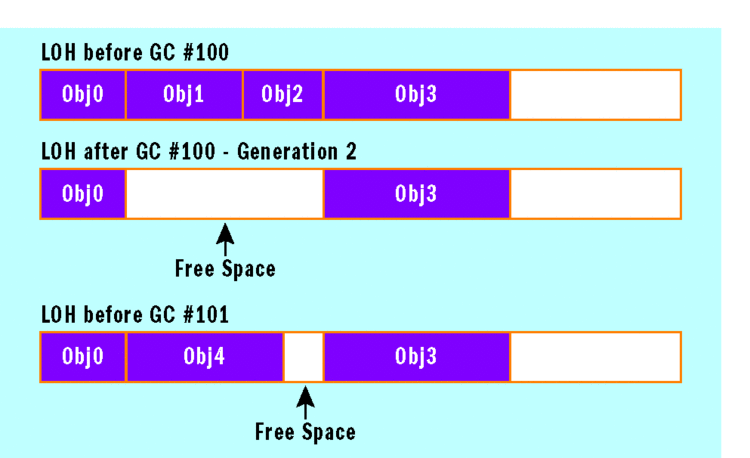
代是垃圾回收器区分内存区域的逻辑视图。从物理存储角度看，对象分配在不同的托管堆上。一个托管堆(managed heap)是垃圾回收器从操作系统申请的内存区。当CLR载入内存后，会初始化两个托管堆，大对象堆(large object heap)和小对象堆。

对于SOH，对象在执行一次垃圾回收后，存活下来的对象会进入到下一代，一直到最高代-2代。

当垃圾回收时，垃圾回收器会在小对象堆做碎片整理；大对象堆出于性能考虑，不做整理，而是当下一次申请大对象时，如果大小合适的话，会放入之前清理的碎片内。(直到.net 4.0，4.0之后不清楚)



小对象垃圾回收



大对象垃圾回收

如果大对象堆上没有足够的空闲容纳要申请的大对象空间，CLR首先向操作系统申请内存，如果申请失败，则触发一次二代回收来释放内存。

**什么时候回收大对象？**

* 垃圾回收发生情况：

1. 申请的空间超过0代内存大小或者大对象堆的阈值，多数的托管堆垃圾回收在这种情况下发生；

2. 在程序代码中调用GC.Collect方法时；如果在调用GC.Collect方法时传入GC.MaxGeneration参数时，会执行所有代对象的垃圾回收，包括大对象堆的垃圾回收；

3. 操作系统内存不足时，当应用程序收到操作系统发出的高内存通知时；

4. 如果垃圾回收算法认为做二代回收是有收效时会触发二代垃圾回收；

5. 每一代对象堆都有一个所占空间大小阈值的属性，当你分配对象到某一代，你增长了内存总量接近了该代的阈值，或者分配对象导致这一代的堆大小超过了堆阈值，就会发生一次垃圾回收。因此当你分配小对象或者大对象时，会对应消耗0代堆或者大对象堆的阈值。当垃圾回收器将对象代数提升到1代或者2代时，会消耗1、2代的阈值。在程序运行中这些阈值是动态变化的。

* 大对象堆性能影响

分配大对象的代价

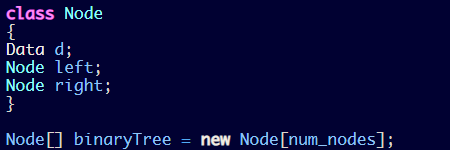
：CLR为每个新对象分配内存时，都要保证这些清空的内存是没有被其他对象使用的，这就意味着分配的代价完全被清理的代价控制着。如果清空1byte需要2个周期（cycles），就意味着清除一个最小的大对象需要170,000个周期。

回收大对象的代价

大对象和2代对象一起回收。如果大对象堆因为超过阈值回收，而此时2代堆很大，对象很多，过多的2代回收就会导致性能问题。如果是临时性分配的大对象，就需要很多的时间来运行垃圾回收；也就是说如果你持续的使用大对象然后又释放大对象对性能会有很大的负面影响。

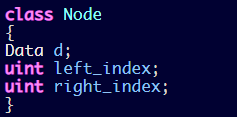
举例说明：

大对象堆上的巨大对象通常是数组(很少有一个对象很大的情况)。如果对象中的元素是强引用，垃圾回收的代价会很高。如下声明了Node类型的数组，每个节点都需要查看2个引用元素



强引用节点类型

另一种方法是在节点中保存左右节点元素的数组索引号，这样的话，元素间的引用关系就去掉了。垃圾回收器在回收时就不需要查看相关的引用元素了。



无引用节点类型