1. **如何把一个数组复制到ArrayList里？反过来从ArrayList到数组呢？**

循环

ArrayList的方法

1. **string str=null和string str=””有什么区别**

从内存分配上解释

1. **StringBuilder有什么作用**

在与非托管代码交互时也有作用。

从string类型和不可变性开始解释。

1. **泛型有什么好处？**

减少代码冗余，可以从具体的例子如List<T>类型解释，List<int>、List<string>等都有Add、Remove、Sort等方法，而且方法逻辑都是一样的，如果都要一个一个实现，那么会有很多方法都是冗余的，而泛型正好可以解决这个问题。

1. **拥有一个字符串数组{“2”,”3”,”4”},如何转换为整数数组。**

这个题目考的是数组转换问题，解决方法有

循环

Array静态方法ConverAll，顺便说一下效率问题。

1. **什么是序列化，.net中是如何实现序列化的？**

序列化概念：把数据转换成特定的格式，如xml、json等格式。

.net中序列化的实现有几个类型

XmlSerialize、JsonFormmater、SoapFormmater、BinaryFormmater

1. **什么是编码?什么是Unicode**
2. **Base64编码主要应用在什么场合**

公钥、证书、邮件附件、网站图片。

1. **字符串池是如何提高系统性能的。**

从字符串的不可变性开始解释

字符串的不可变性导致字符串是线程安全的，所以即便是几个变量指向同一个字符串，也不会出现线程安全问题。字符串池之所以能提高系统性能，是因为线程池的机制，导致在声明相同的字符串变量时，不用额外分配内存，只需要指向字符串池中已有的字符串。

1. **加密字符串类型和普通字符串类型有什么区别？**

加密字符串类型内存是在非托管堆中分配的，不过加密字符串中实现了Dispose和Finalize机制，所以能被垃圾回收器回收。而不同字符串是在托管堆中分配的内存。

1. **.NET设计小组为什么会设计System.Object这个类型。**

这个是CLS的规定，System.Object是所有类型的根，没有System.Object根的类型不是安全类型，不被公共语言系统兼容，不能跟.net中其他语言开发的模块实现良好互操作。

1. **System.Object类型有基类吗**

没有

1. **同一进程下的线程可以共享以下？(BD)**

A. Stack 栈

B. data section 数据段

C. register set 寄存器组

D. file fd 文件描述符

线程的栈和寄存器集都是私有的，线程在切换时会保存一个线程控制块的结构，里面包含了线程的栈和寄存器集等信息，用来下个时间片运行时恢复数据。

1. **后台线程和一般线程有什么区别？**

new Thread方式创建的线程默认都是前台线程，当所有前台线程都终止的时候，进程才会终止，线程终止时，所有后台线程都会被终止。

1. Lock、Monitor和Mutex的区别

这三个作用是一样的，可以实现线程同步

Lock是使用Monitor来实现的。

Monitor可以使用TryEnter来加锁，这样可以防止线程一直等待，所以可以阻止死锁的发生。而Lock内办法阻止死锁。

Mutex是内核级别的对象，消耗较大的资源（这点有点疑问，需要找资料），由于是内核对象，所以可以跨进程操作，也就是实现跨进程的同步。比如果防止程序打开两个实例。

1. 其他线程同步方式
   1. SpinLock 自旋锁。
   2. AutoResetEvent也可以实现类似锁的功能。
   3. 信号量Semaphore,当信号量阈值为1时，可以作为锁使用。当阈值大于1时，往往用来做流量控制。
2. 多线程安全问题
   1. 锁 lock、monitor、mutex、spinlock、semaphore(阈值为1时)
   2. Cas(compare and swap)比较和交换，先取值，对值进行操作后，在存储的时候进行比较交换，如果内存中的值与取出来的值不一样，说明已经被其他线程变更过了，再自旋重复操作。Cas是一个cpu指令，操作是原子性的，也就是在比较并判断是否交换的过程是一步实现的。.net中的InterLock类中有cas操作。
   3. 原子操作。InterLock类操作都是原子操作，包含了自增和cas等操作。
   4. 线程安全类。这些类型是在System.Collection.Concurrent命名空间下。

BlockingCollection<T>

ConcurrentBag<T>

ConcurrentDictionary<Tkey,Tvalue>

ConcurrentQueue<T>

ConcurrentStack<T>

ConcurrentQueue<T>和ConcurrentStack<T>是使用cas+自旋实现的,.net中就是使用Interlocked.CompareExchange+ SpinWait

ConcurrentDictionary<Tkey,Tvalue>有所不同，是使用锁(Monitor)来实现的，不过他会把hash值的范围分块，当插入某个值时，只是锁定该值所在的块，其他值不影响，这也就是说，当几个值同时插入时，如果hash值是在不同的块中，几个值的插入操作不会阻塞。

ConcurrentBag<T>好像是维护有两个集合。（待研究）

1. 什么时候考虑使用多线程？
   1. 避免主线程阻塞。典型的例子就是窗体程序。当某个操作耗时比较长时，不使用多线程，会造成ui线程长期阻塞，给用户带来不好的体验。
   2. 繁琐耗时的任务。如果任务可以拆分时，可以通过多线程来实现高效处理，当然还要看系统cpu核，单核的话反而会影响效率。
   3. IO等待(网络、磁盘、外部设备等IO输入输出)，可以使用多线程和异步，这样可以利用等待时间去完成其他事。
2. 线程间通讯
   1. AsynchronizationContext
   2. ManualResetEvent、ManualResetEventSlim 和AutoResetEvent
   3. Barrier
   4. Semaphore、SemaphoreSlim
3. Session的存储方式
   1. 服务进程中
   2. Sql server
   3. Redis等缓存数据库中。
4. 常见的http code（200、403、404、500）
5. 什么是多层架构，什么是mvc
6. Cookie和session的实现原理。
7. Sql注入
8. 后台跳转和前台跳转