C#.NET 面试题中高级篇

一、多线程

1、描述线程与进程的区别?

2、using 关键字有什么用？跟 IDisposable 有啥关系?

3、前台线程和后台线程有什么区别?

4、什么是线程互斥?

5、如何查看和设置线程池的上下限?

6、Task 状态机的实现和工作机制是什么?

7、await 的作用和原理，并说明和 GetResult()有什么区别?

8、多线程有什么用?

9、Task 和 Thread 有区别吗?

10、为什么 GUI 不支持跨线程调用?有什么解决方法?

11.说说常用的锁，lock 是一种什么样的锁?

12.lock 为什么要锁定一个参数（可否为值类型？）参数有什么要求?

13.多线程和异步的区别和联系?

14.线程池的优点有哪些？又有哪些不足

15.Mutex 和 lock 有什么不同？一般用哪一种比较好?

16.Thread 类有哪些常用的属性和方法?

17.聊聊任务 Task 与并行 Parallel

18.下面代码输出结果是什么？为什么?

19.多线程并行（Parallelism）和并发（Concurrency）的区别

20.C# Parallel.For 和普通 For 的区别

21 请简述 async 函数的编译方式?

22.请简述 Task 状态机的实现和工作机制?

二、Linq 和 EF 面试题

1.EF(Entity Framework)是什么？

2.什么是 **ORM?**

3.为什么用 EF 而不用原生的 ADO.NET?

4.如何提高 LINQ 性能问题？

5.什么是 IEnumerable？

6.IEnumerable 的缺点有哪些？

7.延迟执行 (Lazy Loading)是什么？

8.LINQ 可视化工具简单介绍一下？

9.LINQ to Object 和 LINQ to SQL 有何区别？

10.除了 EF，列举出你知道的 ORM 框架？

11.在哪些类型额项目中你会选择 EF? 为什么？

12.请说明 EF 中映射实体对象的几种状态？

三、泛型、数组、数据结构等

1. IList 接口与 List 的区别是什么?

2.泛型的主要约束和次要约束是什么?

3. 如何把一个 array 复制到 arraylist 里?

4.数组和 list 和 arraylist 的区别?

5. Set 里的元素是不能重复的，那么用什么方法来区分重复与否呢？是用==还是 equals()？它们有何区别?

6.有 50 万个 int 类型的数字，现在需要判断一下里面是否存在重复的数字，请你简要说一下思路。

7.数组有没有 length()这个方法? String 有没有 length()这个方法?

8.一个整数 List 中取出最大数（找最大值）。不能用 Max 方法。

9.利用 IEnumerable 实现斐波那契数列生成?

10.请利用 foreach 和 ref 为一个数组中的每个元素加 1

11.Serializable 特性在.NET 中有什么作用?

12.在.NET 中的委托是什么?

13.在.NET 中可以自定义委托吗?

14 .NET 默认的委托类型有哪几种?

15.什么是泛型委托?

16. 什么是匿名方法?

17、什么是序列化，什么时候会用到序列化

18、什么是闭包？C#实现方法

四、异常处理

1. C#异常类返回哪些信息？

2. 如何创建一个自定义异常？

3.如何针对不同的异常进行捕捉？

4. 如何避免类型转换时的异常？

5、Debug.Write()和 Trace.Write()之间有什么区别？二者分别应该用于何处？

6.如何解决.net 中的内存泄漏问题？用到过哪些检测工具？

7、内存泄漏和内存溢出的区别是什么？

一、多线程

1、描述线程与进程的区别？

线程(Thread)与进程（Process）二者都定义了某种边界，不同的是进程定义的是应用程序与应用程序之间的边界，不同的进程之间不能共享代码和数据空间，而线程定义的是代码执行堆栈和执行上下文的边界。一个进程可以包括若干个线程，同时创建多个线程来完成某项任务，便是多线程。

而同一进程中的不同线程共享代码和数据空间。用一个比喻来说，如果一个家庭代表一个进程，在家庭内部，各个成员就是线程，家庭中的每个成员都有义务对家庭的财富进行积累，同时也有权利对家庭财富进行消费，当面对一个任务的时候，家庭也可

以派出几个成员来协同完成，而家庭之外的人则没有办法直接消费不属于自己家庭的财产。

--->详解

2、using 关键字有什么用？跟 IDisposable 有啥关系？

有用，using 可以声明 namespace 的引入，还可以实现非托管资源的释放，实现了 IDisposiable 的类在 using 中创建，using 结束后会自动调用该对象的 Dispose 方法，释放资源。加分的补充回答：using 其实等价于 try……finally，用起来更方便。

--->详解

3.前台线程和后台线程有什么区别？

通过将 Thread.IsBackground 属性设置为 true，就可以将线程指定为后台线程

托管线程可以是后台线程，也可以是前台线程。后台线程和前台线程几乎完全相同，只有一处不同，即后台线程不会确保托管执行环境一直运行。一旦托管进程（其中 .exe 文件为托管程序集）中的所有前台线程都停止，系统会停止并关闭所有后台线程。

前台线程：应用必须结束掉所有的前台线程才能结束程序，只要有一个前台线程没退出进程就不会自动退出，当然线程是依附在进程上的，所以你直接把进程 KO 掉了的话自然所有前台线程也会退出。

后台线程：进程可以不考虑后台直接自动退出，进程自动退出后所有的后台线程也会自动销毁。

4、什么是线程互斥?

当多个线程访问同一个全局变量，或者同一个资源(比如打印机)的时候，需要进行线程间的互斥操作来保证访问的安全性。

--->详解

5.如何查看和设置线程池的上下限?

线程池的线程数是有限制的，通常情况下，我们无需修改默认的配置。但在一些场合，我们可能需要了解线程池的上下限和剩余的线程数。线程池作为一个缓冲池，有着其上下限。在通常情况下，当线程池中的线程数小于线程池设置的下限时，线程池会设法创建新的线程，而当线程池中的线程数大于线程池设置的上限时，线程池将销毁多余的线程。

PS：在.NET Framework 4.0 中，每个 CPU 默认的工作者线程数量最大值为 250 个，最小值为 2 个。而 IO 线程的默认最大值为 1000 个，最小值为 2 个。

在.NET 中，通过 ThreadPool 类型提供的 5 个静态方法可以获取和设置线程池的上限和下限，同时它还额外地提供了一个方法来让程序员获知当前可用的线程数量，下面是这五个方法的签名：

① static void GetMaxThreads(out int workerThreads, out int completionPortThreads)

② static void GetMinThreads(out int workerThreads, out int completionPortThreads)

③ static bool SetMaxThreads(int workerThreads, int completionPortThreads)

④ static bool SetMinThreads(int workerThreads, int completionPortThreads)

⑤ static void GetAvailableThreads(out int workerThreads, out int completionPortThreads)

6、Task 状态机的实现和工作机制是什么?

CPS 全称是 Continuation Passing Style，在.NET 中，它会自动编译为：1. 将所有引用的局部变量做成闭包，放到一个隐藏的状态机的类中；2. 将所有的 await 展开成一个状态号，有几个 await 就有几个状态号；3. 每次执行完一个状态，都重复回调状态机的 MoveNext 方法，同时指定下一个状态号；4. MoveNext 方法还需处理线程和异常等问题。

--->详解

7、await 的作用和原理，并说明和 GetResult()有什么区别?

从状态机的角度出发，await 的本质是调用 Task.GetAwaiter() 的 UnsafeOnCompleted(Action)回调，并指定下一个状态号。

从多线程的角度出发，如果 await 的 Task 需要在新的线程上执行，该状态机的 MoveNext() 方法会立即返回，此时，主线程被释放出来了，然后在 UnsafeOnCompleted 回调的 action 指定的线程上下文中继续 MoveNext()和下一个状态的代码。

而相比之下，GetResult()就是在当前线程上立即等待 Task 的完成，在 Task 完成前，当前线程不会释放。注意：Task 也可能不一定在新的线程上执行，此时用 GetResult() 或者 await 就只有会不会创建状态机的区别了。

--->详解

8、多线程有什么用?

（1）发挥多核 CPU 的优势

随着工业的进步，现在的笔记本、台式机乃至商用的应用服务器至少也都是双核的，4 核、8 核甚至 16 核的也都不少见，如果是单线程的程序，那么在双核 CPU 上就浪费了 50%，在 4 核 CPU 上就浪费了 75%。单核 CPU 上所谓的”多线程”那是假的多线程，同一时间处理器只会处理一段逻辑，只不过线程之间切换得比较快，看着像多个线程”同时”运行罢了。多核 CPU 上的多线程才是真正的多线程，它能让你的多段逻辑同时工作，多线程，可以真正发挥出多核 CPU 的优势来，达到充分利用 CPU 的目的。

（2）防止阻塞

从程序运行效率的角度来看，单核 CPU 不但不会发挥出多线程的优势，反而会因为在单核 CPU 上运行多线程导致线程上下文的切换，而降低程序整体的效率。但是单核 CPU 我们还是要应用多线程，就是为了防止阻塞。试想，如果单核 CPU 使用单线程，那么只要这个线程阻塞了，比方说远程读取某个数据吧，对端迟迟未返回又没有设置超时时间，那么你的整个程序在数据返回回来之前就停止运行了。多线程可以防止这个问题，多条线程同时运行，哪怕一条线程的代码执行读取数据阻塞，也不会影响其它任务的执行。

（3）便于建模

这是另外一个没有这么明显的优点了。假设有一个大的任务 A，单线程编程，那么就要考虑很多，建立整个程序模型比较麻烦。但是如果把这个大的任务 A 分解成几个小任务，任务 B、任务 C、任务 D，分别建立程序模型，并通过多线程分别运行这几个任务，那就简单很多了。

9、Task 和 Thread 有区别吗?

Task 和 Thread 都能创建用多线程的方式执行代码，但它们有较大的区别。Task 较新，发布于.NET 4.5，能结合新的 async/await 代码模型写代码，它不止能创建新线程，还能使用线程池（默认）、单线程等方式编程，在 UI 编程领域，Task 还能自动返回 UI 线程上下文，还提供了许多便利 API 以管理多个 Task。

--->详解

10.为什么 GUI 不支持跨线程调用?有什么解决方法?

因为 GUI 应用程序引入了一个特殊的线程处理模型，为了保证 UI 控件的线程安全，这个线程处理模型不允许

其他子线程跨线程访问 UI 元素。解决方法比较多的：

利用 UI 控件提供的方法，Winform 是控件的 Invoke 方法，WPF 中是控件的 Dispatcher.Invoke 方法；

使用 BackgroundWorker；

使用 GUI 线程处理模型的同步上下文 SynchronizationContext 来提交 UI 更新操作

11.说说常用的锁，lock 是一种什么样的锁?

常用的如如 SemaphoreSlim、ManualResetEventSlim、Monitor、ReadWriteLockSlim，lock 是一个混合锁，其实质是 Monitor

12.lock 为什么要锁定一个参数（可否为值类型？）参数有什么要求?

lock 的锁对象要求为一个引用类型。她可以锁定值类型，但值类型会被装箱，每次装箱后的对象都不一样，会导致锁定无效。

对于 lock 锁，锁定的这个对象参数才是关键，这个参数的同步索引块指针会指向一个真正的锁（同步块），这个锁（同步块）会被复用。

13.多线程和异步的区别和联系?

多线程是实现异步的主要方式之一，异步并不等同于多线程。实现异步的方式还有很多，比如利用硬件的特性、使用进程或线程等。

在.NET 中就有很多的异步编程支持，比如很多地方都有 Begin、***End*** 的方法，就是一种异步编程支持，她内部有些是利用多线程，有些是利用硬件的特性来实现的异步编程。

14.线程池的优点有哪些？又有哪些不足

优点：减小线程创建和销毁的开销，可以复用线程；也从而减少了线程上下文切换的性能损失；在 GC 回收时，较少的线程更有利于 GC 的回收效率。

缺点：线程池无法对一个线程有更多的精确的控制，如了解其运行状态等；不能设置线程的优先级；加入到线程池的任务（方法）不能有返回值；对于需要长期运行的任务就不适合线程池。

--->详解

15.Mutex 和 lock 有什么不同？一般用哪一种比较好?

Mutex 是一个基于内核模式的互斥锁，支持锁的递归调用，而 Lock 是一个混合锁，一般建议使用 Lock 更好，因为 lock 的性能更好。

16.Thread 类有哪些常用的属性和方法?

属性:

CurrentContext：获取线程正在其中执行的当前上下文。

CurrentCulture：获取或设置当前线程的区域性。

CurrentPrincipal：获取或设置线程的当前负责人（对基于角色的安全性而言）。

CurrentThread：获取当前正在运行的线程。

CurrentUICulture：获取或设置资源管理器使用的当前区域性以便在运行时查找区域性特定的资源。

IsBackground：获取或设置一个值，该值指示某个线程是否为后台线程。

Priority：获取或设置一个值，该值指示线程的调度优先级。

ThreadState：获取一个值，该值包含当前线程的状态。

方法:

public void Abort()

在调用此方法的线程上引发 ThreadAbortException，以开始终止此线程的过程。调用此方法通常会终止线程。

```
public static void ResetAbort()
```

取消为当前线程请求的 Abort。

public void Start()

开始一个线程。

```
public static void Sleep( int millisecondsTimeout )
```

让线程暂停一段时间。

```
public static bool Yield()
```

导致调用线程执行准备好在当前处理器上运行的另一个线程。由操作系统选择要执行的线程。

-->详解

17、聊聊任务 Task 与并行 Parallel

任务 Task 与并行 Parallel 本质上内部都是使用的线程池，提供了更丰富的并行编程的方式。任务 Task 基于线程池，可支持返回值，支持比较强大的任务执行计划定制等功能，下面是一个简单的示例。Task 提供了很多方法和属性，通过这些方法和属性能够对 Task 的执行进行控制，并且能够获得其状态信息。Task 的创建和执行都是独立的，因此可以对关联操作的执行拥有完全的控制权。

```
//创建一个任务
Task<int> t1 = new Task<int>(n =>
{
    System.Threading.Thread.Sleep(1000);
    return (int)n;
}, 1000);
//定制一个延续任务计划
t1.ContinueWith(task =>
{
    Console.WriteLine("end" + t1.Result);
}, TaskContinuationOptions.AttachedToParent);
t1.Start();
//使用Task.Factory创建并启动一个任务
var t2 = System.Threading.Tasks.Task.Factory.StartNew(() =>
{
    Console.WriteLine("t1:" + t1.Status);
});
Task.WaitAll();
Console.WriteLine(t1.Result);
```

并行 Parallel 内部其实使用的是 Task 对象（TPL 会在内部创建 System.Threading.Tasks.Task 的实例），所有并行任务完成后才会返回。少量短时间任务建议就不要使用并行 Parallel 了，并行 Parallel 本身也是有性能开销的，而且还要进行并行任务调度、创建调用方法的委托等等。

18、下面代码输出结果是什么？为什么？

```
int a = 0;
System.Threading.Tasks.Parallel.For(0, 100000, (i) =>
{
    a++;
});
Console.Write(a);
```

输出结果不稳定，小于等于 100000。因为多线程访问，没有使用锁机制，会导致有更新丢失。

19、多线程并行（Parallelism）和并发（Concurrency）的区别

并行：**同一时刻**有多条指令在多个处理器上**同时**执行，无论从宏观还是微观上都是同时发生的。

并发：是指在**同一时间段内**，宏观上看多个指令看起来是同时执行，微观上看是多个指令进程在快速的切换执行，同一时刻可能只有一条指令被执行。

20、C# Parallel.For 和普通 For 的区别

Parallel 类是.NET 4 中新增的抽象线程类。Parallel.For()方法类似于 C#的 for 循环语句，也是多次执行一个任务。但是使用 Parallel.For()方法，可以并行运行。

对于 Parallel.For、Parallel.Foreach 的使用应该要特别小心，它们的优势是处理列表很长，且对列表内的元素进行很复杂的业务逻辑，且不会使用共享资源，只针对自身的业务逻辑处理，方才能提升效率。

--->详解

21、请简述 async 函数的编译方式?

async/await 是 C# 5.0 推出的异步代码编程模型，其本质是编译为状态机。只要函数前带上 async，就会将函数转换为状态机。

具体在实际编码中怎么用呢?举个例子:

小明早上起床要吃早餐:要刷牙洗脸(2 分钟),要冲牛奶(5 分钟),要煎鸡蛋(3 分钟),然后所有都准备好了,开吃.

如果这个需求以同步的方式进行,总共会花 10 分钟,才能成功吃到早餐.

但是在函数上添加 async,在刷牙洗脸,冲牛奶,煎鸡蛋前添加 await,因为是同步执行,并等待所有线程执行完成,就可以成功吃到早餐,只需要 5 分钟.

还有一种情况:在函数上添加 async,在刷牙洗脸,冲牛奶,煎鸡蛋前不添加 await,这样函数里的要干的事情,将不会阻塞,直接 2 分钟就跳出函数.但是不能成功吃到早餐.因为未添加 await,函数结束后,不能保证所有动作都执行完成.

22、请简述 Task 状态机的实现和工作机制?

CPS 全称是 Continuation Passing Style，在.NET 中，它会自动编译为：

（1）将所有引用的局部变量做成闭包，放到一个隐藏的状态机的类中；

（2）将所有的 await 展开成一个状态号，有几个 await 就有几个状态号；

（3）每次执行完一个状态，都重复回调状态机的 MoveNext 方法，同时指定下一个状态号；

（4）MoveNext 方法还需处理线程和异常等问题。

欢迎关注微信公众：DotNet 开发跳槽

二、Linq 和 EF 面试题

1.EF(Entity Framework)是什么？

实体框架 **EF** 是 **http://ADO.NET** 中的一组支持开发面向数据的软件应用程序的技术，是微软的一个 **ORM** 框架。

主要有三种方式：

Database FirstDatabase First”模式

我们称之为“数据库优先”，前提是你的应用已经有相应的数据库，你可以使用 **EF** 设计工具根据数据库生成数据数据类，你可以使用 **Visual Studio** 模型设计器修改这些模型之间对应关系。

Model FirstModel First 我们称之为“模型优先”，这里的模型指的是“**ADO.NET Entity Framework Data Model**”，此时你的应用并没有设计相关数据库，在 **Visual Studio** 中我们通过设计对于的数据模型来生成数据库和数据类。

Code FirstCode First 模式我们称之为“代码优先”模式，是从 **EF4.1** 开始新建加入的功能。使用 **Code First** 模式进行 **EF** 开发时开发人员只需要编写对应的数据类（其实就是领域模型的实现过程），然后自动生成数据库。这样设计的好处在于我们可以针对概念模型进行所有数据操作而不必关心数据的存储关系，使我们可以更加自然的采用面向对象的方式进行面向数据的应用程序开发。

2. 什么是 ORM?

ORM 指的是面向对象的对象模型和关系型数据库的数据结构之间的互相转换。
(表实体跟表之间的相互转换**)**
ORM 框架有很多，**EF** 框架是 **ORM** 框架的其中一种，是实现了 **ORM** 思想的框架。
O=>表实体
M=>映射关系
R=>数据库.表
--->详解

3. 为什么用 EF 而不用原生的 ADO.NET?

1）.极大的提高开发效率:**EF** 是微软自己的产品,开发中代码都是强类型的,
xiefl 代码效率非常高,自动化程度非常高,命令式的编程.
2）.**EF** 提供的模型设计器非常强大,不仅仅带来了设计数据库的革命,也附带来的自动化模型代码的
功能也极大的提高开发和架构设计的效率.
3）.**EF** 跨数据支持的是 **ORM** 框架主要功能点之一,带来的是可以通过仅仅改变配置就可以做到跨数据库的能力

4. 如何提高 LINQ 性能问题?

提升从数据库中拿数据的速度，可以参考以下几种方法：

1).在数据库中的表中定义合适的索引和键

2).只获得你需要的列（使用 **ViewModel** 或者改进查询）和行（使用 **IQueryable**）

3).尽可能使用一条查询而不是多条

4).只为了展示数据，而不进行后续修改时，可以使用 **AsNoTracking**。它不会影响生成的 **SQL**，但它可以令系统少维护很多数据，从而提高性能

5).使用 **Reshaper** 等工具，它可能会在你写出较差的代码时给出提醒

---->详解

5. 什么是 IEnumerable?

IEnumerable 及 **IEnumerable** 的泛型版本 **IEnumerable** 是一个接口，它只含有一个方法 **GetEnumerator**。**Enumerable** 这个静态类型含有很多扩展方法，其扩展的目标是 **IEnumerable**。
实现了这个接口的类可以使用 **Foreach** 关键字进行迭代（迭代的意思是对于一个集合，可以逐一取出元素并遍历之）。实现这个接口必须实现方法 **GetEnumerator**。
---->详解

6. IEnumerable 的缺点有哪些？

IEnumerable 功能有限，不能插入和删除。
访问 **IEnumerable** 只能通过迭代，不能使用索引器。迭代显然是非线程安全的，每次 **IEnumerable** 都会生成新的 **IEnumerator**，从而形成多个互相不影响的迭代过程。
在迭代时，只能前进不能后退。新的迭代不会记得之前迭代后值的任何变化。

7. 延迟执行 (Lazy Loading)是什么？

大部分 **LINQ** 语句是在最终结果的第一个元素被访问的时候（即在 **foreach** 中调用 **MoveNext** 方法）才真正开始运算的，这个特点称为延迟执行。一般来说，返回另外一个序列（通常为 **IEnumerable** 或 **IQueryable**）的操作，使用延迟执行，而返回单一值的运算，使用立即执行。
IEnumerable 是延迟执行的，当没有触发执行时，就不会进行任何运算。**Select** 方法不会触发 **LINQ** 的执行。一些触发的方式是：**foreach** 循环，**ToList**，**ToArray**，**ToDictionary** 方法等

8. LINQ 可视化工具简单介绍一下？

LINQPad 工具是一个很好的 **LINQ** 查询可视化工具。它由 **Threading in C#**和 **C# in a Nutshell** 的作者 **Albahari** 编写，完全免费。它的下载地址是 **http://www.linqpad.net/**
进入界面后，**LINQPad** 可以连接到已经存在的数据库（不过就仅限微软的 **SQL Server** 系，如果要连接到其他类型的数据库则需要安装插件）。某种程度上可以代替 **SQL Management Studio**，是使用 **SQL Management Studio** 作为数据库管理软件的码农的强力工具，可以用于调试和性能优化（通过改善编译后的 **SQL** 规模）。
LINQPad 支持使用 **SQL** 或 **C#**语句（点标记或查询表达式）进行查询。你也可以通过点击橙色圈内的各种不同格式，看到查询表达式的各种不同表达方式：
Lambda：查询表达式的 **Lambda** 表达式版本，
SQL：由编译器转化成的 **SQL**，通常这是我们最关心的部分，
IL：**IL** 语言

9. LINQ to Object 和 LINQ to SQL 有何区别？

LINQ to SQL 可以将查询表达式转换为 **SQL** 语句，然后在数据库中执行。相比 **LINQ to Object**，则是将查询表达式直接转化为 **Enumerable** 的一系列方法，最终在 **C#**内部执行。
LINQ to Object 的数据源总是实现 **IEnumerable**(所以不如叫做 **LINQ to IEnumerable**)，相对的，**LINQ to SQL** 的数据源总是实现 **IQueryable** 并使用 **Queryable** 的扩展方法。
将查询表达式转换为 **SQL** 语句并不保证一定可以成功。

10. 除了 EF，列举出你知道的 ORM 框架？

dapper EntityFramework、 EJB、Hibernate、IBATIS、SqlSugar 、freesql

11. 在哪些类型额项目中你会选择 EF？为什么？

这个要结合 **EF** 的特点来说：**EF** 主要是以面向对象的思想来做数据库数据操作，对 **Sql** 语句能力没什么要求，开发使用效率高！便于上手，一般来说，使用 **EF** 框架，肯定会比直接使用 **ADO.NET**，消耗的时间多一些。所以在一般企业级开发，管理型系统，对数据性能要求不是特别高的情况下，优先选择 **EF,**这样可以大大的推进开发效率！如果像一些互联网项目中，对性能要求精度很高！可以另外做技术选型，选择原生 **ADO.NET**。

12. 请说明 EF 中映射实体对象的几种状态？

Detached：该实体未由上下文跟踪。刚使用新运算符或某个 **System.Data.Entity.DbSet Create** 方法创建实体后，实体就处于此状态。
Unchanged：实体将由上下文跟踪并存在于数据库中，其属性值与数据库中的值相同。
Added：实体将由上下文跟踪，但是在数据库中还不存在。
Deleted：实体将由上下文跟踪并存在于数据库中，但是已被标记为在下次调用 **SaveChanges** 时从数据库中删除。

Modified：实体将由上下文跟踪并存在于数据库中，已修改其中的一些或所有属性值。欢

迎关注微信公众：DotNet 开发跳槽

三、泛型、集合、数据结构等

1、IList 接口与 List 的区别是什么？

IList 泛型接口是 **Icollection** 接口的子代，并且是所有非泛型列表的基接口。**Ilist** 实现有三种类别：只读、固定大小、可变大小。无法修改只读 **Ilist**。固定大小的 **Ilist** 不允许添加或移除元素，但允许修改现有元素。可变大小的 **Ilist** 允许添加、移除和修改元素。
IList 是个接口,定义了一些操作方法这些方法要你自己去实现，当你只想使用接口的方法时，这种方式比较好.他不获取实现这个接口的类的其他方法和字段，有效的节省空间.
List 是个类型 已经实现了 **IList** 定义的那些方法。

-

```
List List11 =new List ();
```

是想创建一个 **List**，而且需要使用到 **List** 的功能，进行相关操作。
而

-

```
IList IList11 =new List ();
```

只是想创建一个基于接口 **IList** 的对象的实例，只是这个接口是由 **List** 实现的。所以它只是希望使用到 **IList** 接口规定的功能而已。

2. 泛型的主要约束和次要约束是什么？

当一个泛型参数没有任何约束时，它可以进行的操作和运算是非常有限的，因为不能对实参进行任何类型上的保证，这时候就需要用到泛型约束。泛型的约束分为：主要约束和次要约束，它们都使实参必须满足一定的规范，**C#**编译器在编译的过程中可以根据约束来检查所有泛型类型的实参并确保其满足约束条件。

（**1**）主要约束

一个泛型参数至多拥有一个主要约束，主要约束可以是一个引用类型、**class** 或者 **struct**。如果指定一个引用类型（**class**），那么实参必须是该类型或者该类型的派生类型。相反，**struct** 则规定了实参必须是一个值类型。下面的代码展示了泛型参数主要约束：

```
public class ClassT1<T> where T : Exception
{
    private T myException;
    public ClassT1(T t)
{
        myException = t;
    }
    public override string ToString()
{
        // 主要约束保证了myException拥有source成员
        return myException.Source;
    }
}

public class ClassT2<T> where T : class
{
    private T myT;
    public void Clear()
{
        // T是引用类型，可以置null
        myT = null;
    }
}

public class ClassT3<T> where T : struct
{
    private T myT;
    public override string ToString()
{
        // T是值类型，不会发生NullReferenceException异常
        return myT.ToString();
    }
```

(2) 次要约束

次要约束主要是指实参实现的接口的限定。对于一个泛型，可以有 0 到无限的次要约束，次要约束规定了实参必须实现所有的次要约束中规定的接口。次要约束与主要约束的语法基本一致，区别仅在于提供的不是一个引用类型而是一个或多个接口。例如我们为上面代码中的 ClassT3 增加一个次要约束：

```
public class ClassT3<T> where T : struct, IComparable
{
    ......
}
```

3. 如何把一个 array 复制到 arraylist 里?

```
foreach( object arr in array)
{
  arrayist.Add(arr);
}
```

4. **数组和 list 和 arraylist 的区别?**

数组：是存储同类型数据列表，数组在内存中是连续存储的。优点:存储、修改、读取速度快。缺点:初始化需要指定长度，无法扩展，插入数据麻烦

ArrayList: ArrayList 是.Net Framework 提供的用于数据存储和检索的专用类，它是命名空间 System.Collections 下的一部分。它的大小是按照其中存储的数据来动态扩充与收缩的。优点:可扩展，无指定长度，可插入删除

缺点:因存储不同类型，执行装箱拆箱操作，读取、存储速度慢。

List:在数组和 ArrayList 基础上优化，存储通用类型数据列表。优点:可扩展示，初始化无需指定长度，可插入指定位置数据

5. Set 里的元素是不能重复的，那么用什么方法来区分重复与否呢? 是用==还是 equals()? 它们有何区别?

Set 里的元素是不能重复的，那么用 **iterator()**方法来区分重复与否。**equals()**是判读两个 **Set** 是否相等。

equals()和**==**方法决定引用值是否指向同一对像，**equals()**在类中被覆盖，为的是当两个分离的对象的内容和类型相配的话，返回真值。

6. 有 50 万个 int 类型的数字，现在需要判断一下里面是否存在重复的数字，请你简要说一下思路。

.使用 **C#**的 **List** 集合自带的去重方法，例如 **Distinct()**，**GroupBy()**等

.利用 **Dictionary** 的 **Key** 值唯一的特性，**HashSet** 元素值唯一的特性 进行判断

7.数组有没有 length()这个方法? String 有没有 length()这个方法?

数组没有 **length()**这个方法，有 **length** 的属性。**String** 有有 **length()**这个方法。

8.一个整数 List 中取出最大数（找最大值）。不能用 Max 方法。

```
        private static int GetMax(List<int> list)
{
            int max = list[0];
            for (int i = 0; i < list.Count; i++)
            {
                if (list[i]>max)
                {
                    max = list[i];
                }
            }
            return max;
        }
```

9. 利用 IEnumerable 实现斐波那契数列生成?

```
IEnumerable<int> GenerateFibonacci(int n)
{
    if (n >= 1) yield return 1;

    int a = 1, b = 0;
    for (int i = 2; i <= n; ++i)
    {
        int t = b;
        b = a;
        a += t;
        yield return a;
    }
}
```

----->详解

10.请利用 foreach 和 ref 为一个数组中的每个元素加 1

注意 **foreach** 不能用 **var** ，也不能直接用 **int** ，需要 **ref int** ，注意 **arr** 要转换为 **Span** 。

```
int[] arr = { 1, 2, 3, 4, 5};
Console.WriteLine(string.Join(",", arr)); // 1,2,3,4,5

foreach (ref int v in arr.AsSpan())
{
    v++;
}

Console.WriteLine(string.Join(",", arr)); // 2,3,4,5,6
```

----->详解

11. Serializable 特性在.NET 中有什么作用？

通过上面的流类型可以方便地操作各种字节流，但是如何把现有的实例对象转换为方便传输的字节流，就需要使用序列化技术。对象实例的序列化，是指将实例对象转换为可方便存储、传输和交互的流。在.NET 中，通过 Serializable 特性提供了序列化对象实例的机制，当一个类型被申明为 Serializable 后，它就能被诸如 BinaryFormatter 等实现了 IFormatter 接口的类型进行序列化和反序列化。

12.在.NET 中的委托是什么？

委托是寻址的.NET 版本。在 C++中，函数指针只不过是一个指向内存位置的指针，它不是类型安全的。我们无法判断这个指针实际指向什么，像参数和返回类型等项久更无从知晓了。而.NET 委托完全不同，委托是类型安全的类，它定义了返回类型和参数的类型。委托类不仅包含对方法的引用，也可以包含对多个方法的引用。---->详解

13.在.NET 中可以自定义委托吗？

声明一个委托类型，它的实例引用一个方法，该方法获取一个 int 参数，返 void。

```
public delegate void Feedback(int num);
```

理解委托的一个要点是它们的安全性非常高。在定义委托时，必须给出它所表示的方法的签名和返回类型等全部细节。

理解委托的一种比较好的方式是把委托当作这样一件事情：它给方法的签名和返回类型指定名称。其语法类似于方法的定义，需要在定义方法的前面加上 delegate 关键字。定义委托基本上就是定义一个新的类，所以可以在任何地方定义类的相同地方定义委托，也就是说，可以在另一个类的内部定义，也可以在任何类的外部定义，还可以在名称控件中把委托定义为定义为顶层对象。访问修饰符可以是 public/private/protected 等。

---->详解

14 .NET 默认的委托类型有哪几种？

1）**Action ＜ T ＞**泛型 Action 委托表示引用一个 void 返回类型的方法。这个委托类存在 16 种重载方法。例如 Action<in T1,In T2>调用没有参数的方法

2）2）.Func< T >Func 调用带返回类型的方法。有 16 种重载方法。

例如 **Func** 委托类型可以调用带返回类型且无参数的方法，**Func<in T,out TResult>**委托类型调用带有 **4** 个参数和一个返回类型的方法。

---->详解

15.什么是泛型委托?

A　ction 就是泛型委托。注意事项：

1）.建议尽量使用这些委托类型，而不是在代码中定义更多的委托类型。这样可以减少系统中的类型数目，同时简化编码

2）.如果需要使用 ref 或 out 关键字，以传引用的方式传递一个参数，就可能不得不定义自己的委托：delegate void Test(ref int i)

3).如果委托要通过 C#的 params 关键字获取可变数量的额参数，要为委托的任何桉树指定默认值,或者要对委托的泛型类型参数进行约束,也必须定义自己的委托类型 delegate void EventHandler(Object sender, TEventArgs e)where TEventArgs : EventArgs;

4).使用获取泛型实参和返回值的委托时，可利用逆变与协变。逆变：父类转换为子类；协变：子类转换为父类

---->详解

16. 什么是匿名方法?

匿名方法是用作委托的参数的一段代码。

```
  //匿名方法,例1
Func<int, int> anon = delegate(int i)
{
        i = i+1;
        return i;
};
//输出2
Console.WriteLine(anon(1));
//匿名方法,例2
Action<int> anon2 = delegate(int i)
{
        i = i + 1;
};
//输出2
Console.WriteLine(anon(1));
```

---->详解

17、什么是序列化，什么时候会用到序列化

序列化（Serialization)是将对象的状态信息转换为可以存储或传输的形式的过程。在序列化期间，对象将其当前状态写入到临时或持久性存储区。以后，可以通过从存储区中读取或反序列化对象的状态，重新创建该对象。

可以用到序列化的地方：

a、数据持久化：比如一个电商平台，有数万个用户并发访问的时候会产生数万个 session 对象，这个时候内存的压力是很大的。我们可以把 session 对象序列化到硬盘中,需要时在反序列化，减少内存压力。

b、网络传输：我们将系统拆分成多个服务之后，服务之间传输对象，不管是何种类型的数据，都必须要转成二进制流来传输，接受方收到后再转为数据对象。

18、什么是闭包？C#实现方法

闭包就是能够读取其他函数内部变量的函数。C#通过 Lambda 表达式可以访问 Lambda 表达式块外部的变量，这成为 c#闭包。当引用外部变量时，需要注意，外部变量变化时，lambda 表达式的结果也可能会随着外部变量变化而变化。如下面的例子：

```
int y = 5;
Func<int, int> lambda = x => x + y;
Console.WriteLine(lambda(1));
y = 10;
Console.WriteLine(lambda(1));
```


四、异常处理

1. C#异常类返回哪些信息？

C#中，所有异常都继承自 System.Exception 类，Exception 类定义了 C#异常应该具有的信息和方法。值得注意的属性有：

```
public virtual string Message { get; }// 错误的信息，文字描述
public virtual string StackTrace { get; }// 发生异常的调用堆栈信息
public System.Reflection.MethodBase TargetSite { get; }//引发这个错误的方法
public Exception InnerException { get; }// 子异常
```

2. 如何创建一个自定义异常?

根据类继承原则和异常处理原则，我们可以使用以下方式来自定义一个类：

```
public class CustomException : Exception
{
}
```

--->详解

3.如何针对不同的异常进行捕捉?

```
public class Program
    {
        public static void Main(string[] args)
        {
            Program p = new Program();
            p.RiskWork();

            Console.ReadKey();
        }

        public void RiskWork()
        {
            try
            {
                // 一些可能会出现异常的代码
            }
            catch (NullReferenceException ex)
            {
                HandleExpectedException(ex);
```

```
            }
            catch (ArgumentException ex)
            {
                HandleExpectedException(ex);
            }
            catch (FileNotFoundException ex)
            {
                HandlerError(ex);
            }
            catch (Exception ex)
            {
                HandleCrash(ex);
            }
        }

        // 这里处理预计可能会发生的，不属于错误范畴的异常
        private void HandleExpectedException(Exception ex)
        {
            // 这里可以借助 log4net 写入日志
            Console.WriteLine(ex.Message);
        }
        // 这里处理在系统出错时可能会发生的，比较严重的异常
        private void HandlerError(Exception ex)
        {
            // 这里可以借助 log4net 写入日志
            Console.WriteLine(ex.Message);
            // 严重的异常需要抛到上层处理
            throw ex;
        }

        // 这里处理可能会导致系统崩溃时的异常
        private void HandleCrash(Exception ex)
        {
            // 这里可以借助 log4net 写入日志
            Console.WriteLine(ex.Message);
            // 关闭当前程序
            System.Threading.Thread.CurrentThread.Abort();
        }
    }
```

4. 如何避免类型转换时的异常？

其中有些是确定可以转换的（比如将一个子类类型转为父类类型），而有些则是尝试性的（比如将基类引用的对象转换成子类）。当执行常识性转换时，我们就应该做好捕捉异常的准备。当一个不正确的类型转换发生时，会产生 InvalidCastException 异常，有时我们

会用 try-catch 块做一些尝试性的类型转换，这样的代码没有任何错误，但是性能却相当糟糕，为什么呢？异常是一种耗费资源的机制，每当异常被抛出时，异常堆栈将会被建立，异常信息将被加载，而通常这些工作的成本相对较高，并且在尝试性类型转换时，这些信息都没有意义。在.NET 中提供了另外一种语法来进行尝试性的类型转换，那就是关键字 is 和 as 所做的工作。（1）is 只负责检查类型的兼容性，并返回结果：true 和 false。→ 进行类型判断两者的共同之处都在于：不会抛出异常！综上比较，as 较 is 在执行效率上会好一些，在实际开发中应该量才而用，在只进行类型判断的应用场景时，应该多使用 is 而不是 as。

```
public static void Main(string[] args)
  {
      object o = new object();
      // 执行类型兼容性检查
      if(o is ISample)
      {
          // 执行类型转换
          ISample sample = (ISample)o;
          sample.SampleShow();
      }

      Console.ReadKey();
  }
```

（2）as 不仅负责检查兼容性还会进行类型转换，并返回结果，如果不兼容则返回 null 。→ 用于类型转型

```
public static void Main(string[] args)
  {
      object o = new object();
      // 执行类型兼容性检查
      ISample sample = o as ISample;
      if(sample != null)
      {
          sample.SampleShow();
      }
      Console.ReadKey();
  }
```

两者的共同之处都在于：不会抛出异常！综上比较，as 较 is 在执行效率上会好一些，在实际开发中应该量才而用，在只进行类型判断的应用场景时，应该多使用 is 而不是 as。

5、Debug.Write()和 Trace.Write()之间有什么区别？二者分别应该用于何处？

Debug.Write 是调试的时候向跟踪窗口输出信息。

当编译模式为 debug 的时候才有效，为 release 的时候 Debug.Write 在编译的时候会忽略,而 Trace 则是在 debug 和 release 两种模式下均可以向跟踪窗口输出信息。

3.如何解决.net 中的内存泄漏问题？用到过哪些检测工具？

.NET 内存泄漏，更准确的说应该是对象超过生命周期而不能被 GC 回收。

常见的内存泄露有：

a、静态引用；b、控件不使用后未销毁；c、调用非托管资源而未释放；d、事件注册后未解除注册，等。

解决方案：

(1) Dispose()的使用

如果使用的对象提供 Dispose（）方法，那么当你使用完毕或在必要的地方（比如 Exception）调用该方法，
特别是对非托管对象,一定要加以调 用，以达到防止泄露的目的。

(2) using 的使用

using 除了引用 Dll 的功用外，还可以限制对象的适用范围，当超出这个界限后对象自动释放，比如 using 语句的用途定义一个范围，将在此范围之外释放一个或多个对象。

(3) 事件的卸载

这个不是必须的，推荐这样做。之前注册了的事件，关闭画面时应该手动注销，有利于 GC 回收资源。

(4) API 的调用

一般的使用 API 了就意味着使用了非托管资源，需要根据情况手动释放所占资源，特别是在处理大对象时。

4.5 继承 IDisposable 实现自己内存释放接口 Net 如何继承 IDisposable 接口，实现自己的 Dispose()函数

(5)弱引用（WeakReference ）

通常情况下，一个实例如果被其他实例引用了，那么他就不会被 GC 回收，而弱引用的意思是，如果一个实例没有被其他实例引用（真实引用），

而仅仅是被弱引 用，那么他就会被 GC 回收。

等等

诊断工具：

a、大多使用 windows 自带的 perfmon.msc，b、用过的工具里面 CLRProfiler 和 dotTrace 还行，windbg 也还行。不过坦白的说，准确定位比较费劲，最好还是按常规的该 Dispose 的加 Dispose，也可以加 GC.Collect（）

6、内存泄漏和内存溢出的区别是什么？

简单来说，操作系统就像资源分配人员，你要使用内存的时候分给你，你用完了还给它。如果你使用了没有分配给你的内存就是内存溢出，如果你用完了没有了就是内存泄漏。

会引起的问题：

内存溢出存在的问题是你用了没有分配给你的内存，系统是不知道的，他又把内存分配给了其他程序，结果就是别人也写了或者读了这个内存。程序可能崩溃。当然也可能没问题，所以内存溢出往往不好查。

内存泄漏的问题就比理解，你没有还给系统，系统的内存就越来越少。直到没有可用内存。

泄漏是占着不用了，溢出是用不该用的地方；溢出一般会出事，泄漏在内存无限时不会出事。

泄漏是说你的程序有 BUG 导致内存不释放。溢出是指内存不够用了 导致不够用的原因很多 泄漏只是其中一种。

更多初中高面试题可以扫码关注公众号↓ ↓ ↓ ↓ ↓ ↓ ↓ 欢迎关注

北京群 群1：219690210，群2：377501688，群3：262827065，成都群：209844460
上海群 ：376029918 杭州群：376029918 广州群：344744167 深圳群：542733289
西安群：542733289 高级群：165150112
微信公众号:DotNet开发跳槽