**内存管理**

1. 托管内存
2. 非托管内存
3. 栈
4. 堆

**栈是内存向下填充的，主要用于值类型**

**堆内存是向上分配，主要用于引用类型对象的创建**

托管堆

只要在栈上有引用指向托管堆对象，则托管堆上的对象就不会被删除。

非托管堆

**垃圾回收**

1. 程序会根据栈上的引用对象在托管堆里面依次创建对象。
2. 程序会根据栈上的引用对象的释放释放对应托管堆上的对象。
3. 造成托管堆上的分配的内存不连续。
4. 垃圾回收机制会调整托管堆上的对象分布，将未释放的内存放到堆的顶部。
5. 而非托管堆不收垃圾回收机制影响。
6. 缺点：额外的整理内存的工作。

优点：页面分配不频繁，提高I/O性能。

1. 调用GC.Collect()可以调用垃圾回收
   1. 它并不能保证所有的未被引用的对象都会被释放。
   2. 大量内存取消的时候才适合调用。
2. 垃圾回收机制通过托管对象的代数提高垃圾回收的性能。

**非托管资源的释放**

非托管资源： 句柄、网络连接、数据库连接

对象的析构

1. 析构函数
2. 实现IDisposable接口

**析构函数**Finalize

1. Finalize() 会调用~[类名]析构函数。
2. 由于垃圾回收机制，析构函数并不能马上被调用，这样造成非托管资源不能马上被释放。
3. 如果有非托管资源要释放的时候，析构函数需要被调用两次。
4. 第一次调用析构函数。
5. 第二次释放对象的内存空间。

**IDisposable接口**

Interface IDisposable

{

void Dispose();

}

1. 用户可以在使用完对象之后，显示的调用释放非托管资源。
2. Using关键字在释放之间对象的时候，会调用实现了IDisposable接口的对象。
3. 推荐使用try…catch…finally 中调用要释放对象的IDisposable接口。

推荐在析构函数和IDisposable接口中都调用释放非托管资源的函数

**指针和引用**

共同点：都存储了引用（指向）对象的内存地址。

不同点：C#不允许直接访问指向的内存地址，引用是一个类型安全的指针。

C++可以随意修改指针地址，修改指针地址指向的数据。

C#防止用户无意的破坏内存中的数据。

C#支持指针的目的：

1. 兼容API。
   1. 通过DllImport/IntPtr尽量避免使用指针。
2. 提高代码性能。

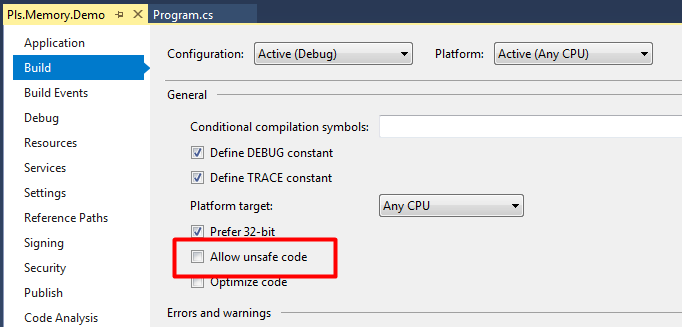
C#支持指针的缺点：

1. 代码复杂度提高。
2. 需要用户给程序额外的安全信任。
3. 不能使用CLR内存检查，安全性不高。

**指针的使用**

1.unsafe标记不安全的代码，可以标识在类、函数、类成员变量上。

2.默认的C#编译器不支持不安全的代码，需要额外勾选



3.指针不能声明为托管类指针，因为和GC冲突。

4.&取地址，\*取指针指向的数据。

5.指针可以算术运算。+/-/+=/-=/++/--

6.sizeof 可以计算结构体大小，且必须放在unsafe中

7.可以用fixed关键字创建类成员变量的指针，fixed关键字告诉GC，这个对象不可以被移动。

**指针的应用**

1. 分配栈数组(unsafe)

通过stackalloc在栈上分配数组