**不安全代码和指针（C# 编程指南）**

为了保持类型安全，默认情况下，C# 不支持指针算法。不过，通过使用 [unsafe](https://msdn.microsoft.com/zh-cn/library/chfa2zb8.aspx) 关键字，可以定义可使用指针的不安全上下文。有关指针的更多信息，请参见主题[指针类型](https://msdn.microsoft.com/zh-cn/library/y31yhkeb.aspx)。

| **https://wizardforcel.gitbooks.io/msdn-csharp/content/img/note.jpg 注意** |
| --- |
| 在公共语言运行时 (CLR) 中，不安全代码是指无法验证的代码。C# 中的不安全代码不一定是危险的；只是其安全性无法由 CLR 进行验证的代码。因此，CLR 只对在完全受信任的程序集中的不安全代码执行操作。如果使用不安全代码，由您负责确保您的代码不会引起安全风险或指针错误。 |

**不安全代码概述**

不安全代码具有下列属性：

* 方法、类型和可被定义为不安全的代码块。
* 在某些情况下，通过移除数组界限检查，不安全代码可提高应用程序的性能。
* 当调用需要指针的本机函数时，需要使用不安全代码。
* 使用不安全代码将引起安全风险和稳定性风险。
* 在 C# 中，为了编译不安全代码，必须用 [/unsafe](https://msdn.microsoft.com/zh-cn/library/ct597kb0.aspx) 编译应用程序。

**固定大小的缓冲区（C# 编程指南）**

在 C# 中，可以使用 [fixed](https://msdn.microsoft.com/zh-CN/library/f58wzh21.aspx) 语句在数据结构中创建带有固定大小数组的缓冲区。使用现有代码（如使用其他语言、预先存在的 DLL 或 COM 项目编写的代码）时，这种方法非常有用。固定数组可采用允许普通结构成员使用的任何特性或修饰符。唯一的限制是，数组类型必须是 **bool**、**byte**、 **char**、 **short**、**int**、**long**、**sbyte**、**ushort**、**uint**、**ulong**、**float** 或 **double**。

private fixed char name[30];

**备注**

在早期版本的 C# 中，声明 C++ 样式的固定大小结构是很困难的，因为包含数组的 C# 结构不包含数组元素。相反，该结构包含对元素的引用。

C# 2.0 添加了在 [struct](https://msdn.microsoft.com/zh-CN/library/ah19swz4.aspx)（当用在 [unsafe](https://msdn.microsoft.com/zh-CN/library/chfa2zb8.aspx) 代码块中时）中嵌入固定大小的数组的功能。

例如，在 C# 2.0 之前，下面的 **struct** 的大小为 8 字节。 pathName 数组是对堆分配数组的引用：

public struct MyArray

{

public char[] pathName;

private int reserved;

}

从 C# 2.0 开始，**struct** 可以包含嵌入的数组。在下面的示例中，fixedBuffer 数组有固定的大小。若要访问数组的元素，应使用 **fixed** 语句建立指向第一个元素的指针。 **fixed** 语句将 fixedBuffer 实例固定到内存中的特定位置。

namespace FixedSizeBuffers

{

internal unsafe struct MyBuffer

{

public fixed char fixedBuffer[128];

}

internal unsafe class MyClass

{

public MyBuffer myBuffer = default(MyBuffer);

}

internal class Program

{

static void Main()

{

MyClass myC = new MyClass();

unsafe

{

// Pin the buffer to a fixed location in memory.

fixed (char\* charPtr = myC.myBuffer.fixedBuffer)

{

\*charPtr = 'A';

}

}

}

}

}

128 个元素的 **char** 数组的大小为 256 字节。在固定大小的 [char](https://msdn.microsoft.com/zh-CN/library/x9h8tsay.aspx) 缓冲区中，每个字符始终占用两个字节，而与编码无关。即使将 char 缓冲区封送到具有 CharSet = CharSet.Auto 或 CharSet = CharSet.Ansi 的 API 方法或结构，也是如此。有关更多信息，请参见 [CharSet](https://msdn.microsoft.com/zh-CN/library/system.runtime.interopservices.charset.aspx)。

另一种常见的固定大小的数组是 [bool](https://msdn.microsoft.com/zh-CN/library/c8f5xwh7.aspx) 数组。 **bool** 数组中元素的大小始终为一个字节。 **bool** 数组不适合于创建位数组或缓冲区。

| **https://wizardforcel.gitbooks.io/msdn-csharp/content/img/note.jpg 注意** |
| --- |
| 除了用 [stackalloc](https://msdn.microsoft.com/zh-CN/library/cx9s2sy4.aspx) 创建的内存之外，C# 编译器和公共语言运行时 (CLR) 不执行任何安全缓冲区溢出检查。与所有不安全代码一样，请谨慎使用。 |

不安全缓冲区与常规数组在以下方面不同：

* 不安全缓冲区只能用在不安全上下文中。
* 不安全缓冲区始终是向量（或一维数组）。
* 数组的声明应包括计数，如 char id[8]。而不能使用 char id[]。
* 不安全缓冲区只能是不安全上下文中的结构的实例字段。

**指针类型（C# 编程指南）**

在不安全的上下文中，类型可以是指针类型、值类型或引用类型。指针类型声明采用下列形式之一：

type\* identifier;

void\* identifier; //allowed but not recommended

以下任一类型均可为指针类型：

* [sbyte](https://msdn.microsoft.com/zh-CN/library/d86he86x.aspx)、[byte](https://msdn.microsoft.com/zh-CN/library/5bdb6693.aspx)、[short](https://msdn.microsoft.com/zh-CN/library/ybs77ex4.aspx)、[ushort](https://msdn.microsoft.com/zh-CN/library/cbf1574z.aspx)、[int](https://msdn.microsoft.com/zh-CN/library/5kzh1b5w.aspx)、[uint](https://msdn.microsoft.com/zh-CN/library/x0sksh43.aspx)、[long](https://msdn.microsoft.com/zh-CN/library/ctetwysk.aspx)、[ulong](https://msdn.microsoft.com/zh-CN/library/t98873t4.aspx)、[char](https://msdn.microsoft.com/zh-CN/library/x9h8tsay.aspx)、[float](https://msdn.microsoft.com/zh-CN/library/b1e65aza.aspx)、[double](https://msdn.microsoft.com/zh-CN/library/678hzkk9.aspx)、[decimal](https://msdn.microsoft.com/zh-CN/library/364x0z75.aspx) 或 [bool](https://msdn.microsoft.com/zh-CN/library/c8f5xwh7.aspx)。
* 任何[枚举](https://msdn.microsoft.com/zh-CN/library/sbbt4032.aspx)类型。
* 任何指针类型。
* 任何仅包含非托管类型字段的用户定义的结构类型。

指针类型不从 [object](https://msdn.microsoft.com/zh-CN/library/9kkx3h3c.aspx) 继承，并且指针类型与 **object** 之间不存在转换。此外，装箱和取消装箱不支持指针。但是，你可在不同的指针类型之间以及指针类型和整型之间进行转换。

在同一个声明中声明多个指针时，星号 (\*) 仅与基础类型一起写入；而不是用作每个指针名称的前缀。例如：

int\* p1, p2, p3; // Ok

int \*p1, \*p2, \*p3; // Invalid in C#

指针不能指向引用或包含引用的 [struct](https://msdn.microsoft.com/zh-CN/library/ah19swz4.aspx)，因为无法对对象引用进行垃圾回收，即使有指针指向它也是如此。垃圾回收器并不跟踪是否有任何类型的指针指向对象。

myType\* 类型的指针变量的值为 myType 类型的变量的地址。下面是指针类型声明的示例：

| **示例** | **说明** |
| --- | --- |
| int\* p | p 是指向整数的指针。 |
| int\*\* p | p 是指向整数的指针的指针。 |
| int\*[] p | p 是指向整数的指针的一维数组。 |
| char\* p | p 是指向字符的指针。 |
| void\* p | p 是指向未知类型的指针。 |

指针间接寻址运算符 \* 可用于访问位于指针变量所指向的位置的内容。例如，请考虑以下声明：

int\* myVariable;

表达式 *myVariable 表示在 myVariable 中包含的地址处找到的 \*int* 变量。

主题[fixed 语句（C# 参考）](https://msdn.microsoft.com/zh-CN/library/f58wzh21.aspx)和[指针转换（C# 编程指南）](https://msdn.microsoft.com/zh-CN/library/etb4xxec.aspx)包含了多个指针示例。下面的示例显示需要 **unsafe** 关键字和 **fixed** 语句以及如何递增内部指针。你可将此代码粘贴到控制台应用程序的 Main 函数中来运行它。（记住在**“项目设计器”**中启用不安全代码；选择菜单栏上的**“项目”**、**“属性”**，然后选择**“生成”**选项卡中的**“允许不安全代码”**。）

// Normal pointer to an object.

int[] a = new int[5] {10, 20, 30, 40, 50};

// Must be in unsafe code to use interior pointers.

unsafe

{

// Must pin object on heap so that it doesn't move while using interior pointers.

fixed (int\* p = &a[0])

{

// p is pinned as well as object, so create another pointer to show incrementing it.

int\* p2 = p;

Console.WriteLine(\*p2);

// Incrementing p2 bumps the pointer by four bytes due to its type ...

p2 += 1;

Console.WriteLine(\*p2);

p2 += 1;

Console.WriteLine(\*p2);

Console.WriteLine("--------");

Console.WriteLine(\*p);

// Deferencing p and incrementing changes the value of a[0] ...

\*p += 1;

Console.WriteLine(\*p);

\*p += 1;

Console.WriteLine(\*p);

}

}

Console.WriteLine("--------");

Console.WriteLine(a[0]);

Console.ReadLine();

// Output:

//10

//20

//30

//--------

//10

//11

//12

//--------

//12

你无法对 void\* 类型的指针应用间接寻址运算符。但是，你可以使用强制转换将 void 指针转换为任何其他指针类型，反之亦然。

指针可以为 **null**。将间接寻址运算符应用于 null 指针将导致由实现定义的行为。

请注意，在方法之间传递指针会导致未定义的行为。示例通过 Out 或 Ref 参数或作为函数结果返回一个指向局部变量的指针。如果已在固定块中设置指针，则它指向的变量不再是固定的。

下表列出了可在不安全的上下文中对指针执行的运算符和语句：

| **运算符/语句** | **用法** |
| --- | --- |
| \* | 执行指针间接寻址。 |
| -> | 通过指针访问结构的成员。 |
| [] | 为指针建立索引。 |
| & | 获取变量的地址。 |
| ++ 和 -- | 递增和递减指针。 |
| + 和 - | 执行指针算法。 |
| ==、!=、<、>、<= 和 >= | 比较指针。 |
| **stackalloc** | 在堆栈上分配内存。 |
| **fixed** 语句 | 临时固定变量以便找到其地址。 |