به نام کسی که آفرید از عدم به انسان عطا کرد فکر و قلم

موضوع مقاله: اشاره گرها در C#

تاریخ ارائه : ۸۹/۶/۱۴

دانشجو: نسترن عباس زاده

نام استاد: مهندس محمد سلیمی

C# یک زبان برنامه نویسی کامل برای از میان برداشتن نگرانی برنامه نویسان از مدیریت حافظه می باشد. چرا که به مدد garbage collector و استفاده از reference ها مدیریت حافظه تا حدود زیادی بصورت خودکار انجام می شود. با این همه شاید مواردی پیش آید که نیاز به دسترسی مستقیم به حافظه باشد. بعنوان مثال شما قصد استفاده از تابعی از یک DLL که با .NET نوشته نشده است را دارید بطوریکه آن تابع پارامتری از نوع اشاره گر داشته باشد. یا اینکه شما قصد بالا بردن سرعت و کارایی برنامه تان را داشته باشید. آنچه در ادامه می خوانید امکانات C# برای دسترسی مستقیم به محتویات حافظه را بازگو می کند.

اشاره گرها:

شما reference ها را به سادگی در جای جای برنامه خود استفاده کرده اید. یک reference به زبان ساده تر یک اشاره گر از نوع type-safe می باشد. قبلا" دیده اید که متغیرها که در برگیرنده object ها و یا آرایه ها می باشند، چگونه آدرس خانه هایی از حافظه را که داده های اصلی در آنجا ذخیره شده اند را در خود نگه می دارند. به زبات ساده تری می توان گفت که اشاره گر یک متغیر است با این تفاوت که به جای ذخیره کردن مقدار (value) ، آدرس خانه یا خانه هایی از حافظه را بعنوان reference ذخیره می کند. تفاوت اینجاست که c# به راحتی دسترسی مستقیم به خانه ای از حافظه که یک متغیر از نوع reference آن را اشغال کرده، نمی دهد.

Reference ها در c# به منظور استفاده ساده تر از یک زبان برنامه نویسی پیشرفته و همچنین مهم تر از همه برای جلوگیری از برخی اشتباهات سهوی در هنگام برنامه نویسی که منجر به خرابی و از بین رفتن محتویات حافظه می شود، طراحی شده اند. اما با استفاده از اشاره گرها می توان به آدرس حافظه اصلی دست پیدا کرده و کارهای مختلفی را با آن انجام داد. دو دلیل عمده برای استفاده از اشاره گرها در c# وجود دارند که عبارتند از:

۱- سازگاری با توابع غیر .NET مثل Windows API ها.

۲- کارایی و سرعت بالاتر

دسترسی به حافظه Low-level کار ساده ای نیست و اگر با دقت کافی همراه نباشد به قیمت ایجاد باگهایی در برنامه تمام می شود که پیدا کردن آن ها به سادگی دیگر باگها نیست. همچنین روش کدنویسی و Syntax استفاده از اشاره گرها نسبت به reference ها از پیچیدگی بیشتری برخوردار است لذا برنامه نویس برای بکارگیری آنها باید از دانش و مهارت کافی در این زمینه برخوردار باشد.

علی رغم همه اینها، اشاره گرها همچنان بعنوان ابزاری قدرتمند و انعطاف پذیر برای نوشتن کدهایی با راندمان بلا شناخته می شوند. اما قویا" توصیه می شود که فقط و فقط در صورت نیاز مبرم باید به سراغ استفاده از اشاره گرها رفت.

نوشتن unsafe code :

بخاطر ریسک استفاده از اشاره گرها، c# استفاده از آنها را تنها در بلاکهایی که برای این منظور در نظر گرفته شده اند مجاز می داند. عبارتی که برای مشخص کردن این بلاک بکار می رود unsafe است. بعنوان مثال تابعی که در آن از اشاره گر استفاده می شود بصورت زیر تعریف می شود.

```
unsafe int GetSomeNumber()
{
    // code that can use pointers
}
```

واژه کلیدی **unsafe** می تواند برای هر تابعی صرف نظر از نوع آن که مثلا" **static** باشد یا **virtual** بکار رود.

همچنین می توان **unsafe** را برای کل یک کلاس یا **structure** بکار برد که این بدان معناست که همه عضوها و توابع آن بعنوان **unsafe** فرض شده اند.

```
unsafe class MyClass
{
    // any method in this class can now use pointers
}
```

مشابها" می توان تنها یک عضو کلاس را **unsafe** در نظر گرفت.

```
class MyClass
{
     unsafe int* pX;   // declaration of a pointer field in a class
}
```

همچنین می توان تنها بخشی از یک تابع را بعنوان بلاک **unsafe** در نظر گرفت.

```
void MyMethod()
{
     // code that doesn't use pointers
     unsafe
     {
         // unsafe code that uses pointers here
     }
     // more 'safe' code that doesn't use pointers
}
```

توجه داشته باشید که متغیرهای محلی که در توابع تعریف می شوند **نمی توانند** **unsafe** در نظر گرفته شوند..

```
int MyMethod()
{
    unsafe int* pX;  // WRONG
}
```

کامپایل کردن کدهای unsafe:

کامپایل کردن معمولی برنامه ای که شامل **unsafe code** باشد با خطا همراه خواهد بود پس باید به نوعی کامپایلر را از اینکه **unsafe code** ها را مد نظر داشته باشد آگاه کرد. برای این منظور اگر از طریق **Command Line** قصد کامپایل کردن برنامه را داشته باشیم، می بایست حتما" از واژه کلیدی **unsafe** در خط دستور استفاده کرد. مثلا" برای کامپایل کردن فایل **ySource.cs** که حاوی کدهای **unsafe** می باشد، اینگونه عمل می کنیم:

csc /unsafe MySource.cs و یا **csc -unsafe MySource.cs**

اگر از **Visual Studio 2005** برای کامپایل کردن برنامه استفاده شود می بایست در **Build Tab** که در پنجره **Properties** پروژه مورد نظر موجود است، گزینه **unsafe code Allow** را انتخاب کرد.

نحوه تعریف اشاره گرها:

حال که بلاکی از کد را توسط **unsafe** برای بکارگیری اشاره گرها تعیین نمودیم می توان اشاره گرها را بصورت زیر در آن تعریف کنیم.

```
int* pWidth, pHeight;

double* pResult;

byte*[] pFlags;
```

این کد چهار متغیر تعریف می کند که **pWidth** و **pHeight** اشاره گرهایی از نوع **integer** و **pResult** اشاره گری از نوع **double** **pFlag** ارایه ای از اشاره گرها از نوع **byte** می باشند. همانطور که مشاهده می شود بهتر است در اینجا برای تعریف اشاره گرها اولین حرف آنها را **p** انتخاب کنیم تا بدین ترتیب براحتی از سایر متغیرها متمایز باشند.

برنامه نویسان **C++** باید متوجه باشند که بین تعریف اشاره گر در **C++** و **C#** فرق کوچکی وجود دارد و آن اینست که در **C#** علامت * چسبیده به نوع اشاره گر است حال آنکه در **C++** علامت * چسبیده به نام اشاره گر می باشد.

```
int* p1, p2, p3    //ok

int *p1, *p2, *p3  // Invalid in C#
```

برای کار با اشاره گرها نیاز است که با دو عملگر & و * آشنا شویم:

& به معنای گرفتن آدرس یک متغیر می باشد که آدرس یک نوع داده مقدار دهی شده را بر می گرداند.

* به معنای گرفتن یک خانه حافظه با استفاده از آدرس آن خانه می باشد. پس اگر * در ابتدای یک اشاره گر آورده شود مقدار موجود در خانه ای از حافظه که اشاره گر بدان اشاره می کند بدست می آید.

کد زیر مثالهایی در این مورد را نشان می دهد.

```
int x = 10;

int* pX, pY;

pX = &x;

pY = pX;

*pY = 20;
```

در کد فوق متغیر **x** با عدد ۱۰ مقدار دهی شده است سپس دو اشاره گر **pX** و **pY** تعریف شده اند. اشاره گر **pX** با آدرس خانه ای از حافظه که متغیر **x** بدان اشاره می کند مقدار دهی می شود و در خط بعد اشاره گر **pY** برابر با **pX** قرار داده می شود.

یعنی **pY** نیز به همان جایی اشاره می کند که **pX** اشاره می کند. سپس در خط آخر مقدار خانه ای از حافظه که اشاره گر **pY** بدان اشاره می کند برابر با عدد ۲۰ قرار داده می شود. پس بسادگی می توان نتیجه گرفت که در پایان این روند مقدار متغیر **x** به ۲۰ تغییر می یابد.

هر اشاره گر می تواند به هر نوع مقدار (**value type**) به شرط آنکه **unmanaged** باشد، اشاره کند. این **unmanaged type** ها در

#c عبارتند از:

- **sbyte, byte, short, ushort, int, uint, long, ulong, char, float, double, decimal, bool**
- **enum type**
- **pointer type**
- **Any user-defined struct type (contains fields of unmanaged types only).**

اشاره گر نمی تواند به یک کلاس یا یک ارایه اشاره کند. بطور کلی می توان گفت که نوع یک اشاره گر در #c نمی تواند خود از **object** مشتق شده باشد. این بدین خاطر است که انجام این کار می تواند باعث اختلال در عملکرد **garbage collector** شود.
برای آنکه **garbage collector** بتواند به خوبی از عهده مدیریت حافظه بر آید نیاز به این دارد که دقیقا" بداند چه نمونه هایی (**instances**) از کلاس در **heap** ایجاد شده و در کجا واقع شده اند. اما اگر کدی شروع به کار کردن با کلاسها از طریق اشاره گرها کند، اطلاعاتی از کلاسها که در **heap** برای **garbage collector** نگهداری می شوند را براحتی از بین می برد.
داده هایی که **garbage collector** می تواند به آنها دسترسی داشته باشد همه **managed type** هستند حال آنکه اشاره گرها تنها می توانند بصورت **unmanaged type** تعریف شوند. پس **garbage collector** نمی تواند با اشاره گرها دمخور شود.

تبدیل نوع اشاره گرها:

از آنجا که اشاره گرها در واقع یک مقدار صحیح که بیانگر آدرسی از حافظه است را در خود نگه می دارند، لذا براحتی می توان آن را به یک **integer type** تبدیل کرد. با همه اینها این تبدیل نوع را نمی توان بصورت مطلق و بدون شرط انجام داد بلکه باید از **casting** استفاده کرد. به کد زیر توجه کنید:

```
int x = 10;
int* pX, pY;
pX = &x;
pY = pX;
*pY = 20;
uint y = (uint)pX;
int* pD = (int*)y;
```

می توان هر اشاره گر را به هر تایپ **integer** ، **cast** نمود. اما باید توجه داشت که هر آدرس ۴ بایت از حافظه یک سیستم ۳۲ بیتی را اشغال می کند. لذا **cast** کردن یک اشاره گر به هر تایپی به غیر از **uint** ، **long** و یا **ulong** مستعد بروز خطا می باشد. از

آنجا که نوع داده **int** بازه ای حدود ۲- بیلیون تا ۲+ بیلیون را شامل می شود و یک آدرس که همانطور که قبلا " گفته شد ۴ بایت حافظه را اشغال می کند بازه ای از صفر تا ۴ بیلیون را شامل می شود. لذا **cast** کردن یک اشاره گر به نوع داده **int** می تواند مشکل آفرین باشد.

در پردازنده های ۶۴ بیتی یک آدرس معادل ۸ بایت از حافظه را اشغال می کند. پس می توان گفت که در سیستمهای ۶۴ بیتی **cast** کردن اشاره گرها به هر تایپی به غیر از **ulong** خطای سر ریز شدن حافظه را به همراه خواهد داشت.

checked keyword در C# برای چک کردن سر ریز شدن نوع های داده صحیح **(integer data type)** بکار می رود. بدین ترتیب که با بکار بردن **checked** در هنگام سر ریز شدن متغیرهای با نوع صحیح، خطای **Overflow Exception** رخ می دهد. اما اید توجه داشت که عبارت **checked** را نمی توان برای **casting** اشاره گرها بکار برد چراکه حتی اگر **casting** با سرریزی حافظه مواجه باشد، خطای **Overflow Exception** رخ نمی دهد. بطور کلی **.NET** در **runtime** فرض را برین می گیرد که اگر شما از اشاره گر ها استفاده می کنید این خود شما هستید که می دانید چگونه باید اشاره گرها را کنترل نمایید تا با خطای سر ریز شدن حافظه مواجه نشوید.

منبع : وب سایت **www.persiadevelopers.ir**