عنوان: **مقایسه value type و reference type**
نویسنده: امیر هاشم زاده
تاریخ: ۱۳۹۲/۱۰/۲۳ ۱۰:۵۵
آدرس: www.dotnettips.info
گروه‌ها: C#, class, struct

در سی شارپ دو نوع class و struct وجود دارد که تقریباً مشابه یکدیگرند در حالیکه یکی از آنها-value type و دیگری reference-type است.

struct چیست؟

structها مشابه classها هستند با این تفاوت که structها finalizer ندارند و از ارث بری پشتیبانی نمی‌کنند. structها کاملا مشابه classها تعریف می‌شوند و در تعریف آنها از کلمه کلیدی struct استفاده می‌شود. آنها شامل فیلدها، متدها، خصوصیت‌ها نیز می‌شوند. در زیر نحوه تعریف آن را مشاهده می‌کنید:

```
struct Point
{
   private int x, y;             // private fields

   public Point (int x, int y)   // constructor
   {
         this.x = x;
         this.y = y;
   }

   public int X                  // property
   {
         get {return x;}
         set {x = value;}
   }

   public int Y
   {
         get {return y;}
         set {y = value;}
   }
}
```

reference type و value type

تفاوت دیگری که بین class و struct، از اهمیت ویژه‌ای برخوردار است آن است که **classها** reference-type و **structها** value-type هستند و در زمان اجرا با آنها متفاوت رفتار می‌شود و در ادامه به تشریح آن می‌پردازیم.
وقتی یک وهله از value-type ایجاد شود، یک فضای خالی از حافظه‌ی اصلی (RAM) برای ذخیره سازی مقدار آن تخصیص داده می‌شود. نوع‌های اصلی مانند int, float, bool و char از نوع value type هستند. در ضمن سرعت دسترسی به آنها بسیار بالاست.
ولی وقتی یک وهله از reference-type ایجاد شود، یک فضا برای object و فضایی دیگر برای اشاره‌گر به آن شیء در حافظه اصلی ذخیره می‌شود. در واقع دو فضا از حافظه برای ذخیره سازی آنها اشغال می‌شود. برای درک بهتر به مثال زیر توجه کنید:

```
Point p1 = new Point();         // Point is a *struct*
Form f1 = new Form();           // Form is a *class*
```

نکته: Point از نوع struct و Form از نوع reference است. در مورد اول، یک فضا از حافظه برای p1 تخصیص داده می‌شود و در مورد دوم، دو فضا از حافظه اصلی یکی برای ذخیره کردن اشاره‌گر f1 برای اشاره به Form object و دیگری برای ذخیره کردن Form object تخصیص داده می‌شود.

```
Form f1;                        // Allocate the reference
f1 = new Form();                // Allocate the object
```

به قطعه کد زیر دقت کنید:

```
Point p2 = p1;
Form f2 = f1;
```

همانطور که قبلاً گفته شد p2، یک نوع struct است بنابراین در مورد اول مقدار p2 یک کپی از مقدار p1 خواهد بود ولی در مورد دوم، آدرس f1 را درون f2 کپی می‌کنیم در واقع f1 و f2 به یک شیء اشاره خواهند کرد. (یک شیء با 2 اشاره گر)
در سی شارپ، پارامترها (بصورت پیش فرض) بصورت یک کپی از آنها به متدها ارسال می‌شوند، یعنی اگر پارامتر از نوع value-type باشد یک کپی از آن وهله و اگر پارامتر reference-type یک کپی از آدرس ارسال خواهد شد. برای توضیح بهتر به مثال زیر توجه کنید:

```
Point myPoint = new Point (0, 0);      // a new value-type variable
Form myForm = new Form();              // a new reference-type variable
Test (myPoint, myForm);                // Test is a method defined below

void Test (Point p, Form f)
{
      p.X = 100;                       // No effect on MyPoint since p is a copy
      f.Text = "Hello, World!";        // This will change myForm’s caption since
                                       // myForm and f point to the same object
      f = null;                        // No effect on myForm
}
```

انتساب null به f درون متد Test هیچی اثری بر روی آدرس myForm ندارد چون f، یک کپی از آدرس myForm است.
حال می‌توانیم روش پیش فرض را با افزودن کلمه کلید ref تغییر دهیم. وقتی از ref استفاده کنیم متد با پارامترهای فراخوانی کننده (caller's arguments) بصورت مستقیم در تعامل است در کد زیر می‌توانیم تصور کنیم که پارامترهای p و f متد Test همان متغیرهای myPoint و myForm است.

```
Point myPoint = new Point (0, 0);      // a new value-type variable
Form myForm = new Form();              // a new reference-type variable
Test (ref myPoint, ref myForm);        // pass myPoint and myForm by reference

void Test (ref Point p, ref Form f)
{
      p.X = 100;                       // This will change myPoint’s position
      f.Text = “Hello, World!”;        // This will change MyForm’s caption
      f = null;                        // This will nuke the myForm variable!
}
```

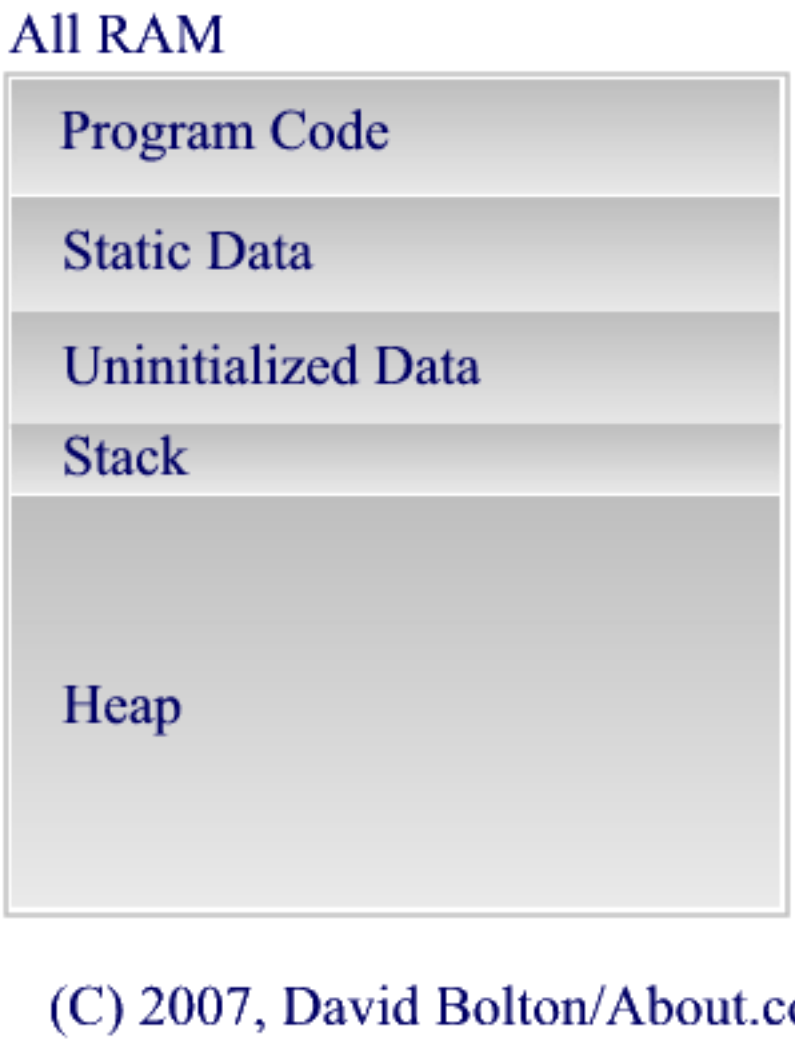
در کد بالا انتساب null به f باعث تهی شدن myForm می‌شود بدلیل اینکه متد مستقیماً به آن دسترسی داشته است.

تخصیص حافظه

CLR اشیاء را در دو قسمت ذخیره می‌کند:

stack یا پشته

heap



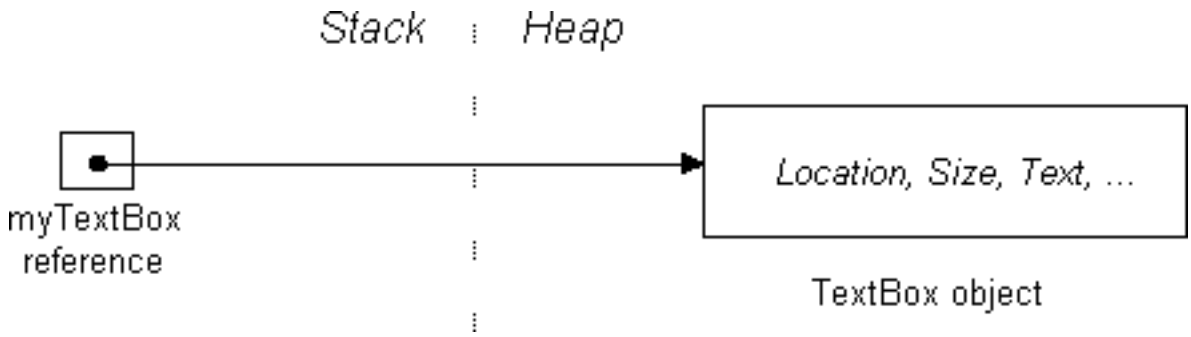
ساختار stack یا پشته first-in last-out است که دسترسی به آن سریع است. زمانی که متدی فراخوانی می‌شود، CLR پشته را نشانه گذاری می‌کند. سپس متد data را به پشته جهت اجرا push می‌کند و زمانی که اجرایش به اتمام رسید، CLR پشته را تا محل نشانه گذاری شده مرحله قبل، پاک می‌کند (pop).

ولی ساختار heap بصورت تصادفی است. یعنی اشیاء در محل‌های تصادفی قرار داده می‌شوند بهمین دلیل آنها دارای 2 سربار memory manager و garbage-collector هستند.

برای آشنایی با نحوه استفاده پشته و heap به کد زیر توجه کنید:

```
void CreateNewTextBox()
{
      TextBox myTextBox = new TextBox();            // TextBox is a class
}
```

در این متد، ما یک متغیر محلی ایجاد کرده ایم که به یک شیء اشاره می‌کند.



پشته همیشه برای ذخیره سازی موارد زیر استفاده می‌شود:
قسمت reference متغیرهای محلی و پارامترهای از نوع reference-typed (مانند myTextBox)
متغیرهای محلی و پارامترهای متد از نوع value-typed (مانند integer, bool, char, DateTime و ...)

همچنین از heap برای ذخیره سازی موارد زیر استفاده می‌شود:
محتویات شیء از نوع reference-typed
هر چیزی که قرار است در شیء از نوع reference-typed ذخیره شود.

آزادسازی حافظه در heap

در کد بالا وقتی اجرای متد CreateNewTextBox به اتمام برسد متغیر myTextBox از دید (Scope) خارج می‌شود. بنابراین از پشته نیز خارج می‌شود ولی با خارج شدن myTextBox از پشته چه اتفاقی برای TextBox object رخ خواهد داد؟! پاسخ در garbage-collector نهفته است. garbage-collector بصورت خودکار عملیات پاکسازی heap را انجام می‌دهد و اشیائی که اشاره گر معتبر ندارند را حذف می‌نماید. در حالت کلی اگر شیء از حافظه خارج شد باید منابع سایر قسمت‌های اشغال شده توسط آن هم آزاد شود، که این آزاد سازی بعهده garbage-collector است.

حال آزاد سازی برای کلاسهایی که اینترفیس IDisposable را پیاده سازی می‌کنند به دو صورت انجام می‌شود:

دستی: با فراخوانی متد Dispose میسر است.

خودکار: افزودن شیء به Net Container. مانند Form, Panel, TabPage یا UserControl. این نگهدارندها این اطمینان را به ما می‌دهند در صورتیکه آنها از حافظه خارج شدند کلیه عضوهای آن هم از حافظه خارج شوند.

برای آزادسازی دستی می‌توانیم مانند کدهای زیر عمل کنیم:

```
using (Stream s = File.Create ("myfile.txt"))
{
   ...
}
```

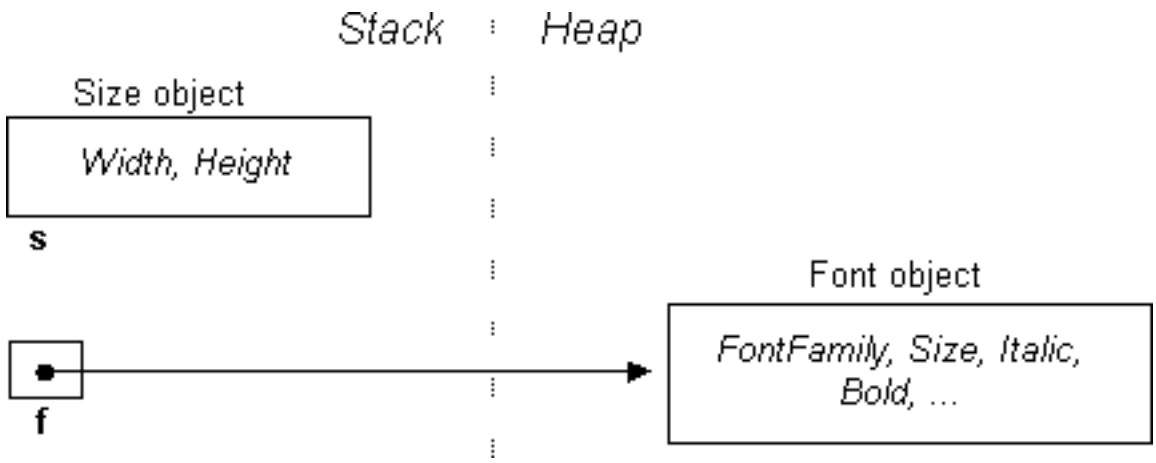
یا

```
Stream s = File.Create ("myfile.txt");
try
{
   ...
}
finally
{
   if (s != null) s.Dispose();
```

```
}
```

مثالی از Windows Forms

فرض کنید قصد داریم فونت و اندازه یک ویندوز فرم را تغییر دهیم.

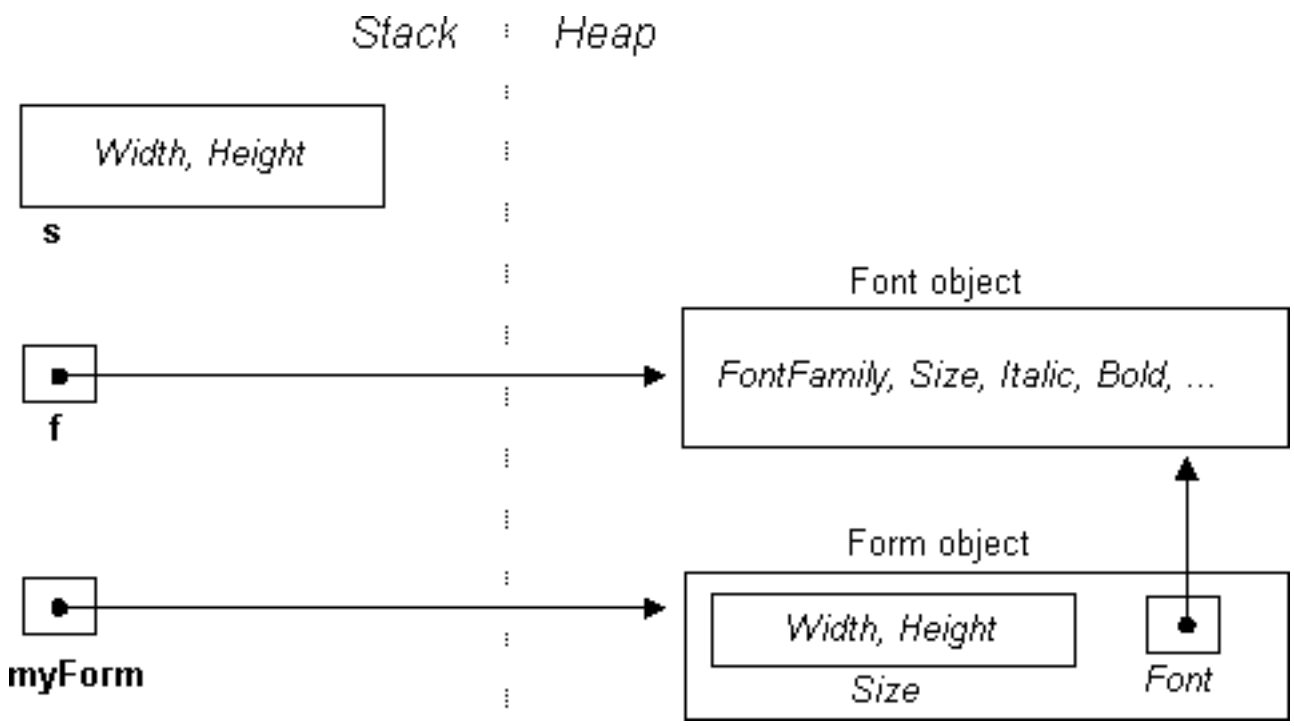



```
Size s = new Size (100, 100);          // struct = value type
Font f = new Font (“Arial”,10);        // class = reference type

Form myForm = new Form();

myForm.Size = s;
myForm.Font = f;
```

توجه کنید که ما در کد بالا از اعضای myForm استفاده کردیم نه از کلاسهای Font و Size که این دو گانگی قابل قبول است. حال به تصویر زیر که به پیاده سازی کد بالا اشاره دارد توجه کنید.



همانطور که مشاهد می‌کنید محتویات s و آدرس f را در Form object ذخیره کرده ایم که نشان می‌دهد تغییر در s برروی فرم تغییر ایجاد نمی‌کند ولی تغییر در f باعث ایجاد تغییر فرم می‌شود. Form object دو اشاره گر به Font object دارد.

In-Line Allocation (تخصیص درجا)

در قبل گفته شد برای ذخیره متغیرهای محلی از نوع value-typed از پشته استفاده می‌شود آیا شیء Size جدید هم در پشته ذخیره می‌شود؟ خیر، بدلیل اینکه آن متغیر محلی نیست و در شیء دیگر ذخیره می‌شود (در مثال بالا در یک فرم ذخیره شده است) که آن شیء هم در heap ذخیره شده است پس شیء جدید Size هم در heap ذخیر می‌شود که به این نوع ذخیره سازی In-Line گفته می‌شود.

تله (Trap)

فرض کنید کلاس Form بشکل زیر تعریف شده است:

```
class Form
{
      // Private field members
      Size size;
      Font font;

      // Public property definitions
      public Size Size
      {
            get    { return size; }
            set    { size = value; fire resizing events }
      }

      public Font Font
      {
            get    { return font;  }
            set    { font = value; }
      }
}
```

حال ما قصد داریم ارتفاع آن را دو برابر کنیم، بنابراین از کد زیر استفاده می‌کنیم:

```
myForm.ClientSize.Height = myForm.ClientSize.Height * 2;
```

ولی با خطای کامپایلر زیر روبرو می‌شویم:

```
Cannot modify the return value of 'System.Windows.Forms.Form.ClientSize' because it is not a variable
```

علت چیست؟ بدلیل اینکه myForm.ClientSize شیء Size که از نوع Struct است را بر می‌گرداند و این Struct از نوع value-typed است و این شیء یک کپی از اندازه فرم است و ما همزمان قصد دو برابر نمودن آن کپی را داریم که کامپایلر خطای بالا را نمایش می‌دهد.

برای توضیح بیشتر می‌توانید به این سوال مراجعه کنید و در تکمیل آن این لینک را هم بررسی کنید.

پس بنابراین کد بالا را به کد زیر اصلاح می‌کنیم:

```
myForm.ClientSize = new Size (myForm.ClientSize.Width, myForm.ClientSize.Height * 2);
```

برای اصلاح خطای کامپایلر، ما باید یک شیء جدیدی را برای اندازه فرم تخصیص بدهیم.