عنوان: **intern pool جدول نگهداری رشته‌ها در دات‌نت**
نویسنده: رحمت اله رضایی
تاریخ: ۱۳۹۲/۰۲/۲۹ ۲۲:۳۰
آدرس: www.dotnettips.info
برچسب‌ها: C#, Tips, CLR, string, intern, pool

کد زیر را در نظر بگیرید :

```
object text1 = "test";
object text2 = "test";

object num1 = 1;
object num2 = 1;

Console.WriteLine("text1 == text2 : " + (text1 == text2));
Console.WriteLine("num1 == num2 : " + (num1 == num2));
```

به نظر شما چه چیزی در خروجی نمایش داده میشود؟

هر چهار متغییر text1 و text2 و num1 و num2 از نوع object هستند. با اینکه مقدار text1 و text2 یکی و مقدار num1 و num2 هم یکی است، نتیجه text1==text2 برابر true است اما num1==num2 برابر false.

خطی که text2 تعریف شده است را تغییر میدهیم :

```
object text2 = "test".ToLower();
```

اینبار با این که باز مقدار text1 و text2 یکی و هر دو "test" است، اما نتیجه text1==text2 برابر false است. انتظار ما هم همین است. دو object ایجاد شده است و یکی نیستند. تنها در صورتی باید نتیجه == آنها true باشد که هر دو به یک object اشاره کنند.

اما چرا در کد اولی اینگونه نبود؟

دلیل این کار برمیگردد به رفتار دات‌نت نسبت به رشتههایی که به صورت صریح در برنامه تعریف میشوند. CLR یک جدول برای ذخیره رشتهها به نام **intern pool** برای برنامه میسازد. هر رشتهای تعریف میشود، اگر در intern pool رشتهای با همان مقدار وجود نداشته باشد، یک رشته جدید ایجاد و به جدول اضافه میشود، و اگر موجود باشد متغییر جدید فقط به آن اشاره میکند. در واقع اگر 100 جای برنامه حتی در کلاسهای مختلف، رشتههایی با مقادیر یکسان وجود داشته باشند، برای همه آنها یک نمونه وجود دارد.

بنابراین text1 و text2 در کد اولی واقعا یکی هستند و یک نمونه برای آنها ایجاد شده است.

البته چند نکته در اینجا هست :
اگر text1 و text2 به صورت string تعریف شوند، نتیجه text1==text2 در هر دو حالت فوق برابر true است. چون عملگر == در کلاس string یکبار دیگر overload شده است:

```
public sealed class String : ...
{
    ...
    public static bool operator ==(string a, string b)
    {
```

```
      return string.Equals(a, b);
    }
   ...
}
```

این که کدام یک از overloadها اجرا شوند (کلاس پایه، کلاس اصلی، ...) به نوع دو متغییر اطراف == بستگی دارد. مثلا در کد زیر :

```
string text1 = "test";
string text2 = "test".ToLower();

Console.WriteLine("text1 == text2 (string) : " + (text1 == text2));
Console.WriteLine("text1 == text2 (object) : " + ((object)text1 == (object)text2));
```

اولین نتیجه true و دومی false است. چون در اولی عملگر == تعریف شده در کلاس string مورد استفاده قرار می‌گیرد اما در دومی عملگر == تعریف شده در کلاس object.

اگر دقت نشود این رفتار مشکلزا میشود. مثلا حالتی را در نظر بگیرید که text1 ورودی کاربر است و text2 از بانک اطلاعاتی خوانده شده است و با اینکه مقادیر یکسان دارند نتیجه == آنها false است. اگر تعریف عملگرها در کلاس object به صورت virtual بود و در کلاس‌های دیگر override می‌شد، این تغییر نوع‌ها تاثیری نداشت. اما عملگرها به صورت static تعریف می‌شوند و امکان override شدن ندارند. به همین خاطر کلاس object متدی به اسم Equals در اختیار گذاشته که کلاس‌ها آنرا override می‌کنند و معمولا از این متد برای سنجش برابری دو کلاس استفاده می‌شود :

```
object text1 = "test";
object text2 = "test".ToLower();

Console.WriteLine("text1 Equals text2 : " + text1.Equals(text2));
Console.WriteLine("text1 Equals text2 : " + object.Equals(text1, text2));
```

البته یادآور می‌شوم که **فقط رشته‌هایی که به صورت صریح در برنامه تعریف شده‌اند، در intern pool قرار می‌گیرند و این فهرست شامل رشته‌هایی که از فایل یا بانک اطلاعاتی خوانده می‌شوند یا در برنامه تولید می‌شوند، نیست.** این کار منطقی است وگرنه حافظه زیادی مصرف خواهد شد.

با استفاده از متد string.Intern می‌توان یک رشته را که در intern pool وجود ندارد، به فهرست آن افزود. اگر رشته در intern pool وجود داشته باشد، reference آنرا بر می‌گرداند در غیر اینصورت یک reference به رشته جدید به intern pool اضافه می‌کند و آنرا برمی‌گرداند.

یک مورد استفاده آن هنگام lock روی رشته‌هاست. برای مثال در کد زیر DeviceId یک رشته است که از بانک اطلاعاتی خوانده می‌شود و باعث می‌شود که چند job همزمان به یک دستگاه وصل نشوند :

```
lock (job.DeviceId)
{
    job.Execute();
}
```

اگر یک job با DeviceId برابر COM1 در حال اجرا باشد، این lock جلوی اجرای همزمان job دیگری با همین DeviceId را

نمی‌گیرد. زیرا هر چند مقدار DeviceId دو job یکی است ولی به یک نمونه اشاره نمی‌کنند.

می‌توان lock را اینگونه اصلاح کرد :

```
lock (string.Intern(job.DeviceId))
{
    job.Execute();
}
```

نظرات خوانندگان

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۲/۳۰ ۰:۳۵

ممنون. البته شرایط کد خودتون رو کامل اینجا قرار ندادید ولی در حالت کلی توصیه میشه که برای استفاده از lock یک شیء private object در سطح کلاس تعریف بشه و از اون استفاده بشه تا حالت‌های دیگر .

نویسنده: رحمت اله رضایی
تاریخ: ۱۳۹۲/۰۲/۳۱ ۹:۵۶

- البته این فقط یک مثال بود برای درک متد string.Intern .
- چگونگی شی معرفی شده به lock هم بسته به شرایط ممکن است متفاوت باشد. ممکن است یک private object در سطح همان کلاسی که lock در آن استفاده می‌شود، جوابگو باشد. اما در شرایط دیگری ممکن است اینگونه نباشد. مانند مثال فوق.