یاد خدا آرام بخش قلبهاست....

موضوع :Delegate ها

ارائه دهنده :پدیده محمدی

نام استاد : مهندس محمد سلیمی

WWW.SALIMITEACH.COM

فهرست :

- ✓ مقدمه
- ✓ درك اينكه يك delegate چيست؟
- ✓ اعلان و پيادەسازي delegate ها
- ✓ درك اينكه يك رخداد يا يك event چيست؟
- ✓ نكات كليدي درباره رخدادها و delegate ها

✓ مقدمه

در گذشته، پس از اجراي يك برنامه، برنامه مراحل اجراي خود را مرحله به مرحله اجرا مي‌نمود تا به پايان برسد. در صورتيكه نياز به ارتباط و تراكنش با كاربر نيز وجود داشت، اين امر محدود و بسيار كنترل شده صورت مي‌گرفت و معمولاً ارتباط كاربر با برنامه تنها پر كردن و يا وارد كردن اطلاعات خاصي در فيلدهايي مشخص بود.

امروزه با پيشرفت كامپيوتر و گسترش تكنولوژيهاي برنامه نويسي و با ظهور رابطهاي كاربر گرافيكي (GUI) ارتباط بين كاربر و برنامه بسيار گسترش يافته و ديگر اين ارتباط محدود به پر كردن يكسري فيلد نيست، بلكه انواع عمليات از سوي كاربر قابل انجام است. انتخاب گزينه‌اي خاص در يك منو، كليك كردن بر روي دكمه‌ها براي انجام عملياتي خاص و ... . رهيافتي كه امروزه در برنامه‌نويسي مورد استفاده است، تحت عنوان "برنامه‌نويسي بر پايه رخدادها" (Event-Based Programming) شناخته مي‌شود. در اين رهيافت برنامه همواره منتظر انجام عملي از سوي كاربر مي‌ماند و پس از انجام عملي خاص، رخداد مربوط به آن را اجرا مي‌نمايد. هر عمل كاربر باعث اجراي رخدادي مي‌شود. در اين ميان برخي از رخدادها بدون انجام عملي خاص از سوي كاربر اجرا مي‌شوند، همانند رخدادهاي مربوط به ساعت سيستم كه مرتباً در حال اجرا هستند.

رخدادها (Events) بيان اين مفهوم هستند كه در صورت اتفاق افتادن عملي در برنامه، كاري بايد صورت گيرد. در زبان C# مفاهيم Event و Delegate دو مفهوم بسيار وابسته به يكديگر هستند و با يكديگر در تعامل مي‌باشند. براي مثال، مواجهه با رخدادها و انجام عمل مورد نظر در هنگام اتفاق افتادن يك رخداد، نياز به يك event handler دارد تا در زمان بروز رخداد، بتوان به آن مراجعه نمود. Event handler ها در C# معمولاً با delegate ها ساخته مي‌شوند.

از delegate ، مي‌توان به عنوان يك Callback ياد نمود، بدين معنا كه يك كلاس مي‌تواند به كلاسي ديگر بگويد : "اين عمل خاص را انجام بده و هنگاميكه عمليات را انجام دادي منرا نيز مطلع كن". با استفاده از delegate ها، همچنين مي‌توان متدهايي تعريف نمود كه تنها در زمان اجرا قابل دسترسي باشند.

✓ درك اينكه يك delegate چيست؟

Delegate ها، يكي ديگر از انواع مرجعي زبان C# هستند كه با استفاده از آنها مي‌توانيد مرجعي به يك متد داشته باشيد، بدين معنا كه delegate ها، آدرس متدي خاص را در خود نگه ميدارند. در صورتيكه قبلاً با زبان C برنامه‌نويسي كرده‌ايد، حتماً با اين مفهوم آشنايي داريد. در زبان C اين مفهوم با اشاره‌گرها (pointer) بيان مي‌شود. اما براي افرادي كه با زبانهاي ديگري برنامه‌نويسي مي‌كرده‌اند و با اين مفهوم مانوس نيستند، شايد اين سوال مطرح شود كه چه نيازي به داشتن آدرس يك متد وجود دارد. براي پاسخ به اين سوال اندكي بايد تامل نماييد.

بطور كلي مي‌توان گفت كه delegate نوعي است شبيه به متد و همانند آن نيز رفتار مي‌كند. در حقيقت delegate انتزاعي (Abstraction) از يك متد است. در برنامه‌نويسي ممكن به شرايطي برخورد كرده باشيد كه در آنها مي‌خواهيد عمل خاصي را انجام دهيد اما دقيقاً نمي‌دانيد كه بايد چه متد يا شيء‌اي را براي انجام آن عمل خاص مورد استفاده قرار دهيد. در برنامه‌هاي تحت ويندوز اين گونه مسائل مشهودتر هستند. براي مثال تصور كنيد در برنامه شما، دكمه‌اي قرار دارد كه پس از فشار دادن اين دكمه توسط كاربر شيء‌اي يا متدي بايد فراخواني شود تا عمل مورد نظر شما بر روي آن انجام گيرد. مي‌توان بجاي اتصال اين دكمه به شيء يا متد خاص، آنرا به يك delegate مرتبط نمود و سپس آن delegate را به متد يا شيء خاصي در هنگام اجراي برنامه متصل نمود.

ابتدا، به نحوه استفاده از متدها توجه نماييد. معمولاً، براي حل مسايل خود الگوريتم‌هايي طراحي مي‌نائيم كه اين الگوريتمهاي كارهاي خاصي را با استفاده از متدها انجام مي‌دهد، ابتدا متغيرهايي مقدار دهي شده و سپس متدي جهت پردازش آنها فراخواني مي‌گردد. حال در نظر بگيريد كه به الگوريتمي نياز داريد كه بسيار قابل انعطاف و قابل استفاده مجدد (reusable) باشد و همچنين در شرايط مختلف قابليت‌هاي مورد نظر را در اختيار شما قرار دهد.

Delegateیک اشاره گر به تابع است(function pointer)

تعریف بالا با توجه به اینکه غلط نیست، اما کامل هم نیست، در حقیقت delegate آبجکتی است که رفرنسی از یک متد یا مجموعه ای از متدها رو در خودش نگه میداره. اما این مجموعه متدها بایستی از یک سری قوانین تبعیت کنند تا بتونن به لیست متدهایی که اون delegate بهشون اشاره میکنه اضافه بشن، به عبارت دیگه برای اینکه متدی به لیست Reference Methodهای یک delegate اضافه بشه باید با تعریف اون delegate هماهنگ باشه، به این هماهنگی و تطابق اصطلاحا امضا (Signature) میگن، پس مجوز ورود یک متد به لیست Reference Method یک delegateاینه که امضای اون متد با تعریف delegateمورد نظر یکی باشه، به این ترتیب که:

1. نوع بازگشتی متد مورد نظر برابر با نوع بازگشتی مشخص شده در تعریف delegateباشه.

2. نوع و تعداد پارامترهای ورودی متد مورد نظر برابر با نوع و تعداد پارامترهای ورودی مشخص شده در تعریف delegate باشه.

لازم به ذکر است که به لیست Reference Methods مربوط به یک delegate اصطلاحا Invocation List گفته میشود.

مثال
به تعریف delegate زیر نگاه کنید:

```
public delegate void myDelegate(string argument);
```

طبق تعریف delegate مورد نظر، نام delegate ما myDelegate است و متدهایی را میتواند به Invocation List خود اضافه کند که:

1. هیچ مقداری بر نگردانند.
2. یک پارامتر از نوع string بگیرند.

به این متد نگاه کنید:

```
private void myMethod(string msg)
{
     // method body
}
```

همانطور که در کد بالا میبینید، امضای متد myMethod با تعریف myDelegate مطابقت دارد، پس این متد میتواند به Invocation List آبجکت myDelegate ما اضافه شود:

```
myDelegate del = new myDelegate(myMethod);
```

متد myMethod اولین متدی است که به Invocation List آبجکت myDelegate یعنی del اضافه میشود، برای اضافه کردن متدهای بیشتر بایستی به شیوه ی زیر عمل کنیم:

```
del += new myDelegate(myMethod2);
```

همچنین برای حذف یک متد از Invocation List یک آبجکت delegate میبایست به شیوه ی زیر عمل نماییم:

```
del -= new myDelegate(myMethod2);
```

حال، این متد را در نظر بگیرید:

```
private void myMethod3()
{
            //  method body
                                                                    }
```

آیا این متد میتواند به Invocation List آبجکت del اضافه شود ؟ پاسخ منفی است، به دلیل اینکه امضای آن با امضای delegateاعلان شده ی ما مطابقت ندارد (هیچ پارامتری دریافت نمیکند.)
حال اگر آبجکت delegate مورد نظر را Invoke کنیم، تمام متدهایی که در Invocation List آن موجود هستند اجرا میشوند، نحوه ی Invokeکردن یک آبجکتdelegate مانند فراخوانی یک متد و ارسال پارامتر (در صورت نیاز) به آن میباشد:

```
del("my message value");
```

خب، حالا هرچی یاد گرفتیم رو میخواهیم در قالب یک پروژه ی Console Application پیاده سازی نماییم، پس یک پروژه از نوع Console Application بسازید و کد زیر رو در آن وارد نمایید

```
class Program
{
        delegate void myDelegate(string argument);
        static void Main(string[] args)
        {
            myDelegate del = new myDelegate(myMethod);
            del += new myDelegate(myMethod2);
            del("sample string value");

            Console.Read();
        }

        private static void myMethod(string msg)
        {
            Console.WriteLine(msg+" from myMethod");
        }

        private static void myMethod2(string msg)
        {
            Console.WriteLine(msg + " from myMethod2");
       }
```

خب، با Invoke شدن آبجکت del هر دو متد myMethod و myMethod2 اجرا میشوند و خروجی آنها نمایش داده میشوند.

چرا در Event ها از Delegate استفاده میکنیم ؟
اگرdelegate ، فراخوانی ساده ی یک متد به صورت غیر مستقیم است، پس چه نیازی هست که از delegate استفاده کنیم ؟
پاسخ این است، کامپوننتهایی که شامل رویداد های مختلفی هستند را در نظر بگیرید، در زمان تولید نرم افزار شما کنترلی را به فرمتون اضافه میکنید و در صورت لزوم، رویدادهای مختلف آن را مدیریت میکنید، زمانی که شما رویدادی رو Handle میکنید، در حقیقت متدی رو در Invocation List آبجکت delegate درون کنترلتون اضافه یا **Register**کردید، متدی شبیه متد زیر:

```
private void anyMethod(object sender, EventArgs e)
{
            //  method body
}
```

کنترل شما، حقیقتا هیچ اطلاعی در مورد متد مورد نظر نداره، فقط میدونه که این متد با امضای تعریف Delegate درونی خودش مطابقت داره و به Invocation Listاش اضافه شده و هر زمان که آبجکت delegate مورد نظر Invokeشد، اونو اجرا میکنه، در حقیقت شما با ایجاد یک متد Event Handler در سورس کدتون، اونو دارید به delegateدرون کنترل مورد نظر پیوند میدید)با توجه به تطابق امضای متد با (delegate و آبجکت delegate مورد نظر بدون اینکه اطلاعی در مورد متدهای رجیستر شده در Invocation List اش داشته باشه، اونها رو به صورت غیر مستقیم در زمان Invoke شدن خودش اجرا میکنه .این مهمترین دلیل استفاده از delegate ها در مدیریت رویداد میتونه باشه.

ساختار درونی: Delegate

زمانی که شما delegate ای رو اعلان میکنید، کامپایلر #C با توجه به اعلان delegateشما، کلاسی رو به assemblyخروجی اضافه میکنه که نام کلاس مورد نظر، برابر با نام delegate شما و متدهای کلاس مورد نظر دارای امضایی مطابق delegate شما میباشند، این کلاس از کلاسی به نام System.MulticastDelegateمشتق میشه و دارای متدهایی است که امضای اونا با delegateشما مطابقت داره (متدهایی با نام Invoke,BeginInvoke و EndInvoke)که متد Invoke برای Synchronous Invocation و BeginInvoke و EndInvokeهم برای Asynchronous Invocationبه کار برده میشوند. لازم به ذکر است که شما نمی توانید خودتان کلاسی رو از System.MulticastDelegateمشتق کنید، تنها کامپایلر #C مجاز به چنین کاری می باشد، به عبارت دیگه با اعلان

یک delegate شما به کامپایلر C# میگویید که کلاسی رو طبق delegateتعریف شده ی شما از System.MulticastDelegateمشتق کنه.

اما علت نامی که برای System.MulticastDelegate در نظر گرفته شده (Multicast)اینه که، delegateمیتونه ارجاع بیشتر از یک متد رو در خودش نگه داره، و در زمان Invoke شدن، تمام رفرنس های متدهایی که در Invocation Listاش وجود دارد رو فراخوانی کنه.

✓ رخدادها (Events)

در برنامه‌هاي Console ، برنامه منتظر ورود اطلاعات يا دستوراتي از سوي كاربر مي‌ماند و با استفاده از اين اطلاعات كار مورد نظر را انجام مي‌دهند. اين روش برقراري ارتباط با كاربر، روشي ناپايدار و غير قابل انعطاف است. در مقابل برنامه‌هاي Console، برنامه‌هاي مدرن وجود دارند كه با استفاده از GUI با كاربر در ارتباطند و بر پايه رخدادها بنا شده‌اند (Event-Based)، بدين معنا كه رخدادي (منظور از رخداد اتفاقي است كه در سيستم يا محيط برنامه صورت ميگيرد.) در سيستم روي مي‌دهد و بر اساس اين رخداد عملي در سيستم انجام مي‌شود. در برنامه‌هاي تحت ويندوز، نيازي به استفاده از حلقه‌هاي متعدد جهت منتظر ماندن براي ورودي از كاربر نيست، بلكه با استفاده از رخدادها، تراكنش بين سيستم و كاربر كنترل مي‌شود.

يك event در زبان C#، عضوي از كلاس است، كه در صورت بروز رخداد خاصي، فعال مي‌شود و عملي را انجام مي‌دهد. معمولاً براي فعال شده event از دو عبارت fires و raised استفاده مي‌شود. هر متدي كه بخواهد، ميتواند در ليست رخداد ثبت شده و به محض اتفاق افتادن آن رخداد، از آن مطلع گردد.

Delegate و رخدادها در كنار يكديگر كار مي‌كنند تا قابليت‌هاي يك برنامه را افزايش دهند. اين پروسه با شروع يك كلاس كه يك رخداد را تعريف مي‌كند، آغاز مي‌شود. هر كلاسي، كه اين رخداد را درون خود داشته باشد، در آن رخداد ثبت شده است و مي‌تواند متدي را به آن رخداد تخصيص دهد. اين عمل با استفاده از delegate ها صورت مي‌پذيرد، بدين معني كه delegate متدي را كه براي رخداد ثبت مي‌شود را تعيين مي‌نمايد. Delegate ها مي‌توانند هر يك از delegate هاي از پيش تعريف شدة Net. و يا هر delegate ي باشند كه توسط كاربر تعريف شده است. بطور كلي، delegate ي را به رخدادي تخصيص مي‌دهيم تا متدي را كه بهنگام روي دادن رخداد فراخواني مي‌شود، معين گردد.

✓ نكاتي چند درباره delegate ها و event ها

- Delegate ها بطور ضمني از System.Delegate ارثبري مي‌كنند. Delegate حاوي متدها، property ها و عملگرهايي است كه مي‌توان آنها را بعنوان پارامتر به متدهاي ديگر ارسال نمود. همچنين به دليل اينكه System.Delegate بخشي از Net Framework. است، از اينرو delegate هاي ايجاد شده در C# را مي‌توان در زبانهاي ديگري نظير Visual Basic.Net نيز استفاده نمود.

- هنگام اعلان پارامترها براي delegate، حتماً بايد براي آنها نام در نظر بگيريد و فقط به مشخص كردن نوع پارامترها بسنده نكنيد.

- رخدادها عناصر بسیار مفید و پر استفاده‌اي هستند که با بکارگیري delegate ها بسیار قدرتمند ظاهر مي‌شوند. بدست آوردن مهارت در ایجاد و استفاده از آنها نیاز به تمرین و تفکر بسیار دارد.

البته مبحث Delegate و همینطور موارد استفاده ی آنها بسیار وسیع تر از حیطه ی این مقاله س، مثلا شما میتونید delegate هایی که دارای امضاهای مشابه هستند رو با هم جمع کنید و در یک delegate جدید ذخیره نمایید، حاصل delegate ای خواهد شد که Invocation List آن برابر با مجموع Invocation List های delegateهای جمع شده است، میتونید از delegate ها در برنامه های MultiThread استفاده کنید و یا اونها رو در Asynchronous Programming به کار ببرید و...

مثال 2

```
class Employee
    {

        static void Main2()
        {
            Employee oEmployee = new Employee("Ali Vali", 33);
            MyDelegate DelegateInstance = new MyDelegate(oEmployee.DoIt);

            DelegateInstance(5);
        }


        public int Age;
        public string FullName;
        public Employee(string fullName, int age)
        {
            Age = age;
            FullName = fullName;
        }
        public void DoIt(int n)
        {
            Console.WriteLine( + FullName + ",  did it " + n + " times .");
        }
```

✓ **منابع و مآخذ**

http://cslearning.tripod.com/cs/compCs_3_1.htm

http://msdn.microsoft.com/en-us/library/ms173171%28VS.80%29.aspx

http://www.akadia.com/services/dotnet_delegates_and_events.html