عنوان: **مروری بر کاربردهای Action و Func - قسمت سوم**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۵/۲۹ ۲۰:۱۹
آدرس: www.dotnettips.info
گروه‌ها: C#, Refactoring

در ادامه مثال سوم قسمت قبل ، در مورد حذف کدهای تکراری توسط Action و Func، در این قسمت به یک مثال نسبتا پرکاربرد دیگر آن جهت ساده سازی try/catch/finally اشاره خواهد شد.
احتمالا هزاران بار در کدهای خود چنین قطعه کدی را تکرار کرده‌اید:

```
try {
      // code
} catch(Exception ex) {
      // do something
}
```

این مورد را نیز می‌توان توسط Actionها کپسوله کرد و پیاده سازی قسمت بدنه try آن‌را به فراخوان واگذار نمود:

```
void Execute(Action action) {
    try {
       action();
    } catch(Exception ex) {
       // log errors
    }
}
```

و برای نمونه جهت استفاده از آن خواهیم داشت:

```
Execute(() => {open a file});
```

یا اگر عمل انجام شده باید خروجی خاصی را بازگرداند (برخلاف یک Action که خروجی از آن انتظار نمی‌رود)، می‌توان طراحی متد Execute را با Func انجام داد:

```
public static class SafeExecutor
{
    public static T Execute<T>(Func<T> operation)
    {
        try
        {
            return operation();
        }
        catch (Exception ex)
        {
            // Log Exception
        }
        return default(T);
    }
}
```

در این حالت فراخوانی متد Execute به نحو زیر خواهد بود:

```
var data = SafeExecutor.Execute<string>(() =>
{
    // do something
    return "result";
});
```

و اگر در این بین استثنایی رخ دهد، علاوه بر ثبت جزئیات خطای رخ داده شده، نال را بازگشت خواهد داد.

از همین دست می‌توان به کپسوله سازی منطق «سعی مجدد» در انجام کاری اشاره کرد:

```
public static class RetryHelper
{
   public static void RetryOperation(Action action, int numRetries, int retryTimeout)
   {
       if( action == null )
           throw new ArgumentNullException("action");

       do
       {
          try {  action(); return;  }
          catch
          {
              if( numRetries <= 0 ) throw;
              else
                 Thread.Sleep( retryTimeout );
           }
       } while( numRetries-- > 0 );
   }
}
```

برای مثال فرض کنید برنامه قرار است اطلاعاتی را از وب دریافت کند. ممکن است در سعی اول آن، خطای اتصال یا در دسترس نبودن لحظه‌ای سایت رخ دهد. در اینجا نیاز خواهد بود تا این عملیات چندین بار تکرار شود؛ که نمونه‌ای از آن‌را در ذیل ملاحظه می‌کنید:

```
RetryHelper.RetryOperation(() => SomeFunction(), 3, 1000);
```

نظرات خوانندگان

نویسنده: مجتبی صحرائی
تاریخ: ۱۳۹۱/۰۵/۲۹ ۲۱:۶

بسیار زیبا

نویسنده: امیر
تاریخ: ۱۳۹۱/۰۵/۲۹ ۲۲:۴۶

واقعا بحث زیبا و پرکاربردی است.

نویسنده: رضا
تاریخ: ۱۳۹۱/۰۵/۳۰ ۲۲:۲۸

واقعاً مبحث فوق العاده ای هست و شما هم عالی توضیح میدید. حذف کدهای تکراری واقعاً کمک کننده هستش.
ممنون.

نویسنده: Hamid NCH
تاریخ: ۱۳۹۲/۰۹/۱۳ ۱۵:۵۱

در مورد این توضیح میدین.خیلی ممنون

```
return default (T);
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۹/۱۳ ۱۷:۳۳

گاهی از اوقات حین کار با نوع‌های جنریک نیاز دارید که مثلا null بازگشت بدید. در این حالت کامپایلر شما را با خطای Cannot convert null to type parameter T متوقف می‌کند. به همین جهت مرسوم است در این حالت از default T استفاده شود که مقدار پیش فرض نوع را برمی‌گرداند. اگر reference type باشد (مثل کلاس‌ها) این مقدار پیش فرض null خواهد بود؛ اگر value type باشد مانند int صفر بازگشت داده می‌شود.