عنوان: **C# 6 - Exception Filtering**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۷/۱۵ ۱۵:۲۵
آدرس: www.dotnettips.info
گروه‌ها: C#, C# 6.0

سی‌شارپ نیز مانند بسیاری از زبان‌های شیءگرای دیگر، امکان فیلتر کردن استثناءها را بر اساس نوع آن‌ها، دارا است. برای مثال:

```
try
{
    // some code to check  ...
}
catch (InvalidOperationException ex)
{
    // do your handling for invalid operation ...
}
catch (IOException ex)
{
    // do your handling for IO error ...
}
```

در اینجا می‌توان بر اساس نوع استثنای مدنظر، چندین catch را نوشت و مدیریت کرد. اما گاهی از اوقات شاید بهتر باشد بجای مدیریت کلی یک نوع از استثناءها، فقط نوعی خاص را صرفا بر اساس شرایطی مشخص، مدیریت کرد. این قابلیت، تحت عنوان Exception Filtering به C# 6 اضافه شده‌است و شکل کلی آن به صورت ذیل است:

```
catch (SomeException ex) when (someConditionIsMet)
{
    // Your handler logic
}
```

در این حالت ابتدا نوع استثناء بررسی می‌شود و سپس شرطی که در قسمت when ذکر شده‌است. اگر هر دو با هم برقرار بودند، آنگاه این استثنای خاص مدیریت خواهد شد؛ در غیر اینصورت، از مدیریت این نوع استثناء صرفنظر می‌گردد. این قابلیت، از ابتدای ارائه‌ی CLR وجود داشته‌است، اما C#6 تازه شروع به استفاده‌ی از آن کرده‌است (و VB.NET از چند نگارش قبل).

علاوه بر این در اینجا می‌توان چندین بدنه‌ی catch مجزا را به ازای یک نوع استثنای مشخص به همراه whenهای متفاوتی نیز تعریف کرد و از این لحاظ محدودیتی وجود ندارد. فقط در این حالت باید به تقدم و تاخرها دقت داشت. برای نمونه در مثال ذیل، ترکیب چندین شرط متفاوت را بر اساس یک نوع مشخص استثناء، مشاهده می‌کنید. در اینجا اگر برای نمونه شرط ذکر شده‌ی در قسمت when مربوط به catch اولی صادق باشد، همینجا کار خاتمه می‌یابد و سایر catchها بررسی نمی‌شوند:

```
catch (SomeDependencyException ex) when (condition1 && condition2)
{

}
catch (SomeDependencyException ex) when (condition1)
{

}
catch (SomeDependencyException ex)
{

}
```

مورد آخر، حالت catch all را دارد و در صورت شکست دو catch قبلی اجرا می‌شود. اما باید دقت داشت که اگر این catch all بدون شرط و بدون قسمت when را در ابتدا ذکر کنیم، دیگر کار به بررسی سایر catchهای این نوع استثنای خاص نخواهد رسید:

```
catch (SomeDependencyException ex)
{

}
catch (SomeDependencyException ex) when (condition1 && condition2)
```

```
{

}
catch (SomeDependencyException ex) when (condition1)
{

}
```

در مثال فوق هیچگاه دو catch تعریف شده‌ی پس از catch all اولی اجرا نمی‌شوند.

لاگ کردن استثناءها در C# 6 بدون مدیریت آن‌ها

به مثال ذیل دقت کنید:

```
try
{
    DoSomethingThatMightFail(s);
}
catch (Exception ex) when (Log(ex, "An error occurred"))
{
    // this catch block will never be reached
}

...

static bool Log(Exception ex, string message, params object[] args)
{
   Debug.Print(message, args);
   return false;
}
```

در قسمت when می‌توان هر متدی که true یا false را برگرداند، فراخوانی کرد. در این مثال، متدی تعریف شده‌است که false بر می‌گرداند. یعنی این استثناء کلی از نوع Exception هرچند به ظاهر دارای قسمت when است و مدیریت شده‌است، اما چون خروجی متد Log قسمت when آن مساوی false است، مدیریت نخواهد شد. یعنی در اینجا می‌توان بدون مدیریت یک استثناء، اطلاعات کامل آن‌را لاگ کرد!

تفاوت C# 6 - Exception Filtering با if/else نوشتن در بدنه‌ی catch چیست؟

تا اینجا به این نتیجه رسیدیم که کدهای if/else دار داخل بدنه‌ی catch کدهای قدیمی را مانند کد ذیل:

```
try
{
    var request = WebRequest.Create("http://www.google.coom/");
    var response = request.GetResponse();
}
catch (WebException we)
{
    if (we.Status == WebExceptionStatus.NameResolutionFailure)
    {
        //handle DNS error
        return;
    }
    if (we.Status == WebExceptionStatus.ConnectFailure)
    {
        //handle connection error
        return;
    }

    throw;
}
```

می‌توان به شکل جدید C# 6 به همراه when نوشت و تبدیل کرد:

```
try
```

```
{
    var request = WebRequest.Create("http://www.google.coom/");
    var response = request.GetResponse();
}
catch (WebException we) when (we.Status == WebExceptionStatus.NameResolutionFailure)
{
    //Handle NameResolutionFailure Separately
}
catch (WebException we) when (we.Status == WebExceptionStatus.ConnectFailure)
{
    //Handle ConnectFailure Separately
}
```

اما باید دقت داشت که تفاوت مهم قطعه کد دوم، در مباحث Stack unwinding است. در مثال اولی که if/else داخل بدنه‌ی catch نوشته شده‌است، اطلاعات local محل فراخوانی متدی را که سبب بروز استثناء شده‌است، از دست خواهیم داد؛ اما در مثال دوم خیر.
به این معنا که exception filters سبب Stack unwinding نمی‌شوند. با هربار ورود به بدنه‌ی catch، اصطلاحا عملیات Stack unwinding صورت می‌گیرد. یعنی اطلاعات stack مربوط به متدهای پیش از فراخوانی متدی که سبب بروز استثناء شده‌است، از بین می‌روند. به این ترتیب تشخیص مقادیر متغیرهایی که سبب بروز این استثناء شده‌اند نیز میسر نخواهد بود و دیگر نمی‌توان با قطعیت عنوان کرد که چه مقادیری و چه اطلاعاتی سبب بروز این مشکل شده‌اند. اما در حالت exception filters در قسمت when آن هنوز وارد بدنه‌ی catch نشده‌ایم. در اینجا دسترسی کاملی به اطلاعات stack جاری و مقادیر متغیرهای محلی که سبب بروز این استثناء شده‌اند وجود دارد.

تفاوت stack با stack trace چیست؟ stack قطعه‌ای از حافظه‌است که اطلاعاتی در مورد نحوه‌ی فراخوانی متدها، آدرس بازگشتی آن‌ها، آرگومان و همچنین متغیرهای محلی آن‌ها را دارا است. اما stack trace تنها یک رشته‌است و بیانگر نام متدهایی است که هم اکنون بر روی stack قرار دارند. احتمالا پیشتر خوانده بودید که فراخوانی throw داخل بدنه‌ی catch سبب حفظ stack trace می‌شود و اگر throw ex صورت گیرد، این اطلاعات از دست می‌روند و بازنویسی می‌شوند. اما در C# 6 امکان حفظ کل اطلاعات stack به همراه exception filtering میسر شده‌است.