عنوان: **فلسفه وجودی بخش finally در try catch چیست؟**
نویسنده: فانوس
تاریخ: ۱۳۹۲/۰۷/۰۹ ۱۱:۱۰
آدرس: www.dotnettips.info
برچسب‌ها: Exception, finally, exception handling, استثناء

حتما شما هم متوجه شدید که وقتی رخداد یک استثناء را با استفاده از try و catch کنترل می‌کنیم، هر چیزی که بعد از بسته شدن تگ catch بنویسیم، در هر صورت اجرا می‌شود.

```
try {
     int i=0;
     string s = "hello";
     i = Convert.ToInt32(s);
} catch (Exception ex)
{
     Console.WriteLine("Error");
}
Console.WriteLine("I am here!");
```

پس فلسفه استفاده از بخش finally چیست؟

در قسمت finally منابع تخصیص داده شده در try را آزاد می‌کنیم. کد موجود در این قسمت به هر روی اجرا می‌شود چه استثناء رخ دهد چه ندهد. البته اگر استثناء رخ داده شده در لیست استثناء هایی که برای آنها catch انجام دادیم نباشد، قسمت finally هم عمل نخواهد کرد مگر اینکه از catch به صورت سراسری استفاده کنیم.

اما مهمترین مزیتی که finally ایجاد می‌کند در این است که حتی اگر در قسمت try با استفاده از دستوراتی مثل return یا break یا continue از ادامه کد منصرف شویم و مثلا مقداری برگردانیم، چه خطا رخ دهد یا ندهد کد موجود در finally اجرا می‌شود در حالی که کد نوشته شده بعد از try catch finally فقط در صورتی اجرا می‌شود که به طور منطقی اجرای برنامه به آن نقطه برسد. اجازه بدهید با یک مثال توضیح دهم. اگر کد زیر را اجرا کنیم:

```
public static int GetMyInt()
 {
     try {
          for (int i=10;i>=0;i--)
             Console.WriteLine(10/i);
         return 1;
     } catch
     {
         Console.WriteLine("Error!");
     }
     finally {
         Console.WriteLine("ok");
     }
     Console.WriteLine("can you reach here?");
     return -1;
 }
```

برنامه خطای تقسیم بر صفر می‌دهد اما با توجه به کدی که نوشتیم، عدد -1 به خروجی خواهد رفت. در عین حال عبارت ok و can you reach here در خروجی چاپ شده است. اما حال اگر مشکل تقسیم بر صفر را حل کنیم، آیا باز هم عبارت can you reach here در خروجی چاپ خواهد شد؟

```
public static int GetMyInt()
 {
     try {
          for (int i=10;i>=1;i--)
             Console.WriteLine(10/i);
         return 1;
     } catch
     {
         Console.WriteLine("Error!");
     }
     finally {
         Console.WriteLine("ok");
     }
```

```
    Console.WriteLine("can you reach here?");
    return -1;
}
```

مشاهده می‌کنید که مقدار 1 برگردانده می‌شود و عبارت can you reach here در خروجی چاپ نمی‌شود ولی همچنان عبارت ok که در finally ذکر شده در خروجی چاپ می‌شود. یک مثال خوب استفاده از چنین وضعیتی، زمانی است که شما یک ارتباط با بانک اطلاعاتی باز می‌کنید، و نتیجه یک عملیات را با دستور return به کاربر بر می‌گردانید. مسئله این است که در این وضعیت چگونه ارتباط با دیتابیس بسته شده و منابع آزاد می‌گردند؟ اگر در حین عملیات بانک اطلاعاتی، خطایی رخ دهد یا ندهد، و شما دستور آزاد سازی منابع و بستن ارتباط را در داخل قسمت finally نوشته باشید، وقتی دستور return فراخوانی می‌شود، ابتدا منابع آزاد و سپس مقدار به خروجی بر می‌گردد.

```
public int GetUserId(string nickname)
{
    SqlConnection connection = new SqlConnection(...);
    SqlCommand command = connection.CreateCommand();
    command.CommandText = "select id from users where nickname like @nickname";
    command.Parameters.Add(new SqlParameter("@nickname", nickname));
     try {
        connection.Open();
         return Convert.ToInt32(command.ExecuteScalar());
     }
     catch(SqlException exception)
     {
     // some exception handling
        return -1;
     } finally {
         if (connection.State == ConnectionState.Open)
        connection.Close();
     }
     // if all things works, you can not reach here
}
```

نظرات خوانندگان

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۷/۰۹ ۱۱:۲۸

- اینکه شما بروز یک مشکل رو با یک عدد منفی از یک متد بازگشت می‌دید یعنی هنوز دید زبان C رو دارید. در دات نت وجود استثناءها دقیقا برای ننوشتن return 0 یا -1 و شبیه به آن هست. در این حالت برنامه خودکار در هر سطحی که باشد، ادامه‌اش متوقف میشه و نیازی نیست تا مدام خروجی یک متد رو چک کرد.

- اینکه در یک متد کانکشنی برقرار شده و بسته شده یعنی ضعف کپسوله سازی مفاهیم ADO.NET. نباید این مسایل رو مدام در تمام متدها تکرار کرد. میشه یک متد عمومی ExecSQL درست کرد بجای تکرار مدام یک سری کد.

- یک سری از اشیاء اینترفیس IDisposable رو پیاده سازی می‌کنند مثل همین شیء اتصالی که ذکر شد. در این حالت میشه از using استفاده کرد بجای try/finally و اون وقت به دوتا using نیاز خواهید داشت یعنی شیء Command هم نیاز به try/finally داره.

نویسنده: فانوس
تاریخ: ۱۳۹۲/۰۷/۰۹ ۱۱:۵۰

دوست عزیزم. من این رو به عنوان یک مثال ساده برای درک مفهوم مورد بحث نوشتم و نخواستم خیلی برای افرادی که تازه سی شارپ رو شروع می‌کنند پیچیده باشه. قواعدی که شما فرمودید کاملا درست هست. متشکرم.

نویسنده: محمد مهدی
تاریخ: ۱۳۹۲/۰۷/۱۰ ۰:۱

لطف کنید در مورد مدیریت استثناء در لایه‌های مختلف توضیح بدین.اینکه چجوری این استثناءها به لایه بالاتر یا همون اینترفیس منتقل بشه

نویسنده: رضا منصوری
تاریخ: ۱۳۹۲/۰۷/۱۰ ۱۰:۱۹

«*برای افرادی که تازه سی شارپ رو شروع می‌کنند*» با تشکر از مطلبتون **به نظر من** کسی اینجا تازه سی شارپو شروع نکرده اگه میشه مطالبتونو تخصصی‌تر کنید ممنون