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

عنوان: **مدیریت AccessViolationException در برنامه‌های دات نت 4 به بعد**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۱۴ ۱:۲۰
آدرس: www.dotnettips.info
گروه‌ها: C#, C++, exception handling

فرض کنید که از یک برنامه‌ی native ویندوز برای تهیه تصاویر سایت‌ها در یک برنامه‌ی وب استفاده می‌کنید و صبح که به سایت سر زده‌اید پیام در دسترس نبودن سایت قابل مشاهده است. مشکل از کجا است؟!

### یک مثال ساده

```
using System;

namespace AccessViolationExceptionSample
{
    class Program
    {
        private static unsafe void AccessViolation()
        {
            byte b = *(byte*)(8762765876);
        }

        static void Main(string[] args)
        {
            try
            {
                AccessViolation();
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex);
            }

            Console.WriteLine("Press a key...");
            Console.ReadKey();
        }
    }
}
```

برنامه‌ی کنسول فوق را پس از فعال سازی Allow unsafe code در قسمت تنظیمات پروژه، کامپایل کرده و سپس آن‌را خارج از VS.NET اجرا کنید. احتمالا انتظار دارید که قسمت catch این کد حداقل اجرا شود و سپس سطر «کلیدی را فشار دهید» ظاهر گردد. اما ... خیر! کل پروسه کرش کرده و هیچ پیام خطایی را دریافت نخواهید کرد. اگر به لاگ‌های ویندوز مراجعه کنید پیام زیر قابل مشاهده است:

```
System.AccessViolationException.  Attempted to read or write protected memory.
 This is often an indication that other memory is corrupt.
```

و این نوع مسایل هنگام کار با کتابخانه‌های C و C++ زیاد ممکن است رخ دهند. نمونه‌ی آن استفاده از WebControl دات نت است یا هر برنامه‌ی native دیگری. در این حالت اگر برنامه‌ی شما یک برنامه‌ی وب باشد، عملا سایت از کار افتاده‌است. به عبارتی پروسه‌ی ویندوزی آن کرش کرده و بلافاصله از طرف ویندوز خاتمه یافته است.

### چرا قسمت catch اجرا نشد؟

از دات نت 4 به بعد ، زمانیکه دسترسی غیرمجازی به حافظه صورت گیرد، برای مثال دسترسی به یک pointer آزاد شده، استثنای حاصل، توسط برنامه catch نشده و اجازه داده می‌شود تا برنامه کلا کرش کند. به این نوع استثناءها Corrupted State Exceptions یا CSE گفته می‌شود. اگر نیاز به مدیریت آن‌ها توسط برنامه باشد، باید به یکی از دو طریق زیر عمل کرد:
الف) از ویژگی HandleProcessCorruptedStateExceptions بر روی متد فراخوان کتابخانه‌ی native باید استفاده شود. برای مثال در کدهای فوق خواهیم داشت:

```
[HandleProcessCorruptedStateExceptions]
  static void Main(string[] args)
  {
```

ب) و یا فایل کانفیگ برنامه را ویرایش کرده و چند سطر ذیل را به آن اضافه کنید:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <runtime>
     <legacyCorruptedStateExceptionsPolicy enabled="true" />
  </runtime>
</configuration>
```

در این حالت مدیریت اینگونه خطاها در کل برنامه همانند برنامه‌های تا دات نت 3.5 خواهد شد.

## نظرات خوانندگان

نویسنده: سوین
تاریخ: ۱۳۹۲/۱۱/۱۶ ۱۹:۴۶

با سلام
من در فایل کانفیگ یه WPF App ، تغییرات گفته شده را قرار دادم و خطای زیر رو در vs داد
.The type initializer for 'System.Windows.Application' threw an exception

و بیرون از vs اصلا اجرا نشد خیلی نیاز دارم به این مورد ، چون یه پروژه دارم که درست اجرا میشه اما بعضی مواقع برنامه کرش میکنه و نمی‌تونم catch کنم . با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۱۶ ۱۹:۵۹

- مطلب فوق بیشتر مرتبط است به استثناهای کتابخانه‌های native استفاده شده در برنامه‌های دات نت. برای سایر موارد باید در فایل App.xaml.cs موارد ذیل را بررسی کنید:

```
public partial class App
    {
        public App()
        {
            this.DispatcherUnhandledException += appDispatcherUnhandledException;
            AppDomain.CurrentDomain.UnhandledException += CurrentDomain_UnhandledException;
        }
```

+ نمونه تنظیم زیر در فایل app.config یک برنامه WPF کار می‌کند (آزمایش شد):

```
<?xml version="1.0"?>
<configuration>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.0"/>
  </startup>
  <runtime>
     <legacyCorruptedStateExceptionsPolicy enabled="true" />
  </runtime>
</configuration>
```

نویسنده: سوین
تاریخ: ۱۳۹۲/۱۲/۰۲ ۹:۳۸

با سلام؛ من یه پروژه با WPF نوشتم اما یه ایراد داره و اونم اینه که مثلا فرم1 رو 20 بار اجرا می‌کنی خطا نمی‌ده اما بار 21 ام برنامه کرش میکنه و اصلا نمیشه catch کرد. متن خطا در Log ویندوز اینه

```
Error 01 :
Application: MyWPFApp.exe
Framework Version: v4.0.30319
Description: The process was terminated due to an unhandled exception.
Exception Info: exception code c0000005, exception address 77D52239

Error 02 :
Faulting application name: MyWPFApp.exe, version: 1.0.0.0, time stamp: 0x52d550ac
Faulting module name: ntdll.dll, version: 6.1.7601.17514, time stamp: 0x4ce7b96e
Exception code: 0xc0000005
Fault offset: 0x00032239
Faulting process id: 0xa28
Faulting application start time: 0x01cf113ae6813d88
Faulting application path: R:\Source\MyWPFApp\bin\Debug\MyWPFApp.exe
Faulting module path: C:\Windows\SYSTEM32\ntdll.dll
Report Id: 460eda62-7d33-11e3-a572-ac220bc99cf8

Information :
```

```
Fault bucket , type 0
Event Name: APPCRASH
Response: Not available
Cab Id: 0

Problem signature:
P1: MyWPFApp.exe
P2: 1.0.0.0
P3: 52d550ac
P4: ntdll.dll
P5: 6.1.7601.17514
P6: 4ce7b96e
P7: c0000005
P8: 00032239
P9:
P10:

Attached files:
C:\Users\Administrator\AppData\Local\Temp\WERE9B3.tmp.WERInternalMetadata.xml
C:\Users\Administrator\AppData\Local\Temp\WER16AC.tmp.appcompat.txt
C:\Users\Administrator\AppData\Local\Temp\WER18A1.tmp.hdmp
C:\Users\Administrator\AppData\Local\Temp\WER3BFA.tmp.mdmp

These files may be available here:
C:\Users\Administrator\AppData\Local\Microsoft\Windows\WER\ReportQueue\AppCrash_MyWPFApp.exe_125fc667a6
9fcc31c463a5e1b4032657c4ce830_cab_0ac03d3e

Analysis symbol:
Rechecking for solution: 0
Report Id: 460eda62-7d33-11e3-a572-ac220bc99cf8
```

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۱۲/۰۲ ۱۱:۳۷

از چه کامپوننتی استفاده کردی ؟ بهتره اون فایل‌های کرش دامپ dmp رو براشون ارسال کنی.

نویسنده: سوین
تاریخ: ۱۳۹۲/۱۲/۰۲ ۱۹:۲۵

با سلام
از کامپوننت شرکت‌های ثالث استفاده نکردم . آیا راه حل کلی برای پیدا کردن چنین خطاهایی وجود نداره ، اینترنت رو هم سرچ کردم اما کمک زیادی نکرد که بشه فهمید مشکل از چیه و قبل ارسال این پست 2 ساعت تمام آزمایش کردم خطا نداد اما بعضی مواقع این اتفاق می‌افته .
در ضمن این برنامه WPF App که برای اوتوماسیون اداری نوشته شده و از EF 6.2 ، قفل سخت افزاری (که بدون قفل هم این ایراد رو می‌ده)

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۲/۰۳ ۰:۴۶

مثال WPF ایی که AccessViolation عمدی دارد: WpfApplicationC0000005.zip
به فایل‌های App.config و App.xaml.cs آن دقت کنید.
پروژه را کامپایل کرده و خارج از VS.NET اجرا کنید. خطا را نمایش می‌دهد ولی کرش نمی‌کند.

نویسنده: سوین
تاریخ: ۱۳۹۲/۱۲/۰۳ ۹:۲۱

با سلام
من تگ startup رو به صورت زیر نوشته بودم آیا می‌تونه تاثیر داشته باشه

```
<startup useLegacyV2RuntimeActivationPolicy="true">
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.0"/>
    <requiredRuntime version="v4.0.20506"/>
</startup>
```

باز تست می‌کنم ببینم چی میشه ، انشاالله که درست بشه . با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۲/۰۳ ۹:۳۰

تنظیم یاد شده مربوط به تگ runtime است.

عنوان: first chance exception چیست؟
نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۶/۳۱ ۱۲:۱۰
آدرس: www.dotnettips.info
گروه‌ها: WPF, exception handling

چند سال قبل یک datapicker تقویم شمسی را برای سیلورلایت تهیه کردم. بعد از آن نسخه‌ی WPF آن هم به پروژه اضافه شد . تا اینکه مدتی قبل مشکل عدم کار کردن آن در یک صفحه‌ی دیالوگ جدید در ویندوز 8 گزارش شد. در حین برطرف کردن این مشکل، مدام سطر ذیل در پنجره‌ی output ویژوال استودیو نمایش داده می‌شد:

```
A first chance exception of type 'System.ArgumentOutOfRangeException' occurred in mscorlib.dll
```

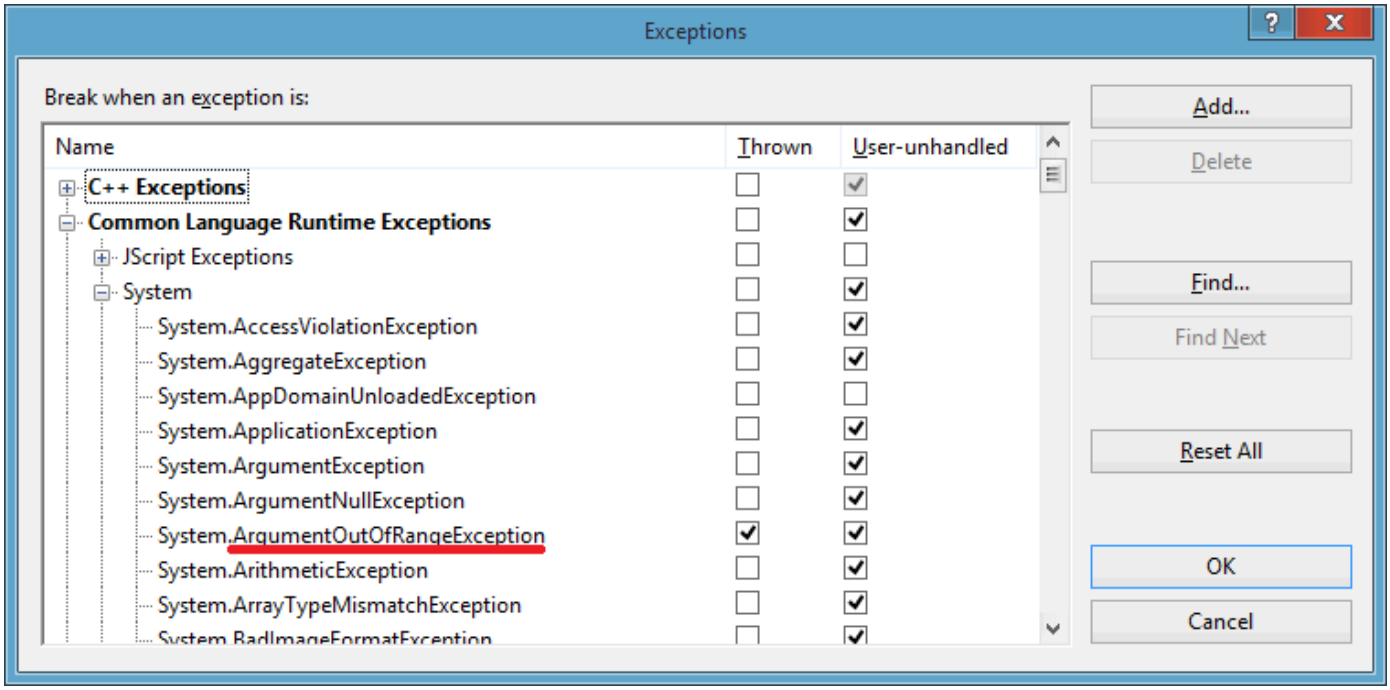
البته برنامه بدون مشکل کار می‌کرد و صفحه‌ی نمایش Exception در VS.NET ظاهر نمی‌شد.

**سؤال: first chance exception چیست؟**

وقتی استثنایی در یک برنامه رخ می‌دهد، به آن یک first chance exception می‌گویند. این اولین شانسی است که سیستم به شما می‌دهد تا استثنای رخ داده را مدیریت کنید. اگر کدهای برنامه یا ابزاری (یک try/catch یا دیباگر) این اولین شانس را ندید بگیرند، یک second chance exception رخ می‌دهد. این‌جا است که برنامه به احتمال زیاد خاتمه خواهد یافت.
مشاهده‌ی پیام‌های A first chance exception در پنجره‌ی output ویژوال استودیو به این معنا است که استثنایی رخ داده، اما توسط یک استثناءگردان مدیریت شده‌است. بنابراین در اکثر موارد، موضوع خاصی نیست و می‌توان از آن صرفنظر کرد.

**سؤال: چگونه می‌توان منشاء اصلی پیام رخ‌دادن یک first chance exception را یافت؟**

ویژوال استودیو در پنجره‌ی output، مدام پیام رخ‌دادن first chance exception را نمایش می‌دهد؛ اما واقعا کدام قطعه از کدهای برنامه سبب بروز آن شده‌اند؟ به صورت پیش فرض صفحه‌ی نمایش استثناءها در VS.NET زمانی نمایان می‌شود که استثنای رخ داده، مدیریت نشده باشد. برای فعال سازی نمایش استثناهای مدیریت شده باید تنظیمات ذیل را اعمال کرد:
- به منوی Debug | Exceptions مراجعه کنید.
- گره Common Language Runtime Exceptions را باز کنید.
- سپس گروه System آن‌را نیز باز کنید.
- در اینجا بر اساس نوع استثنایی که در پنجره‌ی output نمایش داده می‌شود، آن استثناء را یافته و Thrown آن‌را انتخاب کنید.




اینبار اگر برنامه را اجرا کنید، دقیقا محلی که سبب بروز استثنای ArgumentOutOfRangeException شده در VS.NET گزارش داده خواهد شد.

عنوان: **تولید MiniDump در حین کرش برنامه‌های دات نت**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۱۱/۱۶ ۸:۵
آدرس: www.dotnettips.info
گروه‌ها: Debugging, exception handling, Crash Dump

با مطالعه‌ی سورس‌های محصولات اخیرا سورس باز شده‌ی مایکروسافت، نکات جالبی را می‌توان استخراج کرد. برای نمونه اگر سورس پروژه‌ی Orleans را بررسی کنیم، در حین بررسی اطلاعات استثناءهای رخ داده‌ی در برنامه، متد TraceLogger.CreateMiniDump نیز بکار رفته‌است. در این مطلب قصد داریم، این متد و نحوه‌ی استفاده‌ی از حاصل آن‌را بررسی کنیم.

### تولید MiniDump در برنامه‌های دات نت

خلاصه‌ی روش تولید MiniDump در پروژه‌ی Orleans به صورت زیر است:

**الف) حالت‌های مختلف تولید فایل دامپ که مقادیر آن قابلیت Or شدن را دارا هستند:**

```
[Flags]
public enum MiniDumpType
{
    MiniDumpNormal = 0x00000000,
    MiniDumpWithDataSegs = 0x00000001,
    MiniDumpWithFullMemory = 0x00000002,
    MiniDumpWithHandleData = 0x00000004,
    MiniDumpFilterMemory = 0x00000008,
    MiniDumpScanMemory = 0x00000010,
    MiniDumpWithUnloadedModules = 0x00000020,
    MiniDumpWithIndirectlyReferencedMemory = 0x00000040,
    MiniDumpFilterModulePaths = 0x00000080,
    MiniDumpWithProcessThreadData = 0x00000100,
    MiniDumpWithPrivateReadWriteMemory = 0x00000200,
    MiniDumpWithoutOptionalData = 0x00000400,
    MiniDumpWithFullMemoryInfo = 0x00000800,
    MiniDumpWithThreadInfo = 0x00001000,
    MiniDumpWithCodeSegs = 0x00002000,
    MiniDumpWithoutManagedState = 0x00004000
}
```

**ب) متد توکار ویندوز برای تولید فایل دامپ**

```
public static class NativeMethods
{
    [DllImport("Dbghelp.dll")]
    public static extern bool MiniDumpWriteDump(
        IntPtr hProcess,
        int processId,
        IntPtr hFile,
        MiniDumpType dumpType,
        IntPtr exceptionParam,
        IntPtr userStreamParam,
        IntPtr callbackParam);
}
```

**ج) فراخوانی متد تولید دامپ در برنامه**

در اینجا نحوه‌ی استفاده از enum و متد MiniDumpWriteDump ویندوز را مشاهده می‌کنید:

```
public static class DebugInfo
{
    public static void CreateMiniDump(
        string dumpFileName, MiniDumpType dumpType = MiniDumpType.MiniDumpNormal)
    {
        using (var stream = File.Create(dumpFileName))
        {
            var process = Process.GetCurrentProcess();
            // It is safe to call DangerousGetHandle() here because the process is already crashing.
            NativeMethods.MiniDumpWriteDump(
                        process.Handle,
```

```
                            process.Id,
                            stream.SafeFileHandle.DangerousGetHandle(),
                            dumpType,
                            IntPtr.Zero,
                            IntPtr.Zero,
                            IntPtr.Zero);
        }
    }

    public static void CreateMiniDump(MiniDumpType dumpType = MiniDumpType.MiniDumpNormal)
    {
        const string dateFormat = "yyyy-MM-dd-HH-mm-ss-fffZ"; // Example: 2010-09-02-09-50-43-341Z
        var thisAssembly = Assembly.GetEntryAssembly() ?? Assembly.GetCallingAssembly();
        var dumpFileName = string.Format(@"{0}-MiniDump-{1}.dmp",
                    thisAssembly.GetName().Name,
                    DateTime.UtcNow.ToString(dateFormat, CultureInfo.InvariantCulture));

        var path = Path.Combine(getApplicationPath(), dumpFileName);
        CreateMiniDump(path, dumpType);
    }

    private static string getApplicationPath()
    {
        return HttpContext.Current != null ?
            HttpRuntime.AppDomainAppPath :
            Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location);
    }
}
```

متد MiniDumpWriteDump نیاز به اطلاعات پروسه‌ی جاری، به همراه هندل فایلی که قرار است اطلاعات را در آن بنویسد، دارد. همچنین dump type آن نیز می‌تواند ترکیبی از مقادیر enum مرتبط باشد.

**یک مثال:**

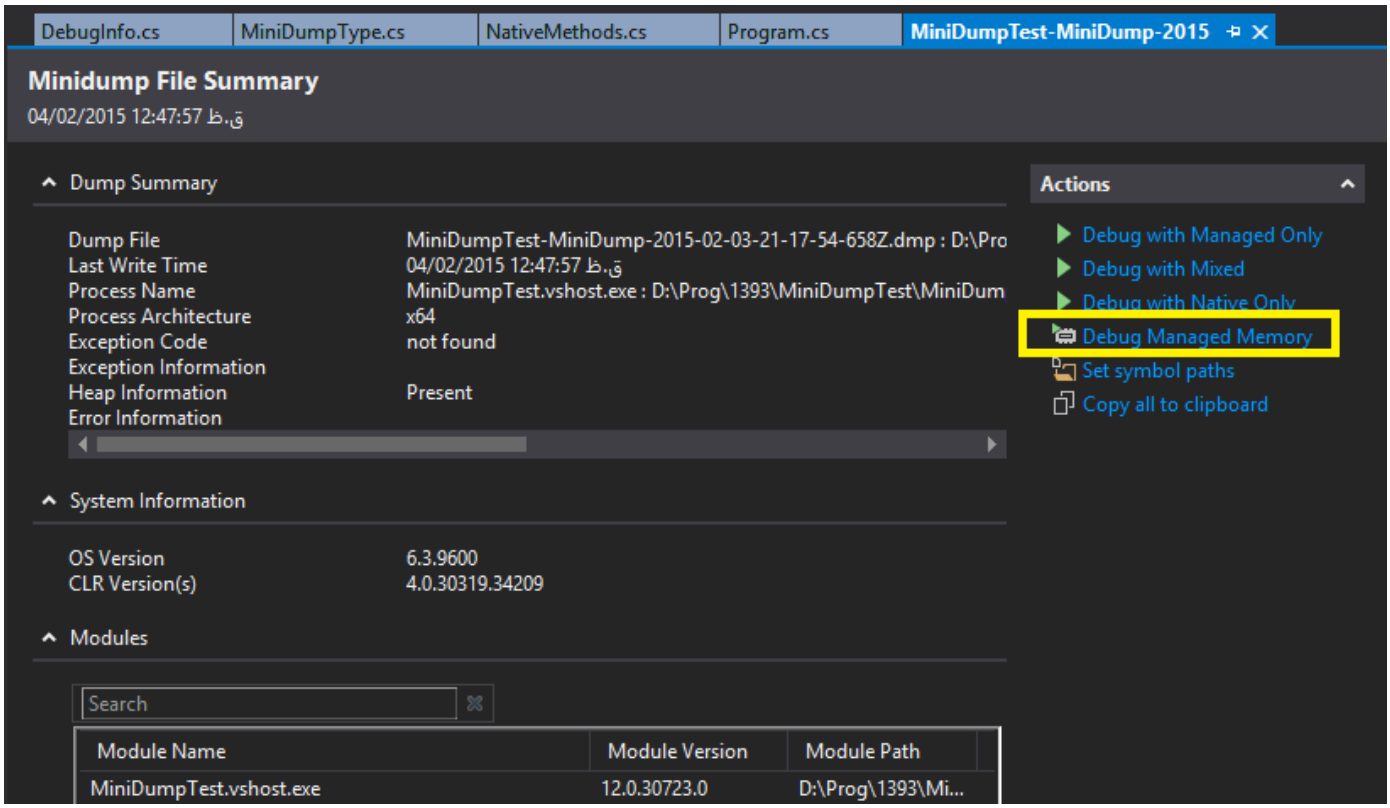
```
class Program
{
    static void Main(string[] args)
    {
        try
        {
            var zero = 0;
            Console.WriteLine(1 / zero);
        }
        catch (Exception ex)
        {
            Console.Write(ex);
            DebugInfo.CreateMiniDump(dumpType:
                                MiniDumpType.MiniDumpNormal |
                                MiniDumpType.MiniDumpWithPrivateReadWriteMemory |
                                MiniDumpType.MiniDumpWithDataSegs |
                                MiniDumpType.MiniDumpWithHandleData |
                                MiniDumpType.MiniDumpWithFullMemoryInfo |
                                MiniDumpType.MiniDumpWithThreadInfo |
                                MiniDumpType.MiniDumpWithUnloadedModules);

            throw;
        }
    }
}
```

در اینجا نحوه‌ی فراخوانی متد CreateMiniDump را در حین کرش برنامه مشاهده می‌کنید. پارامترهای اضافی دیگر سبب خواهند شد تا اطلاعات بیشتری از حافظه‌ی جاری سیستم، در دامپ نهایی قرار گیرند. اگر پس از اجرای برنامه، به پوشه‌ی bin\debug مراجعه کنید، فایل dmp تولیدی را مشاهده خواهید کرد.

**نحوه‌ی بررسی فایل‌های dump**

**الف) با استفاده از Visual studio 2013**

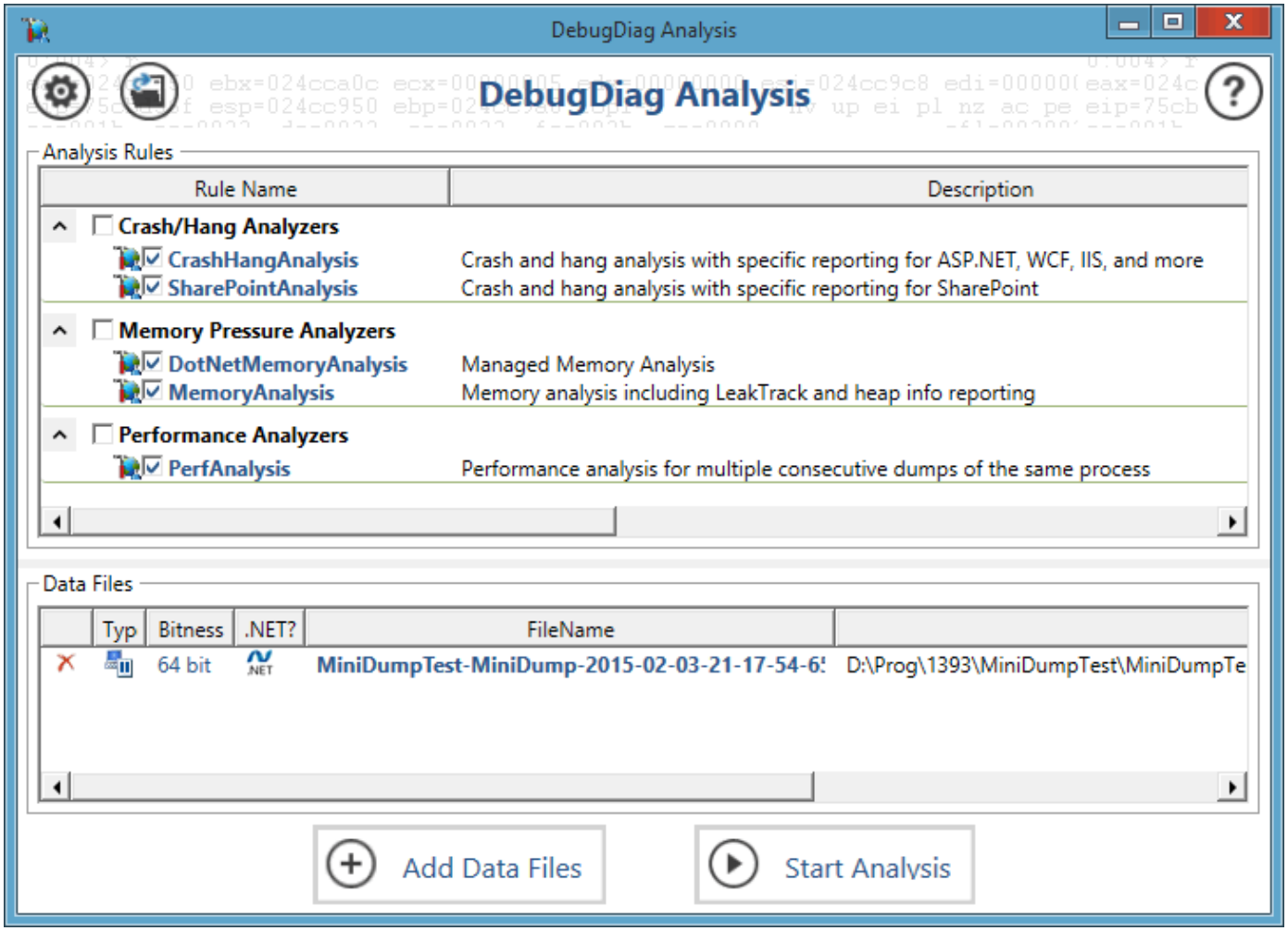
از به روز رسانی سوم VS 2013 به بعد، فایل‌های dump را می‌توان داخل خود VS.NET نیز آنالیز کرد ( ^ و ^ و ^ ). برای نمونه تصویر ذیل، حاصل کشودن فایل کرش مثال فوق است:




در اینجا اگر بر روی لینک debug managed memory کلیک کنید، پس از چند لحظه، آنالیز کامل اشیاء موجود در حافظه را در حین تهیه‌ی دامپ تولیدی، می‌توان مشاهده کرد. این مورد برای آنالیز نشتی‌های حافظه‌ی یک برنامه بسیار مفید است.

**ب) استفاده از برنامه‌ی Debug Diagnostic Tool**

برنامه‌ی Debug Diagnostic Tool را از اینجا می‌توانید دریافت کنید. این برنامه نیز قابلیت آنالیز فایل‌های دامپ را داشته و اطلاعات بیشتری را پس از آنالیز ارائه می‌دهد.




برای نمونه پس از آنالیز فایل دامپ تهیه شده توسط این برنامه، خروجی ذیل حاصل می‌شود:


**کدهای کامل این مثال را از اینجا می‌توانید دریافت کنید:**

MiniDumpTest.zip

عنوان: **نکات کار با استثناءها در دات نت**
نویسنده: سیروان عفیفی
تاریخ: ۱۳۹۳/۱۲/۲۹ ۲۰:۵
آدرس: www.dotnettips.info
گروه‌ها: C#, Design patterns, .NET, Exception, exception handling

**استثناء چیست؟**

واژه‌ی استثناء یا exception کوتاه شده‌ی عبارت exceptional event است. در واقع exception یک نوع رویداد است که در طول اجرای برنامه رخ می‌دهد و در نتیجه، جریان عادی برنامه را مختل می‌کند. زمانیکه خطایی درون یک متد رخ دهد، یک شیء (exception object) حاوی اطلاعاتی درباره‌ی خطا ایجاد خواهد شد. به فرآیند ایجاد یک exception object و تحویل دادن آن به سیستم runtime، اصطلاحاً throwing an exception یا صدور استثناء گفته می‌شود که در ادامه به آن خواهیم پرداخت.

بعد از اینکه یک متد استثناءایی را صادر می‌کند، سیستم runtime سعی در یافتن روشی برای مدیریت آن خواهد کرد.

خوب اکنون که با مفهوم استثناء آشنا شدید اجازه دهید دو سناریو را با هم بررسی کنیم.

- سناریوی اول:

فرض کنید یک فایل XML از پیش تعریف شده (برای مثال یک لیست از محصولات) قرار است در کنار برنامه‌ی شما باشد و باید این لیست را درون برنامه‌ی خود نمایش دهید. در این حالت برای خواندن این فایل انتظار دارید که فایل وجود داشته باشد. اگر این فایل وجود نداشته باشد برنامه‌ی شما با اشکال روبرو خواهد شد.

- سناریوی دوم:

فرض کنید یک فایل XML از آخرین محصولات مشاهده شده توسط کاربران را به صورت cache در برنامه‌تان دارید. در این حالت در اولین بار اجرای برنامه توسط کاربر انتظار داریم که این فایل موجود نباشد و اگر فایل وجود نداشته باشد به سادگی می‌توانیم فایل مربوط را ایجاده کرده و محصولاتی را که توسط کاربر مشاهده شده، درون این فایل اضافه کنیم.

در واقع استثناءها بستگی به حالت‌های مختلفی دارد. در مثال اول وجود فایل حیاتی است ولی در حالت دوم بدون وجود فایل نیز برنامه می‌تواند به کار خود ادامه داده و فایل مورد نظر را از نو ایجاد کند.

استثناها مربوط به زمانی هستند که این احتمال وجود داشته باشد که برنامه طبق انتظار پیش نرود.

برای حالت اول کد زیر را داریم:

```
public IEnumerable<Product> GetProducts()
{
    using (var stream = File.Read(Path.Combine(Environment.CurrentDirectory, "products.xml")))
    {
        var serializer = new XmlSerializer();
        return (IEnumerable<Product>)serializer.Deserialize(stream);
    }
}
```

همانطور که عنوان شد در حالت اول انتظار داریم که فایلی بر روی دیسک موجود باشد. در نتیجه نیازی نیست هیچ استثناءایی را مدیریت کنیم (زیرا در واقع اگر فایل موجود نباشد هیچ روشی برای ایجاد آن نداریم).

در مثال دوم می‌دانیم که ممکن است فایل از قبل موجود نباشد. بنابراین می‌توانیم موجود بودن فایل را با یک شرط بررسی کنیم:

```
public IEnumerable<Product> GetCachedProducts()
{
    var fullPath = Path.Combine(Environment.CurrentDirectory, "ProductCache.xml");
    if (!File.Exists(fullPath))
        return new Product[0];

    using (var stream = File.Read(fullPath))
    {
        var serializer = new XmlSerializer();
        return (IEnumerable<Product>)serializer.Deserialize(stream);
    }
}
```

**چه زمانی باید استثناءها را مدیریت کنیم؟**

زمانیکه بتوان متدهایی که خروجی مورد انتظار را بر می‌گردانند ایجاد کرد.

اجازه دهید دوباره از مثال‌های فوق استفاده کنیم:

```
IEnumerable<Product> GetProducts()
```

همانطور که از نام آن پیداست این متد باید همیشه لیستی از محصولات را برگرداند. اگر می‌توانید اینکار را با استفاده از catch کردن یک استثنا انجام دهید در غیر اینصورت نباید درون متد اینکار را انجام داد.

```
IEnumerable<Product> GetCachedProducts()
```

در متد فوق می‌توانستیم از FileNotFoundException برای فایل موردنظر استفاده کنیم؛ اما مطمئن بودیم که فایل در ابتدا وجود ندارد.
در واقع استثناها حالت‌هایی هستند که غیرقابل پیش‌بینی هستند. این حالت‌ها می‌توانند یک خطای منطقی از طرف برنامه‌نویس و یا چیزی خارج کنترل برنامه‌نویس باشند (مانند خطاهای سیستم‌عامل، شبکه، دیسک). یعنی در بیشتر مواقع این نوع خطاها را نمی‌توان مدیریت کرد.

**اگر می‌خواهید استثناءها را catch کرده و آنها را لاگ کنید در بالاترین لایه اینکار را انجام دهید.**

**چه استثناءهایی باید مدیریت شوند و کدام‌ها خیر؟**
مدیریت صحیح استثناءها می‌تواند خیلی مفید باشد. همانطور که عنوان شد یک استثناء زمانی رخ می‌دهد که یک حالت استثناء در برنامه اتفاق بیفتد. این مورد را بخاطر داشته باشید، زیرا به شما یادآوری می‌کند که در همه جا نیازی به استفاده از try/catch نیست. در اینجا ذکر این نکته خیلی مهم است:
تنها استثناءهایی را catch کنید که بتوانید برای آن راه‌حلی ارائه دهید.
به عنوان مثال اگر در لایه‌ی دسترسی به داده، خطایی رخ دهد و استثناءی SqlException صادر شود، می‌توانیم آن را catch کرده و درون یک استثناء عمومی‌تر قرار دهیم:

```
public class UserRepository : IUserRepository
{
    public IList<User> Search(string value)
    {
        try
        {
             return CreateConnectionAndACommandAndReturnAList("WHERE value=@value",
Parameter.New("value", value));
        }
        catch (SqlException err)
        {
            var msg = String.Format("Ohh no!  Failed to search after users with '{0}' as search
string", value);
            throw new DataSourceException(msg, err);
        }
    }
}
```

همانطور که در کد فوق مشاهده می‌کنید به محض صدور استثنای SqlException آن را درون قسمت catch به صورت یک استثنای عمومی‌تر همراه با افزودن یک سری اطلاعات جدید صادر می‌کنیم. اما همانطور که عنوان شد کار لاگ کردن استثناءها را بهتر است در لایه‌های بالاتر انجام دهیم.
اگر مطمئن نیستید که تمام استثناءها توسط شما مدیریت شده‌اند، می‌توانید در حالت‌های زیر، دیگر استثناءها را مدیریت کنید:
ASP.NET: می‌توانید Aplication_Error را پیاده‌سازی کنید. در اینجا فرصت خواهید داشت تا تمامی خطاهای مدیریت نشده را هندل کنید.
WinForms: استفاده از رویدادهای Application.ThreadException و AppDomain.CurrentDomain.UnhandledException
WCF: پیاده‌سازی اینترفیس IErrorHandler
ASMX: ایجاد یک Soap Extension سفارشی
ASP.NET WebAPI

**چه زمان‌هایی باید یک استثناء صادر شود؟**

صادر کردن یک استثناء به تنهایی کار ساده‌ایی است. تنها کافی است throw را همراه شیء exception object) exception) فراخوانی کنیم. اما سوال اینجاست که چه زمانی باید یک استثناء را صادر کنیم؟ چه داده‌هایی را باید به استثناء اضافه کنیم؟ در ادامه به این سوالات خواهیم پرداخت.
همانطور که عنوان گردید استثناءها زمانی باید صادر شوند که یک استثناء اتفاق بیفتد.

**اعتبارسنجی آرگومان‌ها**

ساده‌ترین مثال، آرگومان‌های مورد انتظار یک متد است:

```
public void PrintName(string name)
{
     Console.WriteLine(name);
}
```

در حالت فوق انتظار داریم مقداری برای پارامتر name تعیین شود. متد فوق با آرگومان null نیز به خوبی کار خواهد کرد؛ یعنی مقدار خروجی یک خط خالی خواهد بود. از لحاظ کدنویسی متد فوق به خوبی کار خود را انجام می‌دهد اما خروجی مورد انتظار کاربر نمایش داده نمی‌شود. در این حالت نمی‌توانیم تشخیص دهیم مشکل از کجا ناشی می‌شود.
مشکل فوق را می‌توانیم با صدور استثنای ArgumentNullException رفع کنیم:

```
public void PrintName(string name)
{
    if (name == null) throw new ArgumentNullException("name");

     Console.WriteLine(name);
}
```

خوب، name باید دارای طول ثابت و همچنین ممکن است حاوی عدد و حروف باشد:

```
public void PrintName(string name)
{
    if (name == null) throw new ArgumentNullException("name");
    if (name.Length < 5 || name.Length > 10) throw new ArgumentOutOfRangeException("name", name, "Name
must be between 5 or 10 characters long");
    if (name.Any(x => !char.IsAlphaNumeric(x)) throw new ArgumentOutOfRangeException("name", name, "May
only contain alpha numerics");

     Console.WriteLine(name);
}
```

برای حالت فوق و همچنین جلوگیری از تکرار کدهای داخل متد PrintName می‌توانید یک متد Validator برای کلاسی با نام Person ایجاد کنید.
حالت دیگر صدور استثناء، زمانی است که متدی خروجی مورد انتظارمان را نتواند تحویل دهد. یک مثال بحث‌برانگیز متدی با امضای زیر است:

```
public User GetUser(int id)
{
}
```

کاملاً مشخص است که متدی همانند متد فوق زمانیکه کاربری را پیدا نکند، مقدار null را برمی‌گرداند. اما این روش درستی است؟ خیر؛ زیرا همانطور که از نام این متد پیداست باید یک کاربر به عنوان خروجی برگردانده شود.
با استفاده از بررسی null کدهایی شبیه به این را در همه جا خواهیم داشت:

```
var user = datasource.GetUser(userId);
if (user == null)
    throw new InvalidOperationException("Failed to find user: " + userId);
// actual logic here
```

به این چنین کدهایی معمولاً The null cancer گفته می‌شود (سرطان نال!) زیرا اجازه داده‌ایم متد، خروجی null را بازگشت دهد. به جای کد فوق می‌توانیم از این روش استفاده کنیم:

```
public User GetUser(int id)
{
    if (id <= 0) throw new ArgumentOutOfRangeException("id", id, "Valid ids are from 1 and above. Do
you have a parsing error somewhere?");

    var user = db.Execute<User>("WHERE Id = ?", id);
    if (user == null)
        throw new EntityNotFoundException("Failed to find user with id " + id);

    return user;
}
```

نکته‌ایی که باید به آن توجه کنید این است که در هنگام صدور یک استثناء اطلاعات کافی را نیز به آن پاس دهید. به عنوان مثال در EntityNotFoundException مثال فوق پاس دادن "Failed to find user with id " + id کار دیباگ را برای مصرف کننده، راحتر خواهد کرد.

**خطاهای متداول حین کار با استثناءها**

صدور مجدد استثناء و از بین بردن stacktrace

کد زیر را در نظر بگیرید:

```
try
{
    FutileAttemptToResist();
}
catch (BorgException err)
{
     _myDearLog.Error("I'm in da cube! Ohh no!", err);
    throw err;
}
```

مشکل کد فوق قسمت throw err است. این خط کد، محتویات stacktrace را از بین برده و استثناء را مجدداً برای شما ایجاد خواهد کرد. در این حالت هرگز نمی‌توانیم تشخیص دهیم که منبع خطا از کجا آمده است. در این حالت پیشنهاد می‌شود که تنها از throw استفاده شود. در این حالت استثناء اصلی مجدداً صادر گردیده و مانع حذف شدن محتویات stacktrace خواهد شد( + ).

اضافه نکردن اطلاعات استثناء اصلی به استثناء جدید

یکی دیگر از خطاهای رایج اضافه نکردن استثناء اصلی حین صدور استثناء جدید است:

```
try
{
    GreaseTinMan();
}
catch (InvalidOperationException err)
{
    throw new TooScaredLion("The Lion was not in the m00d", err); //<---- استثناء اصلی بهتر است به استثناء
جدید پاس داده شود
}
```

**ارائه ندادن context information**

در هنگام صدور یک استثناء بهتر است اطلاعات دقیقی را به آن ارسال کنیم تا دیباگ کردن آن به راحتی انجام شود. به عنوان مثال کد زیر را در نظر داشته باشید:

```
try
{
```

```
   socket.Connect("somethingawful.com", 80);
}
catch (SocketException err)
{
    throw new InvalidOperationException("Socket failed", err);
}
```

هنگامی که کد فوق با خطا مواجه شود نمی‌توان تنها با متن Socket failed تشخیص داد که مشکل از چه چیزی است. بنابراین پیشنهاد می‌شود اطلاعات کامل و در صورت امکان به صورت دقیق را به استثناء ارسال کنید. به عنوان مثال در کد زیر سعی شده است تا حد امکان context information کاملی برای استثناء ارائه شود:

```
void IncreaseStatusForUser(int userId, int newStatus)
{
    try
    {
         var user  = _repository.Get(userId);
         if (user == null)
             throw new UpdateException(string.Format("Failed to find user #{0} when trying to increase
status to {1}", userId, newStatus));

         user.Status = newStatus;
         _repository.Save(user);
    }
   catch (DataSourceException err)
   {
       var errMsg = string.Format("Failed to find modify user #{0} when trying to increase status to
{1}", userId, newStatus);
        throw new UpdateException(errMsg, err);
   }
```

**نحوه‌ی طراحی استثناءها**

برای ایجاد یک استثناء سفارشی می‌توانید از کلاس Exception ارث‌بری کنید و چهار سازنده‌ی آن را اضافه کنید:

```
public NewException()
public NewException(string description )
public NewException(string description, Exception inner)
protected or private NewException(SerializationInfo info, StreamingContext context)
```

سازنده اول به عنوان default constructor شناخته می‌شود. اما پیشنهاد می‌شود که از آن استفاده نکنید، زیرا یک استثناء بدون context information از ارزش کمی برخوردار خواهد بود.

سازنده‌ی دوم برای تعیین description بوده و همانطور که عنوان شد ارائه دادن context information از اهمیت بالایی برخوردار است. به عنوان مثال فرض کنید استثناء KeyNotFoundException که توسط کلاس Dictionary صادر شده است را دریافت کرده‌اید. این استثناء زمانی صادر خواهد شد که بخواهید به عنصری که درون دیکشنری پیدا نشده است دسترسی داشته باشید. در این حالت پیام زیر را دریافت خواهید کرد:

```
“The given key was not present in the dictionary.”
```

حالا فرض کنید اگر پیام به صورت زیر باشد چقدر باعث خوانایی و عیب‌یابی ساده‌تر خطا خواهد شد:

```
“The key ‘abrakadabra’ was not present in the dictionary.”
```

**در نتیجه تا حد امکان سعی کنید که context information شما کاملتر باشد.**

سازنده‌ی سوم شبیه به سازنده‌ی قبلی عمل می‌کند با این تفاوت که توسط پارامتر دوم می‌توانیم یک استثناء دیگر را catch کرده یک استثناء جدید صادر کنیم.

سازنده‌ی سوم زمانی مورد استفاده قرار می‌گیرد که بخواهید از Serialization پشتیبانی کنید (به عنوان مثال ذخیره‌ی استثناءها درون فایل و...)

خوب، برای یک استثناء سفارشی حداقل باید کدهای زیر را داشته باشیم:

```
public class SampleException : Exception
{
    public SampleException(string description)
        : base(description)
    {
        if (description == null) throw new ArgumentNullException("description");
    }

    public SampleException(string description, Exception inner)
        : base(description, inner)
    {
        if (description == null) throw new ArgumentNullException("description");
        if (inner == null) throw new ArgumentNullException("inner");
    }

    public SampleException(SerializationInfo info, StreamingContext context)
        : base(info, context)
    {
    }
}
```

**اجباری کردن ارائه‌ی Context information:**

برای اجباری کردن context information کافی است یک فیلد اجباری درون سازنده تعریف کنیم. برای مثال اگر بخواهیم کاربر HTTP status code را برای استثناء ارائه دهد باید سازنده‌ها را اینگونه تعریف کنیم:

```
public class HttpException : Exception
{
    System.Net.HttpStatusCode _statusCode;

    public HttpException(System.Net.HttpStatusCode statusCode, string description)
        : base(description)
    {
        if (description == null) throw new ArgumentNullException("description");
        _statusCode = statusCode;
    }

    public HttpException(System.Net.HttpStatusCode statusCode, string description, Exception inner)
        : base(description, inner)
    {
        if (description == null) throw new ArgumentNullException("description");
        if (inner == null) throw new ArgumentNullException("inner");
        _statusCode = statusCode;
    }

    public HttpException(SerializationInfo info, StreamingContext context)
        : base(info, context)
    {
    }

    public System.Net.HttpStatusCode StatusCode { get; private set; }

}
```

همچنین بهتر است پراپرتی Message را برای نمایش پیام مناسب بازنویسی کنید:

```
public override string Message
{
        get { return base.Message + "\r\nStatus code: " + StatusCode; }
}
```

مورد دیگری که باید در کد فوق مد نظر داشت این است که status code قابلیت سریالایز شدن را ندارد. بنابراین باید متد GetObjectData را برای سریالایز کردن بازنویسی کنیم:

```
public class HttpException : Exception
{
    // [...]

    public HttpException(SerializationInfo info, StreamingContext context)
        : base(info, context)
    {
```

```
        // this is new
        StatusCode = (HttpStatusCode) info.GetInt32("HttpStatusCode");
    }

    public HttpStatusCode StatusCode { get; private set; }

    public override string Message
    {
        get { return base.Message + "\r\nStatus code: " + StatusCode; }
    }

    // this is new
    public override void GetObjectData(SerializationInfo info, StreamingContext context)
    {
        base.GetObjectData(info, context);
        info.AddValue("HttpStatusCode", (int) StatusCode);
    }
}
```

در اینحالت فیلدهای اضافی در طول فرآیند Serialization به خوبی سریالایز خواهند شد.

در حین صدور استثناءها همیشه باید در نظر داشته باشیم که چه نوع context information را می‌توان ارائه داد، این مورد در یافتن راه‌حل خیلی کمک خواهد کرد.

**طراحی پیام‌های مناسب**

پیام‌های exception مختص به توسعه‌دهندگان است نه کاربران نهایی.
نوشتن این نوع پیام‌ها برای برنامه‌نویس کار خسته‌کننده‌ایی است. برای مثال دو مورد زیر را در نظر داشته باشید:

```
throw new Exception("Unknown FaileType");
throw new Exception("Unecpected workingDirectory");
```

این نوع پیام‌ها حتی اگر از لحاظ نوشتاری مشکلی نداشته باشند یافتن راه‌حل را خیلی سخت خواهند کرد. اگر در زمان برنامه‌نویسی با این نوع خطاها روبرو شوید ممکن است با استفاده از debugger ورودی نامعتبر را پیدا کنید. اما در یک برنامه و خارج از محیط برنامه‌نویسی، یافتن علت بروز خطا خیلی سخت خواهد بود.
توسعه‌دهندگانی که exception message را در اولویت قرار می‌دهند، معتقد هستند که از لحاظ تجربه‌ی کاربری پیام‌ها تا حد امکان باید فاقد اطلاعات فنی باشد. همچنین همانطور که پیش‌تر عنوان گردید این نوع پیام‌ها همیشه باید در بالاترین سطح نمایش داده شوند نه در لایه‌های زیرین. همچنین پیام‌هایی مانند Unknown FaileType نه برای کاربر نهایی، بلکه برای برنامه‌نویس نیز ارزش چندانی ندارد زیرا فاقد اطلاعات کافی برای یافتن مشکل است.
در طراحی پیام‌ها باید موارد زیر را در نظر داشته باشیم:
- امنیت:
یکی از مواردی که از اهمیت بالایی برخوردار است مسئله امنیت است از این جهت که پیام‌ها باید فاقد مقادیر runtime باشند. زیرا ممکن است اطلاعاتی را در خصوص نحوه‌ی عملکرد سیستم آشکار سازند.
- زبان:
همانطور که عنوان گردید پیام‌های استثناء برای کاربران نهایی نیستند، زیرا کاربران نهایی ممکن است اشخاص فنی نباشند، یا ممکن است زبان آنها انگلیسی نباشد. اگر مخاطبین شما آلمانی باشند چطور؟ آیا تمامی پیام‌ها را با زبان آلمانی خواهید نوشت؟ اگر هم اینکار را انجام دهید تکلیف استثناءهایی که توسط Base Class Library و دیگر کتابخانه‌های thirt-party صادر می‌شوند چیست؟ اینها انگلیسی هستند.

در تمامی حالت‌هایی که عنوان شد فرض بر این است که شما در حال نوشتن این نوع پیام‌ها برای یک سیستم خاص هستید. اما اگر هدف نوشتن یک کتابخانه باشد چطور؟ در این حالت نمی‌دانید که کتابخانه‌ی شما در کجا استفاده می‌شود.
اگر هدف نوشتن یک کتابخانه نباشد این نوع پیام‌هایی که برای کاربران نهایی باشند، وابستگی‌ها را در سیستم افزایش خواهند داد، زیرا در این حالت پیام‌ها به یک رابط کاربری خاص گره خواهند خورد.

خب اگر پیام‌ها برای کاربران نهایی نیستند، پس برای کسانی مورد استفاده قرار خواهند گرفت؟ در واقع این نوع پیام می‌تواند به

عنوان یک documentation برای سیستم شما باشند.
فرض کنید در حال استفاده از یک کتابخانه جدید هستید به نظر شما کدام یک از پیام‌های زیر مناسب هستند:

```
"Unecpected workingDirectory"
```

یا:

```
"You tried to provide a working directory string that doesn't represent a working directory. It's not your fault, because it wasn't possible to design the FileStore class in such a way that this is a statically typed pre-condition, but please supply a valid path to an existing directory.

"The invalid value was: "fllobdedy"."
```

یافتن مشکل در پیام اول خیلی سخت خواهد بود زیرا فاقد اطلاعات کافی برای یافتن مشکل است. اما پیام دوم مشکل را به صورت کامل توضیح داده است. در حالت اول شما قطعاً نیاز خواهید داشت تا از دیباگر برای یافتن مشکل استفاده کنید. اما در حالت دوم پیام به خوبی شما را برای یافتن راه‌حل راهنمایی می‌کند.
همیشه برای نوشتن پیام‌های مناسب سعی کنید از لحاظ نوشتاری متن شما مشکلی نداشته باشد، اطلاعات کافی را درون پیام اضافه کنید و تا حد امکان نحوه‌ی رفع مشکل را توضیح دهید

عنوان: **چگونه برنامه‌های دات نت را خارج از ویژوال استودیو دیباگ کنیم؟**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۱/۰۲ ۱۵:۰
آدرس: www.dotnettips.info
گروه‌ها: Debugging, WPF, exception handling, windbg

**مشکل:** نگارش 1.0.808.0 برنامه‌ی DNTProfiler بر روی سایر سیستم‌ها، هنوز به مرحله‌ی نمایش نرسیده، کرش می‌کند. علت چیست؟

این نگارش بر روی سیستم من مشکلی نداشت ولی پس از چند گزارش عدم امکان اجرای آن بر روی سایر سیستم‌ها، یک ماشین مجازی ویندوز 8.1 را تهیه و برنامه را بر روی آن اجرا کردم. بله ... برنامه هنوز به مرحله‌ی نمایش نرسیده، محو می‌شد. در این مرحله‌ی ابتدایی امکان تهیه‌ی لاگ استثنای حاصل توسط برنامه وجود نداشت و تنها این خطا در event viewer ویندوز (Computer management -> event viewer -> windows logs -> application) قابل مشاهده بود:

```
Application: DNTProfiler.exe
Framework Version: v4.0.30319
Description: The process was terminated due to an unhandled exception.
Exception Info: System.Windows.Markup.XamlParseException
Stack:
at System.Windows.Markup.WpfXamlLoader.Load
```

متاسفانه از این استثنای لاگ شده‌ی در لاگ‌های ویندوز، اطلاعات مفیدی را نمی‌توان دریافت کرد. عنوان کرده XamlParse برنامه را نمی‌تواند آغاز کند.
روش حل این نوع مشکلات و بررسی آن‌ها در خارج از VS.NET، با استفاده از برنامه‌ی معروف WinDBG مایکروسافت میسر است.
دو نگارش 32 بیتی و 64 بیتی آن‌را از اینجا می‌توانید دریافت کنید:
http://codemachine.com/downloads.html

پس از دریافت، بهتر است نسخه‌ی 32 بیتی آن‌را اجرا کنید، زیرا برای دیباگ برنامه‌های دات نت این نسخه است که امکان بارگذاری فایل C:\Windows\Microsoft.NET\Framework\v4.0.30319\sos.dll را دارد.

تا اینجا فرض بر این است که نسخه‌ی 32 بیتی windbg را دریافت و اجرا کرده‌اید. در ادامه از منوی File آن گزینه‌ی Open executable را انتخاب کرده و فایل exe برنامه را به آن معرفی کنید.

```
CommandLine: C:\Tests\DNTProfiler.1.0.808.0\DNTProfiler.exe
Symbol search path is: srv*
Executable search path is:
ModLoad: 00150000 00176000 DNTProfiler.exe
ModLoad: 76f30000 77098000 ntdll.dll
ModLoad: 72ee0000 72f36000 C:\Windows\SysWOW64\MSCOREE.DLL
ModLoad: 74be0000 74d20000 C:\Windows\SysWOW64\KERNEL32.dll
ModLoad: 756e0000 757af000 C:\Windows\SysWOW64\KERNELBASE.dll
(684.764): Break instruction exception - code 80000003 (first chance)
eax=00000000 ebx=00000000 ecx=37af0000 edx=00000000 esi=7eb8f000 edi=00000000
eip=76fe2d15 esp=002ff604 ebp=002ff630 iopl=0 nv up ei pl zr na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b efl=00000246
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for ntdll.dll -
ntdll!LdrResolveDelayLoadsFromDll+0xa89:
76fe2d15 cc    int   3
```

برنامه داخل این دیباگر اجرا شده و در همینجا نیز به پایان می‌رسد. در ادامه‌ی کار در textbox ذیل این گزارش، دستور g را صادر کرده و enter کنید.

```
0:000> g
ModLoad: 757b0000 75827000 C:\Windows\SysWOW64\ADVAPI32.dll
ModLoad: 05080000 05102000 Newtonsoft.Json.dll
(684.764): C++ EH exception - code e06d7363 (first chance)
(684.764): C++ EH exception - code e06d7363 (first chance)
(684.764): C++ EH exception - code e06d7363 (first chance)
(684.764): CLR exception - code e0434352 (first chance)
First chance exceptions are reported before any exception handling.
```

```
This exception may be expected and handled.
eax=002fd0a0 ebx=00000005 ecx=00000005 edx=00000000 esi=002fd164 edi=00000001
eip=756f3d67 esp=002fd0a0 ebp=002fd0f8 iopl=0 nv up ei pl nz ac pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b efl=00200216
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for
C:\Windows\SysWOW64\KERNELBASE.dll -
KERNELBASE!RaiseException+0x48:
756f3d67 8b4c2454    mov   ecx,dword ptr [esp+54h] ss:002b:002fd0f4=3fce1188
```

فرمان g سبب می‌شود تا کار دیباگ برنامه ادامه پیدا کند و به اولین CLR exception رسیده و متوقف شود. در ادامه نیاز است تا با صدور فرمان sxe clr، دیباگ clr را فعال کرده و سپس مجددا دستور g را برای ادامه‌ی دیباگ صادر کنیم.

```
0:000> sxe clr
0:000> g
(684.764): C++ EH exception - code e06d7363 (first chance)
(684.764): CLR exception - code e0434352 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=002fe2a0 ebx=00000005 ecx=00000005 edx=00000000 esi=002fe368 edi=00000001
eip=756f3d67 esp=002fe2a0 ebp=002fe2fc iopl=0 nv up ei pl nz ac pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b efl=00200216
KERNELBASE!RaiseException+0x48:
756f3d67 8b4c2454    mov   ecx,dword ptr [esp+54h] ss:002b:002fe2f4=3fce2388
```

بنابراین ابتدا دستور sxe clr را صادر و enter و سپس g و enter. مجددا برنامه پس از رسیدن به یک CLR exception متوقف می‌شود.

اکنون می‌خواهیم پیام استثنای واقعی دات نت را مشاهده کنیم. به همین جهت ابتدا دستور loadby sos clr! و سپس دستور CLRStack! را صادر می‌کنیم:

```
0:000> !loadby sos clr
0:000> !CLRStack
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for
C:\Windows\Microsoft.NET\Framework\v4.0.30319\clr.dll -
PDB symbol for clr.dll not loaded
OS Thread Id: 0x764 (0)
Child SP IP Call Site
002fe3f4 756f3d67 [GCFrame: 002fe3f4]
002fe430 756f3d67 [GCFrame: 002fe430]
002fe528 756f3d67 [GCFrame: 002fe528]
002fe544 756f3d67 [HelperMethodFrame_2OBJ: 002fe544]
System.RuntimeTypeHandle.CreateInstance(System.RuntimeType, Boolean, Boolean, Boolean ByRef,
System.RuntimeMethodHandleInternal ByRef, Boolean ByRef)
DNTProfiler.App.Main()
002ff190 720e2652 [GCFrame: 002ff190]
```

در اینجا کار بررسی stack trace برنامه پایان می‌یابد. اکنون با صدور دستور PrintException! می‌توان علت اصلی استثناء را مشاهده کرد:

```
0:000> !PrintException
Exception object: 021e6f0c
Exception type: System.Reflection.TargetInvocationException
Message: Exception has been thrown by the target of an invocation.
InnerException: System.IO.FileNotFoundException, Use !PrintException 021e6950 to see more.
StackTrace (generated):
<none>
StackTraceString: <none>
HResult: 80131604
0:000> !PrintException 021e6950
Exception object: 021e6950
Exception type: System.IO.FileNotFoundException
Message: Could not load file or assembly 'System.Net.Http.Formatting, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35' or one of its dependencies. The system cannot find the file specified.
InnerException: <none>
StackTrace (generated):
SP IP Function
00000000 00000001 DNTProfiler!DNTProfiler.Services.SelfHostConfig.getHostConfiguration(System.String,
Boolean)+0x2
002FE298 006B4E54 DNTProfiler!DNTProfiler.Services.SelfHostConfig.OpenWait(System.String, Boolean)+0x14
002FE2B0 006B4A3F DNTProfiler!DNTProfiler.ViewModels.MainWindowViewModel.doStart(System.String)+0x1f
```

```
002FE2C0 006B368D
DNTProfiler!DNTProfiler.ViewModels.MainWindowViewModel..ctor(DNTProfiler.Common.Mvvm.ICommonDialogsServ
ice, DNTProfiler.Common.Controls.DialogManagement.Contracts.IDialogManager)+0xcd
002FE2D8 006B04C9 DNTProfiler!DNTProfiler.MainWindow..ctor()+0x91

StackTraceString: <none>
HResult: 80070002
```

همانطور که مشاهده می‌کنید، عنوان کرده‌است که فایل System.Net.Http.Formatting قابل بارگذاری نیست. بله ... این فایل جزو بسته‌ی نگارش اول برنامه نبود و به نظر علت عدم کرش آن، نصب این فایل در GAC سیستم توسط نصاب‌های ASP.NET MVC بوده‌است (نگارش‌های 3 و 4 آن). بنابراین اگر کسی این فایل را در GAC سیستم خود نداشته باشد، قادر به اجرای برنامه نخواهد بود.

روش حل این مشکل، مراجعه به خواص پروژه، یافتن اسمبلی System.Net.Http.Formatting در لیست ارجاعات برنامه و سپس true کردن خاصیت copy to local آن است. به این ترتیب این اسمبلی در کنار فایل اجرایی برنامه کپی خواهد شد.

بنابراین خلاصه‌ی عملیات یافتن علت اصلی کرش برنامه در خارج از ویژوال استودیو به صورت ذیل است:

الف) اجرای نسخه‌ی 32 بیتی برنامه‌ی windbg

ب) انتخاب منوی File آن و سپس باز کردن فایل اجرایی برنامه توسط گزینه‌ی Open executable

ج) اجرای دستورات ذیل به ترتیب:

```
0:000> g
0:000> sxe clr
0:000> g
0:000> !loadby sos clr
0:000> !CLRStack
0:000> !PrintException
```

**چند نمونه‌ی مشابه برای مطالعه‌ی بیشتر**

Finding CLR exceptions without visual studio

Power of WinDBG: The story of mysterious crash of WPF application