عنوان: Entity Framework و InnerException
نویسنده: ایمان محمدی
تاریخ: ۱۳۹۱/۰۵/۲۰ ۱۳:۵۴
آدرس: www.dotnettips.info
برچسب‌ها: Entity framework, Exception, Tips

در Entity Framework بیشتر استثناها تودرتو هستند و ما باید تمام استثناها رو بررسی کنیم تا به پیغام اصلی خطا برسیم. با استفاده از تکه کد زیر به راحتی می‌تونیم استثناها رو پیمایش کنیم و متن خطا را مشخص کنیم.

```
catch (Exception ex)
                {
                    StringBuilder errorMsg = new StringBuilder();
                    for (Exception current = ex; current != null; current = current.InnerException)
                    {
                        if (errorMsg.Length > 0)
                            errorMsg.Append("\n");

                        errorMsg.Append(current.Message.Replace("See the inner exception for details.",
string.Empty));
                    }
                    // log
                    errorMsg.ToString();
                }
```

برای استفاده در قسمت‌های مختلف برنامه یک متد الحاقی مانند زیر تعریف می‌کنیم:

```
public static string ExceptionToString(this Exception ex)
        {
            StringBuilder errorMsg = new StringBuilder();
            for (Exception current = ex; current != null; current = current.InnerException)
            {
                if (errorMsg.Length > 0)
                    errorMsg.Append("\n");
                errorMsg.Append(current.Message.
                    Replace("See the inner exception for details.", string.Empty));
            }
            return errorMsg.ToString();
        }
```

```
catch (Exception ex)
                {
                    // log
                    ex.ExceptionToString();
                }
```

نظرات خوانندگان

نویسنده: f.beigirad
تاریخ: ۱۳۹۲/۰۵/۰۴ ۱۵:۱۳

من که نتونستم از کد بالا استفاده کنم.

لینک رو ببینید : http://barnamenevis.org/showthread.php?410767

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۵/۰۴ ۱۸:۳۹

نتونستید یعنی چکار کردید دقیقا؟ در خطای شما هم InnerException -> UpdateException بوده مثل توضیحات بالا. یعنی اول UpdateException صادر میشه، بعد باید محتویات InnerException اون رو بررسی کرد تا به خطای اصلی رسید مثل کدهای فوق.

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: **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