نوشته‌ی مسعود درویشیان

لینک مستقیم این مطلب در وب‌تارگت

نگاهی دقیق‌تر به Exception Class

تا به این‌جا، ما exception ها را می‌گرفتیم اما هیچ‌کاری با شیء exception نمی‌کردیم. همان‌طور که پیش‌تر توضیح داده شد، قسمت catch به شما اجازه می‌دهد تا exception type و variable را مشخص کنید. Variable شامل یک reference متصل به exception object است. از آن‌جایی که تمامی exception ها از Exception Class ارث‌بری کرده‌اند، همه‌ی آن‌ها به اعضای Exception Class دسترسی دارند. در این‌جا با تعدادی از مفیدترین اعضای Exception Class آشنا خواهید شد.

Exception Class تعدادی property دارد که سه عدد از مهم‌ترین آن‌ها Message، StackTrace و TargetSite هستند. این property ها هرسه read-only هستند. Message شامل یک رشته است که ماهیت خطا را شرح می‌دهد. StackTrace شامل یک رشته است که این رشته شامل فراخوانی‌هایی است که منجر به خطا شده‌اند. TargetSite شامل یک شیء بوده که مشخص‌کننده متد تولیدکننده‌ی exception است.

Exception Class همچنین شامل چندین متد است. یکی از آن متدها که قبلاً استفاده کرده‌اید، ()ToString است. این متد یک رشته را return می‌کند که خطا شرح می‌دهد. به‌عنوان مثال، هنگامی‌که یک exception توسط Console.WriteLine() نمایش داده شود، متد ()ToString به‌طور خودکار فراخوانی خواهد شد.

برنامه‌ی زیر این property ها و متدها را نشان می‌دهد:

```
using System;
class ExcTest
{
    public static void GenException()
    {
        int[] nums = new int[4];

        Console.WriteLine("Before exception is generated.");

        // Generate an index out-of-bounds exception.
        for (int counter = 0; counter < 10; counter++)
        {
            nums[counter] = counter;
```

```
            Console.WriteLine("nums[{0}]: {1}", counter, nums[counter]);
        }

        Console.WriteLine("this won't be displayed");
    }
}
class Program
{
    static void Main(string[] args)
    {
        try
        {
            ExcTest.GenException();
        }
        catch (IndexOutOfRangeException exc)
        {
            Console.WriteLine("Standard message is: ");
            Console.WriteLine(exc); // calls ToString()
            Console.WriteLine("Stack trace: " + exc.StackTrace);
            Console.WriteLine("Message: " + exc.Message);
            Console.WriteLine("TargetSite: " + exc.TargetSite);
        }
        Console.WriteLine("After catch block.");
    }
}

/* Output

Before exception is generated.
nums[0]: 0
nums[1]: 1
nums[2]: 2
nums[3]: 3
Standard message is:
System.IndexOutOfRangeException: Index was outside the bounds of the array.
at ExcTest.GenException()
at UseExcept.Main()
Stack trace: at ExcTest.GenException()
at UseExcept.Main()
Message: Index was outside the bounds of the array.
TargetSite: Void GenException()
After catch block.

*/
```

خروجی برنامه‌ی بالا گویای مطلب فوق است.

System namespace شامل تعدادی exception توکار (built-in) و استاندارد است که همه‌ی آن‌ها (چه به‌صورت مستقیم و چه به‌صورت غیرمستقیم) از SystemException ارث‌بری کرده‌اند.

در زیر لیستی از پر استفاده‌ترین exception های استاندارد را می‌بینید:

```
/* More commonly used standard exceptions

Exception: ArrayTypeMismatchException
```

```
 * Meaning: Type of value being stored is incompatible
   with the type of the array.

Exception: DivideByZeroException
 * Meaning: Division by zero attempted.

Exception: IndexOutOfRangeException
 * Meaning: Array index is out-of-bounds.

Exception: InvalidCastException
 * Meaning: A runtime cast is invalid.

Exception: OutOfMemoryException
 * Meaning: Insufficient free memory exists to continue program
            execution. For example, this exception will be thrown if
            there is not sufficient free memory to create an object
            via new.

Exception: OverflowException
 * Meaning: An arithmetic overflow occurred.

Exception: NullReferenceException
 * Meaning: An attempt was made to operate on a null reference—that
            is, a reference that does not refer to an object.

*/
```

Exception های بالا همه واضح هستند و نام آن‌ها بیان کننده‌ی معنای آن‌هاست. NullReferenceException هنگامی پرتاب می‌شود که قصد داشته باشید از یک reference استفاده کنید درحالی‌که null است. مثل هنگامی‌که بخواهید یک متد را از یک null reference فراخوانی کنید. Null reference عبارت است از reference ای که به هیچ شیءای وصل نیست. یک راه برای ساختن null reference این است که یک reference را برابر با مقدار null (با استفاده از کلمه‌ی کلیدی null) قرار دهید.

به مثال زیر که NullReferenceException را شرح می‌دهد دقت کنید:

```
// Use the NullReferenceException.
using System;
class X
{
    int x;
    public X(int a)
    {
        x = a;
    }

    public int Add(X o)
    {
        return x + o.x;
    }
}
// Demonstrate NullReferenceException.
class NREDemo
```

```
{
    static void Main()
    {
        X p = new X(10);
        X q = null; // q is explicitly assigned null
        int val;

        try
        {
            val = p.Add(q); // this will lead to an exception
        }
        catch (NullReferenceException)
        {
            Console.WriteLine("NullReferenceException!");
            Console.WriteLine("fixing...\n");

            // Now, fix it.
            q = new X(9);
            val = p.Add(q);
        }

        Console.WriteLine("val is {0}", val);
    }
}

/* Output

NullReferenceException!
fixing...

val is 19

*/
```

برنامه‌ی بالا یک کلاس به اسم X دارد که شامل یک عضو به اسم x و یک متد Add() است که یک شیء از جنس کلاس X می‌گیرد و آن را با متغیر x درون کلاس جمع می‌کند. در متد Main() دو شیء از کلاس X ساخته شده که یکی از آن‌ها (p) مقداردهی شده و دیگری (q) برابر با null است. سپس p.Add() با آرگومان q فراخوانی می‌شود. از آن‌جایی که q به هیچ شیءای اشاره نمی‌کند، NullReferenceException تولید شده و در بلوک مربوطه handle می‌شود.

ارث‌بری از Exception Class

اگرچه exception های built-in سی‌شارپ اکثر خطاهای رایج را handle می‌کنند، اما مکانیزم exception-handling در سی‌شارپ به این خطاها محدود نیست. در واقع، بخشی از قدرت رویکرد سی‌شارپ به exception ها، توانایی handle کردن exception type هایی است که خودتان می‌سازید. شما می‌توانید exception های خودتان را برای handle کردن خطاهای به‌وجود آمده بسازید (custom exception). ساختن یک exception بسیار ساده است، تنها کافی است یک

کلاس تعریف کنید و آن کلاس از Exception ارث‌بری کرده باشد. کلاس مشتق شده‌ی شما در واقع نیازی برای اجرای هیچ چیزی ندارد. هنگامی‌که کلاس‌های شما از Exception ارث‌بری می‌کنند، وجود آن‌ها در type system موجب می‌شود تا بتوانید از آن‌ها به‌عنوان exception استفاده کنید. در گذشته custom exception ها از ApplicationException ارث‌بری می‌کردند اما اکنون دیگر مایکروسافت این کار را پیشنهاد نمی‌کند بلکه پیشنهاد مایکروسافت ارث‌بری از Exception Class است.

exception هایی که شما می‌سازید شامل property و method هایی است که Exception class برای آن فراهم می‌سازد. همچنین می‌توانید تعدادی از اعضای آن را در exception class ای که خود ساخته‌اید، override کنید.

هنگامی‌که exception class خودتان را می‌سازید، معمولاً می‌خواهید که کلاس شما از تمامی constructor های موجود در کلاس Exception (والد) پشتیبانی کند. برای custom class exception های کوچک، این کار بسیار راحت است زیرا می‌توانید به‌سادگی، argument های مربوط به هر constructor مرتبط با کلاس Exception (والد) را از طریق کلمه‌ی کلیدی base ارسال کنید.

کلاس Exception شامل constructor های زیر است:

```
public Exception()
public Exception(string message)
public Exception(string message, Exception innerException)
protected Exception(System.Runtime.Serialization.SerializationInfo info,
                    System.Runtime.Serialization.StreamingContext context)
```

البته تنها نیاز به نوشتن آن constructor هایی دارید که قصد دارید در برنامه‌تان از آن‌ها استفاده کنید.

در مثال زیر، کلاس RangeArray از آرایه‌های یک بعدی از نوع int پشتیبانی می‌کند که index شروع و پایان آن‌ها توسط کاربر مشخص می‌شود. برای مثال، آرایه‌ای از بازه‌ی ۵- تا ۲۷ برای RangeArray کاملاً صحیح است.

در مثال زیر، به چگونه‌گی ساخت custom exception و نحوه‌ی استفاده از آن توجه کنید:

```
// Use a custom Exception for RangeArray errors.
using System;
// Create a RangeArray exception.
class RangeArrayException : Exception
{
    /* Implement all of the Exception constructors. Notice that
    the constructors simply execute the base class constructor.
    Because RangeArrayException adds nothing to Exception,
    there is no need for any further actions. */
```

```
    public RangeArrayException() : base() { }
    public RangeArrayException(string message) : base(message) { }
    public RangeArrayException(string message, Exception innerException) :
        base(message, innerException) { }
    protected RangeArrayException(
    System.Runtime.Serialization.SerializationInfo info,
    System.Runtime.Serialization.StreamingContext context) :
        base(info, context) { }

    // Override ToString for RangeArrayException.
    public override string ToString()
    {
        return Message;
    }
}
// An improved version of RangeArray.
class RangeArray
{
    // Private data.
    int[] a; // reference to underlying array
    int lowerBound; // smallest index
    int upperBound; // largest index

    // An auto-implemented, read-only Length property.
    public int Length { get; private set; }

    // Construct array given its size.
    public RangeArray(int low, int high)
    {
        high++;
        if (high <= low)
        {
            throw new RangeArrayException("Low index not less than high.");
        }
        a = new int[high - low];
        Length = high - low;
        lowerBound = low;
        upperBound = --high;
    }

    // This is the indexer for RangeArray.
    public int this[int index]
    {
        // This is the get accessor.
        get
        {
            if (ok(index))
            {
                return a[index - lowerBound];
            }
            else
            {
                throw new RangeArrayException("Range Error.");
            }
        }
        // This is the set accessor.
        set
        {
            if (ok(index))
            {
```

```
                a[index - lowerBound] = value;
            }
            else throw new RangeArrayException("Range Error.");
        }
    }

    // Return true if index is within bounds.
    private bool ok(int index)
    {
        if (index >= lowerBound & index <= upperBound) return true;
        return false;
    }
}
// Demonstrate the index-range array.
class RangeArrayDemo
{
    static void Main()
    {
        try
        {
            RangeArray ra = new RangeArray(-5, 5);
            RangeArray ra2 = new RangeArray(1, 10);

            // Demonstrate ra.
            Console.WriteLine("Length of ra: " + ra.Length);
            for (int i = -5; i <= 5; i++)
                ra[i] = i;
            Console.Write("Contents of ra: ");
            for (int i = -5; i <= 5; i++)
                Console.Write(ra[i] + " ");
            Console.WriteLine("\n");

            // Demonstrate ra2.
            Console.WriteLine("Length of ra2: " + ra2.Length);
            for (int i = 1; i <= 10; i++)
                ra2[i] = i;
            Console.Write("Contents of ra2: ");
            for (int i = 1; i <= 10; i++)
                Console.Write(ra2[i] + " ");
            Console.WriteLine("\n");
        }
        catch (RangeArrayException exc)
        {
            Console.WriteLine(exc);
        }

        // Now, demonstrate some errors.
        Console.WriteLine("Now generate some range errors.");
        // Use an invalid constructor.
        try
        {
            RangeArray ra3 = new RangeArray(100, -10); // Error
        }
        catch (RangeArrayException exc)
        {
            Console.WriteLine(exc);
        }

        // Use an invalid index.
        try
        {
```

```
            RangeArray ra3 = new RangeArray(-2, 2);
            for (int i = -2; i <= 2; i++)
                ra3[i] = i;
            Console.Write("Contents of ra3: ");
            for (int i = -2; i <= 10; i++) // generate range error
                Console.Write(ra3[i] + " ");
        }
        catch (RangeArrayException exc)
        {
            Console.WriteLine(exc);
        }
    }
}

/* Output

Length of ra: 11
Contents of ra: -5 -4 -3 -2 -1 0 1 2 3 4 5

Length of ra2: 10
Contents of ra2: 1 2 3 4 5 6 7 8 9 10

Now generate some range errors.
Low index not less than high.
Contents of ra3: -2 -1 0 1 2 Range Error.

*/
```

در مثال بالا، هنگامی‌که range error اتفاق می‌افتد، کلاس RangeArray یک exception از نوع RangeArrayException پرتاب می‌کند. دقت کنید که در سه قسمت کلاس RangeArray ممکن است خطا به‌وجود آید: در get indexer accessor، در set indexer accessor و در RangeArray constructor. همان‌طور که می‌بینید، constructor های کلاس RangeArray در بدنه‌شان هیچ‌گونه کدی قرار نگرفته است و فقط argument ها را از طریق base به Exception می‌فرستند.

به مثال زیر توجه کنید:

```
using System;
public class ReThrowDemo
{
    public static void Main()
    {
        try
        {
            Console.WriteLine("Trying in Main() method");
            MethodA();
        }
        catch (Exception ae)
        {
            Console.WriteLine("Caught in Main() method --\n {0}",
            ae.Message);
        }
        Console.WriteLine("Main() method is done");
```

```
    }
    public static void MethodA()
    {
        try
        {
            Console.WriteLine("Trying in method A");
            MethodB();
        }
        catch (Exception)
        {
            Console.WriteLine("Caught in method A");
            throw;
        }
    }
    public static void MethodB()
    {
        try
        {
            Console.WriteLine("Trying in method B");
            MethodC();
        }
        catch (Exception)
        {
            Console.WriteLine("Caught in method B");
            throw;
        }
    }
    public static void MethodC()
    {
        Console.WriteLine("In method C");
        throw (new Exception("This came from method C"));
    }
}

/* Output

Trying in Main() method
Trying in method A
Trying in method B
In method C
Caught in method B
Caught in method A
Caught in Main() method --
 This came from method C
Main() method is done

*/
```

در این برنامه، فراخونی متدها و در نهایت پرتاب شدن exception، موجب به‌جود آمدن خروجی بالا می‌شود.