Excepciones

Las excepciones ofrecen varias ventajas respecto a otros métodos de notificación de error, como los códigos devueltos (órdenes **return**) ya que ningún error pasa desapercibido (**las excepciones no pueden ser ignoradas**) y no tienen por qué tratarse en el punto en que se producen. Los valores no válidos no se siguen propagando por el sistema. No es necesario comprobar los códigos devueltos. Es muy sencillo agregar código de control de excepciones para aumentar la confiabilidad del programa.

Una excepción es cualquier situación de error o comportamiento inesperado que encuentra un programa en ejecución. Las excepciones se pueden producir a causa de un error en el código o en el código al que se llama (como una biblioteca compartida), que no estén disponibles recursos del sistema operativo, condiciones inesperadas que encuentra *Common Language Runtime* (por ejemplo, código que no se puede comprobar), etc. La aplicación se puede recuperar de algunas de estas condiciones, pero de otras no.

Las excepciones pueden generarse en un proceso o hebra de nuestra aplicación (con la sentencia **throw**) o pueden provenir del entorno de ejecución de la plataforma .NET.

En .NET Framework, una excepción es un objeto derivado de la clase **Exception**. El mecanismo de control de excepciones en C# es muy parecido al de C++ y Java: la excepción se inicia en un área del código en que se produce un problema. La excepción asciende por la pila hasta que la aplicación la controla o el programa se detiene.

El proceso es el siguiente:

- La sentencia **throw** lanza una excepción (una instancia de una clase derivada de **System.Exception**, que contiene información sobre la excepción: **Message**, **StackTrace**, **HelpLink**, **InnerException**...).
- El bloque **try** delimita código que podría generar una excepción.
- El bloque **catch** indica cómo se manejan las excepciones. Se puede relanzar la excepción capturada o crear una nueva si fuese necesario. Se pueden especificar distintos bloques **catch** para capturar distintos tipos de excepciones. En ese caso, es recomendable poner primero los más específicos (para asegurarnos de que capturamos la excepción concreta).
- El bloque **finally** incluye código que siempre se ejecutará (se produzca o no una excepción). Este bloque es opcional.

El control de excepciones que hemos explicado, es aplicable a cualquier tipo de aplicación de .Net, por ejemplo, aplicación de consola, librería de clase, aplicación web, web service, etc.

En la sección laboratorio de esta clase, puede ejercitar con varios ejercicios sobre excepciones.

Manejo de excepciones

En una aplicación ASP.NET pueden producirse básicamente dos tipos de errores:

- **Errores en la aplicación**: que se controlan creando excepciones en .NET.
- **Errores de servidor**: son errores de IIS en el servidor.

Errores a nivel de aplicación

Supongamos que desde una página ASP.NET intentamos acceder a una clase de negocio que a su vez accede a una clase de acceso a datos y esta última, consulta un servidor de base de datos que se encuentra fuera de línea. Evidentemente ADO.NET producirá una excepción y si la misma no está capturada o controlada desde el código, el usuario recibirá un mensaje de error. Vemos un ejemplo de un mensaje de error de esto tipo:

Uso del Web.Config

Es posible indicar en el archivo de configuración Web.Config, a qué página debe redirigirse al usuario en caso de ocurrir un error de aplicación. Incluso es posible discriminar por tipo de error IIS según un código de error, y de esta forma tener algunas páginas de error genéricas y ya armadas para mostrarle la situación de excepción al usuario. Vemos un ejemplo de código para esta directiva en el web.config:

```
<customErrors defaultRedirect="GenericError.htm" mode="On">
        <error statusCode="404" redirect="InternalError.htm"/>
 </customErrors>
```

Nota: en el siguiente link está el detalle de los Códigos oficiales de Estado HTTP, y la actualización de dichos códigos:

http://librosweb.es/tutorial/los-codigos-de-estado-de-http/

https://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml

Como guía básica la clasificación de dichos errores sigue las siguientes categorías:

1xx: Informational - Request received, continuing process
2xx: Success - The action was successfully received, understood, and accepted
3xx: Redirection - Further action must be taken in order to complete the request
4xx: Client Error - The request contains bad syntax or cannot be fulfilled
5xx: Server Error - The server failed to fulfill an apparently valid request

Uso del Global.ASAX

Global.asax es un archivo opcional usado en las aplicaciones web de ASP.NET para declarar y manejar eventos y objetos a nivel de aplicación y de sesión. El archivo Global.asax reside en el directorio virtual raíz de una aplicación ASP.NET en IIS. En tiempo de ejecución, antes de la llegada de la primera solicitud, Global.asax es analizado y compilado en una clase dinámicamente generada del .NET Framework. ASP.NET está configurado para que cualquier solicitud directa del Global.asax sea automáticamente rechazada, los usuarios externos no pueden ver o descargar el código que hay en él.

El código para manejar eventos de aplicación (tales como el inicio y el fin de una aplicación) residen en el Global.asax. Tales códigos de eventos no pueden residir en una página web o en un servicio web, pues durante el inicio o el fin de la aplicación, su código no ha sido cargado (o descargado). Global.asax es también usado para declarar datos que están disponibles en diferentes solicitudes de la aplicación y a través de diferentes sesiones del navegador. Este proceso es conocido como administración del estado de sesión y aplicación, mencionado en una clase anterior de este curso.

Al agregar el archivo Global.asax a una aplicación web, disponemos varios eventos a los que podemos responder. Mostramos en el siguiente código todos los eventos disponibles en este archivo:

```
using System; using
System.Collections.Generic;
using System.Linq; using
System.Web; using
System.Web.Security;
using System.Web.SessionState;
 namespace
IntegradorASP
{
    public class Global : System.Web.HttpApplication
    {

        protected void Application_Start(object sender, EventArgs e)
        {
```

```
        }

        protected void Session_Start(object sender, EventArgs e)
        {
        }
        protected void Application_BeginRequest(object sender, EventArgs e)
        {
        }
        protected void Application_AuthenticateRequest(object sender, EventArgs
 e)         {
        }

        protected void Application_Error(object sender, EventArgs e)
        {
        }

        protected void Session_End(object sender, EventArgs e)
{
        }

        protected void Application_End(object sender, EventArgs e)
        {
        }
    }
```

Para controlar los errores de la aplicación, podríamos escribir código en el evento **Application_Error():**

```
protected void Application_Error(object sender, EventArgs e)
{
     Session["Error"] = Server.GetLastError();
     Server.Transfer("~/Error.aspx");
}
```

La primera línea guarda la excepción en una variable de sesión llamada "Error" y la segunda transfiere a la página Error.aspx en dónde recuperamos la excepción para informar un mensaje al usuario. En dicha página, tendríamos un código similar al siguiente para mostrar el mensaje de la excepción:

```
Exception ex = (Exception)Session["Error"];
this.lblMensaje.Text = ex.Message;
Session["Error"] = null;
```

Errores a nivel de página

Supongamos que en el código behind de una página ASP.NET se intenta convertir a decimal este string: "$10". Al no reconocer el símbolo $ la librería encargada de la conversión producirá una excepción.

Esta excepción puede ser capturada a través de dos mecanismos diferentes:

- Usando el evento Page_Error()
- Usando el bloque de código try/catch

Uso del evento Page_Error()

Toda excepción no específicamente capturada con un bloque try/catch puede ser capturada utilizando dicho evento:

```
private void Page_Error(object sender, EventArgs e)
{
      Exception ex = Server.GetLastError();
      Session["Error"] = ex;
      Server.ClearError();
      Server.Transfer("~/MostrarError.aspx");
}
```

Bloque try/catch

Por supuesto es posible utilizar el bloque try/catch para capturar excepciones a nivel de instrucción, de misma manera que se hace en cualquier tipo de aplicación de .Net. Verá un ejemplo con una aplicación web aplicando try/catch.