



Diapositiva 1




Día 1:
Descripción general
de C#




1

Diapositiva 2




Objetivos para el día

- Al final de este día, usted podrá:
 - Enumerar de los principales elementos de la Plataforma .Net
 - Describir la estructura del programa C#
 - Realizar una aplicación de consola sencilla
 - Documentar una aplicación
 - Compilar, ejecutar y depurar un programa





2

Diapositiva 3




Contenidos del día

- 1. Descripción general de .Net Framework
- 2. Descripción general del lenguaje C#



3



Diapositiva 4



1. Descripción general de .Net Framework

Preguntas de la sección

- Preguntas resueltas
 - ¿Qué es .NET Framework?
 - ¿Cuáles son las principales características?
 - ¿Cuáles son los estándares de .Net Framework?

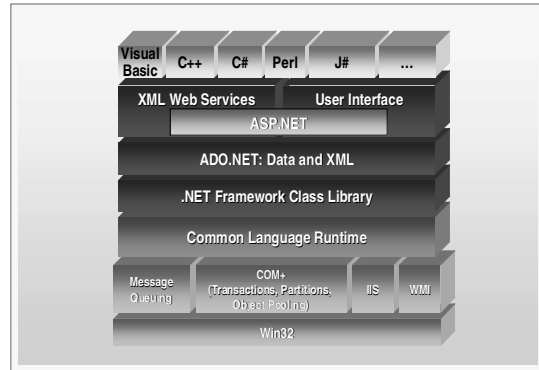


4

Diapositiva 5

.Net Framework

- La combinación de:
 - Framework
 - Motor de ejecución común de los lenguajes
 - Biblioteca de clases
 - ASP.NET
 - Servicios Web
 - .NET Enterprise Servers



La Plataforma .NET de Microsoft® proporciona todas las herramientas y tecnologías requeridas para construir aplicaciones Web distribuidas. Expone un modelo de programación independiente de lenguajes, pero consistente a través de todos los niveles de una aplicación, proporcionando al mismo tiempo una interoperabilidad transparente y una fácil migración de tecnologías existentes. La Plataforma .NET soporta totalmente las tecnologías de Internet neutrales a la plataforma y basadas en estándares, incluyendo el Protocolo de transferencia de hipertexto (HTTP), el Lenguaje de marcación extensible (XML), y SOAP.

Diapositiva 6

Motor de ejecución común de los lenguajes

- Características más importantes

- Sistema de tipo común

- Correlación de tipo de datos.

- Compiladores Justo a tiempo (JIT)

- JIT compila MSIL en código nativo
 - Altamente optimizado para plataformas o dispositivos

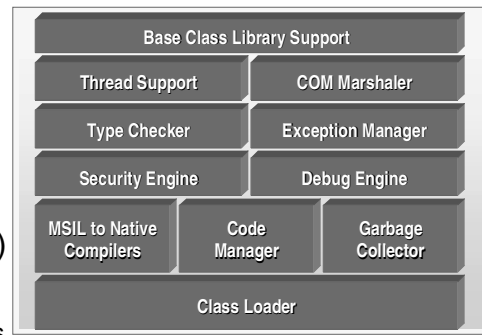
- Recolector de residuos

- Seguridad basada en permisos y políticas

- Excepciones

- Subprocesos

- Diagnóstico y perfil



El motor de ejecución común de los lenguajes es el componente clave de .Net Framework, ya que proporciona todos los servicios para desarrollar y ejecutar aplicaciones de cualquier tipo. Este ambiente de ejecución administra la ejecución del código y proporciona servicios que simplifican el proceso de desarrollo.

El motor de ejecución común de los lenguajes facilita el diseño de componentes y aplicaciones cuyos objetos interactúan entre los lenguajes. Los objetos escritos en diferentes lenguajes se pueden comunicar entre sí, y se pueden integrar muy bien sus comportamientos.

Por ejemplo, puede definir una clase y después utilizar un lenguaje diferente para derivar una clase de la original o llamar el método de la clase original.

También puede pasar una instancia de una clase a un método de una clase escrita en un lenguaje diferente.

Esta integración entre lenguajes es posible debido a que los compiladores de lenguaje y las herramientas que están enfocadas en el motor de ejecución utilizan un sistema de tipo común definido por el tiempo de ejecución y siguen las reglas del tiempo de ejecución para definir nuevos tipos, al igual que para crear, utilizar, persistir y unirse a los tipos.

Diapositiva 7

Infraestructura de lenguaje común

- Permitir la interoperabilidad de los lenguajes
- Estándar ISO
 - ISO/IEC 23270 (C#)
 - ISO/IEC 23271 (CLI)
- Definir un Sistema de tipo común
- Definir las características de un objeto común



7

Para interactuar completamente con otros objetos, independientemente del lenguaje con el cual se implementaron, los objetos deben exponer a los invocadores únicamente aquellas funciones que son comunes a todos los lenguajes con los cuales deben interoperar.

Por esta razón, se ha definido un conjunto de funciones de lenguajes, llamado la Especificación de Lenguaje Común (CLS), que incluye funciones básicas de lenguaje requeridas en muchas aplicaciones.

Las reglas CLS definen un subconjunto del **sistema de tipo común**; esto es, todas las reglas que aplican al sistema de tipo común aplican a la CLS, excepto en donde se definen reglas más estrictas en la CLS.

La CLS ayuda a mejorar y asegurar la interoperabilidad del lenguaje al definir un conjunto de funciones en las que se pueden basar los desarrolladores, ya que están disponibles en una gran variedad de lenguajes. La CLS también establece los requerimientos para el cumplimiento con sí misma; esto ayuda a determinar si


su código administrado cumple con la CLS y en qué medida una herramienta dada soporta el desarrollo de un código administrado que utilice las funciones de CLS.

La CLS es una especificación estándar; a finales de diciembre de 2001, la ECMA presentó las normas y TR a ISO/IEC JTC 1 a través del proceso de Fast-Track de ésta última.

En abril de 2003, ISO ratificó las normas como ISO/IEC 23270 (C#), ISO/IEC 23271 (CLI) y ISO/IEC 23272 (CLI TR).

Las especificaciones equivalentes han sido adoptadas como estándares de segunda edición y TR por ECMA. A continuación se encuentra una visión general del trabajo del comité a la fecha.


Diapositiva 8



2. Descripción general de C#

Preguntas de la sección

- Preguntas resueltas
 - ¿Cuál es la estructura principal del programa C#?
 - ¿Cuáles son los tipos de datos principales?
 - ¿Cuáles son las estructuras condicionales y de iteración?
 - ¿Cómo documentar una aplicación?
 - ¿Cómo compilar y depurar un programa?



8

Diapositiva 9

Estructura del programa C#

- Un programa C# es una colección de clases, estructuras y tipos
- Estos elementos pueden residir en uno o varios archivos
- Todo es un objeto
- C# diferencia entre mayúsculas y minúsculas
- Utilizar los espacios de nombre para organizar clases



9

Una aplicación C# es una colección de clases, estructuras y tipos.

Una clase es una conjunto de propiedades y métodos. Estos elementos pueden residir en uno o más archivos, y un archivo puede contener una o todas las clases que integran una aplicación.

En el lenguaje C# todo es un objeto, y ha sido diseñado desde cero para ser un lenguaje realmente orientado a los objetos y basado en componentes.

Al compilar una aplicación C# o cualquier aplicación escrita en un lenguaje que cumpla con la CLS, la aplicación se compila en MSIL.

Este MSIL entonces se compila a instrucciones nativas de CPU cuando la aplicación se ejecuta por primera vez a través del CLR. (En realidad, únicamente las funciones invocadas se compilan la primera vez que se invocan).

Veamos lo que realmente sucede tras bambalinas:

1. Usted escribe código fuente en C#.

2. Después lo compila utilizando el compilador C# (csc.exe) en un archivo, por ejemplo, EXE.
3. El compilador C# da como resultado un código MSIL y un manifiesto en una parte de sólo lectura del archivo EXE que tiene un encabezado PE (ejecutable portátil de Win32) estándar.
4. Cuando se ejecuta la aplicación, el sistema operativo carga el PE, al igual que todas las bibliotecas de vínculos dinámicos dependientes (DLLs).
5. El cargador del sistema operativo salta entonces al punto de entrada dentro del PE, el cual está colocado ahí mediante el compilador C#.
6. Debido a que el código MSIL no se puede ejecutar directamente, debido a que no está en un formato ejecutable por la máquina, el CLR compila el MSIL utilizando el compilador justo a tiempo (JIT) (o JITter) en instrucciones nativas de CPU a medida que procesa el MSIL. La compilación JIT ocurre a medida que se invocan los métodos en el programa. El código ejecutable compilado se pasa a la memoria caché de la máquina y se recompila únicamente si existe algún cambio en el código fuente.

Diapositiva 10

Método principal

- Punto de inicio de una aplicación
- Definición
 - Puede devolver void
 - `static void Main()`
 - Puede devolver int
 - `static int Main()`
 - Puede tomar argumentos
 - `static int Main(string[] args)`



10

Todas las aplicaciones deben tener un punto de inicio en C#, este punto es el método Main, excepto las aplicaciones Web que tienen la página de inicio. El método Main es donde se crean los objetos y se ejecutan otros métodos. El método Main es un método estático que reside dentro de una clase o una estructura. Existen tres formas de declarar un método Main:

Puede devolver un void:

```
static void Main()  
{  
...  
}
```

También puede devolver un int:

```
static int Main()  
{  
...  
return 0;
```

```
}
```

También puede tomar argumentos:

```
static int Main(string[] args)
```

```
{
```

```
    ...
```

```
    return 0;
```

```
}
```

El parámetro del método Main es una matriz de cadena que representa los argumentos de la línea de comando utilizados para invocar el programa.

Diapositiva 11

Comentarios

■ Comentario en línea

`// Línea comentada`

■ Comentario en bloque

`/* Bloque
comentado */`

■ Documentación generada por XML

`///<summary>Resumen</summary>`



11

Existen dos formas de agregar comentarios en el código C#.

Comentario a una línea: los caracteres `//` convierten el resto de la línea en un comentario, por ejemplo:

```
// A "Hello World!" program in C#
```

Comentario a un bloque: de texto colocándolo entre los caracteres `/*` y `*/`, por ejemplo:

```
Programa /* A "Hello World!" en C#.
```

```
Este programa muestra la cadena "Hello World!" en la pantalla. */
```

Diapositiva 12

Ejemplo

```
using System;
namespace Day1
{
    /// <summary>
    /// Introduccion to C# programming - Day 1
    /// </summary>

    class Example
    {
        [STAThread]
        static void Main(string[] args)
        {
        }
    }
}
```

Esta es la estructura básica de un programa C#

Diapositiva 13

Tipos predefinidos

■ Tipos de valores:

- Contienen directamente sus datos
- Cada uno tiene su propia copia de los datos
- Las operaciones en uno no pueden afectar a otros

■ Tipos de referencia:

- Almacena referencias de sus datos (conocidos como objetos)
- Dos variables de referencia se pueden referir al mismo objeto
- Las operaciones en uno pueden afectar al otro

- Todos los tipos se definen en el espacio de nombre System

- Todos los tipos se derivan al final de System.Object

- El tipo de valor se deriva de System.ValueType

13

En el centro de Microsoft .NET Framework se encuentra un sistema de tipo universal llamado Sistema de tipo común .NET (CTS). El sistema de tipo común define la manera en la que se declaran, utilizan y administran los tipos en tiempo de ejecución, y también es una parte importante del soporte al motor de ejecución para la integración entre lenguajes. El sistema de tipo común lleva a cabo las siguientes funciones:

- Establece un marco que permite la integración entre lenguajes, la seguridad de tipos y una ejecución de código de alto rendimiento.
- Proporciona un modelo orientado a objetos que soporta la implementación completa de muchos lenguajes de programación.
- Define las reglas que deben seguir los lenguajes, lo cual ayuda a asegurar que los objetos escritos en diferentes lenguajes pueden interactuar entre sí.

Todos los demás tipos de datos se definen en el espacio de nombre System.
Todos los tipos se derivan en última instancia de System.Object. Los tipos de valor se derivan de System.ValueType.

Los siguientes son tipos integrados: System.SByte, System.Byte, System.Int16, System.UInt16, System.Int32, System.UInt32, System.Int64, System.UInt64, System.Char, System.Single, System.Double, System.Boolean, System.Decimal

Diapositiva 14

Declarar variables

- Declaración simple → tipo de datos + nombre de variable
 - `string ColorName;`
- Declaración múltiple → tipo de datos + nombre de variable 1, nombre de variable 2, nombre de variable “n”
 - `int ItemID, CategoryID, ItemCount`
- No puede utilizar variables no inicializadas

14

Puede declarar una variable local al especificar el tipo de datos seguido por el nombre de variable. Por ejemplo:

```
string ProductName;
```

Puede declarar múltiples variables en una sola declaración al utilizar un separador de coma. Por ejemplo:

```
string ProductName, ProductDescription;
```

En C#, no puede utilizar variables no inicializadas.

Usted utiliza los operadores de asignación para asignar un nuevo valor a la variable. Si ya está declarada una variable, utilice el operador de asignación (=).


Por ejemplo:

```
int cardNumber;  
cardNumber = 44;
```

También puede utilizar las instrucciones de selección para determinar qué código se debe ejecutar y cuándo se debe ejecutar. Por ejemplo:

```
int employeeNumber = 69;
```

Diapositiva 15





Instrucciones condicionales

- **Si la instrucción**
si la (expresión)
statement1
[else
statement2]
- **Instrucción Switch**
switch (switch_expression)
{ case constant-expression:
statement
jump-statement

case constant-expressionN:
statementN
[default]
}

```
if (i == 0 )  
  
    Console.WriteLine("This");  
  
else  
  
    Console.WriteLine("That");  
};
```

```
switch (i)  
  
case 0:  
    Console.WriteLine("This");  
    break;  
case 1:  
    Console.WriteLine("That");  
    break;  
default:  
    Console.WriteLine("Else");  
    break;  
}
```



15

Utilice las instrucciones de selección para determinar qué código se debe ejecutar y cuándo se debe ejecutar. C# incluye dos instrucciones de selección: la instrucción switch que se utiliza para ejecutar el código con base en un valor, y la instrucción "if" que ejecuta el código con base en una condición lógica. La instrucción de selección que se utiliza con mayor frecuencia es "if".

Al utilizar la instrucción switch, puede especificar una expresión que devuelva un valor de número entero y una o más partes del código que se ejecutarán dependiendo del resultado de la expresión. Es similar a utilizar varias instrucciones if/else, pero a pesar de que puede especificar múltiples instrucciones condicionales (posiblemente no relacionadas) con varias instrucciones if/else, una instrucción switch únicamente consiste de la instrucción condicional seguida por todos los resultados que su código pueda manejar.

Diapositiva 16

Instrucciones de iteración

■ while y do while...

while (expresión lógica)
sentencias

```
while ( !MyFile.EOF )  
    Console.Write(MyFile.Read());  
}
```

■ Do until...

sentencias
while (expresión lógica)

```
foreach ( MyObject o in MyObjects )  
    Console.Write(o.Name);  
}
```

■ for

for (inicialización; expresión lógica; paso)
sentencias

```
for (i=0; i< 10; i++ )  
    Console.Write(i);  
}
```

■ For each

For each (tipo en expresión)
sentencias

En C#, las instrucciones while, do/while, for y foreach le permiten llevar a cabo una iteración controlada o en circuito. En cada caso, una instrucción específica simple o compuesta se ejecuta hasta que una expresión lógica resuelve a verdadero, excepto en el caso de la instrucción "foreach", la cual se utiliza para iterar a lo largo de una lista de objetos.

Diapositiva 17

Operaciones básicas de entrada / salida

■ Usar la clase Console

■ Read

- Lee el siguiente carácter del flujo de entrada estándar.

■ ReadLine

- Lee la siguiente línea de caracteres del flujo de entrada estándar.

■ Write

- Escribe la información específica para el flujo de salida estándar.

■ WriteLine

- Escribe los datos especificados, seguidos de un terminador de línea actual, para el flujo de salida estándar.



17

Representa la entrada, la salida y los flujos de error estándar para las aplicaciones de consola. Esta clase no se puede heredar.

La clase Console ofrece un soporte básico para las aplicaciones que lee y escribe caracteres en la consola. Si la consola no existe, como en una aplicación basada en Windows, las escrituras a la consola no se muestran y no se genera una excepción.

Los datos de la consola se lee desde un flujo de entrada estándar; los datos normales hacia la consola se escriben en un flujo de salida estándar; los datos de error hacia la consola se escriben en un flujo de salida de error estándar. Estos flujos se asocian automáticamente con la consola cuando se inicia la aplicación y se presentan como propiedades **In**, **Out** y **Error**.

Diapositiva 18

Documentación XML

- Documentación que utiliza el formato XML
- Los comentarios se inician con tres diagonales (///) seguidas por una etiqueta de documentación XML
- Utilizar la opción de compilador /doc:nombre del archivo para generar el archivo con la documentación
 - Ejemplo: csc MyClass /doc:MyDoc.xml

```
///<summary>  
/// This function serves to calculate the  
/// overall value of the item including all  
/// taxes  
/// </summary>  
/// <remarks>  
/// Tax calculates in CalcTax()  
/// </remarks>  
public decimal GetTotalValue()  
  
}
```

C# proporciona un mecanismo para que los desarrolladores documenten su código utilizando XML. En los archivos de código fuente, las líneas que inicien con /// y que preceden al tipo definido por el usuario tal como clase, delegado o interfaz; un miembro como un campo, evento, propiedad o método, o una declaración de espacio de nombre se pueden procesar como comentarios y colocar en un archivo.

Diapositiva 19

Compilar, ejecutar y depurar

■ Compilar

- La compilación genera un código MSIL
- Utilizar el comando CSC
 - Ejemplo: `csc /out:MyExecutable.exe MyProgram.cs`

■ Ejecutar

- Desde la línea de comandos, escribir el nombre de la aplicación
- Desde Visual Studio .Net, presionar CTRL+F5. Iniciar sin depurar

■ Depurar

- Utilizar los puntos de interrupción en el código
- Desde Visual Studio .Net, presionar F5. Iniciar
- Utilizar Examinadores para evaluar y modificar variables

19

Modificadores comunes del compilador

Puede especificar un número de modificadores del compilador C# al utilizar el comando `csc`. Los siguientes elementos describen los modificadores más comunes.

`/?, /help`

Muestra las opciones del compilador en una salida estándar.

`/out`

Especifica el nombre del ejecutable.

`/main`

Especifica la clase que contiene el método Main (si más de una clase en la aplicación incluye el método Main).

`/optimize`

Habilita y deshabilita el optimizador de códigos.

/warn

Establece un nivel de advertencia del compilador.

/warnaserror

Trata todas las advertencias como errores que abortan la compilación.

/target

Especifica el tipo de aplicación generada.

/checked

Indica si el sobreflujo aritmético generará una excepción en el tiempo de ejecución.

/doc

Procesa comentarios de la documentación para producir un archivo XML.

/debug

Genera la información de depuración.

Diapositiva 20



Resumen de recursos

- ¿Qué es un CLS?
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconwhatiscmonlanguagespecification.asp>
- Estándares de ECMA e ISO/IEC C# y de la infraestructura de lenguaje común <http://msdn.microsoft.com/net/ecma/>
- Infraestructura compartida del lenguaje común de la fuente, versión 1.0
<http://www.microsoft.com/downloads/details.aspx?FamilyId=3A1C93FA-7462-47D0-8E56-8DD34C6292F0&displaylang=en>



Resumen de recursos

- Usar el depurador
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vsdebug/html/vxtskverifyingassertionsinmanagedcode.asp>
- Modelo de objetos del depurador de Visual Studio
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vsdebugext/html/vxoriDebuggerObjectModel.asp>
- Etiquetas de documentación XML
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/csref/html/vclrftagsfordocumentationcomments.asp>
- Los comentarios de XML le permiten desarrollar documentación directamente de sus archivos fuente de Visual Studio .NET
<http://msdn.microsoft.com/msdnmag/issues/02/06/XMLC/default.aspx>

