PROGRAMACIÓN MULTIMEDIA

Y

DISPOSITIVOS MÓVILES

UT1. La plataforma .NET

Departamento de Informática y Comunicaciones

CFGS Desarrollo de Aplicaciones Multiplataforma

Segundo Curso

IES Virrey Morcillo

Villarrobledo (Albacete)

Índice

1. La plataforma .NET

.NET es una plataforma de desarrollo de aplicaciones de código abierto, multiplataforma y gratuita para crear diferentes de aplicaciones. Con .NET, se puede usar múltiples lenguajes de programación, editores y bibliotecas para compilar aplicaciones para web, móvil, escritorio, juegos e IoT.

Hay cuatro implementaciones principales de .NET que Microsoft desarrolla y mantiene activamente:

- .NET Core.
- .NET Framework.
- Mono.
- UWP.

Cada implementación de .NET incluye los siguientes componentes:

- ✓ Uno o varios entornos de ejecución. Ejemplos: CLR para .NET Framework, CoreCLR y CoreRT para .NET Core.
- ✓ Una biblioteca de clases que implementa .NET Standard y puede implementar API adicionales. Ejemplos: biblioteca de clases base de .NET Framework, biblioteca de clases base de .NET Core.
- ✓ Opcionalmente, uno o varios marcos de trabajo de la aplicación. Ejemplos: ASP.NET, Windows Forms y Windows Presentation Foundation (WPF) se incluyen en .NET Framework.
- ✓ Opcionalmente, herramientas de desarrollo. Algunas herramientas de desarrollo se comparten entre varias implementaciones.

1.1. .NET Core

.NET Core es una implementación multiplataforma de .NET diseñada para controlar cargas de trabajo de servidor y en la nube a escala. Se ejecuta en Windows, macOS y Linux. Implementa .NET Standard, de forma que cualquier código que tenga como destino .NET Standard se puede ejecutar en .NET Core. ASP.NET Core se ejecuta en .NET Core.

.NET Core tiene las siguientes características:

- ✓ Multiplataforma: se ejecuta en los sistemas operativos Windows, MacOS y Linux.
- ✓ Coherente en todas las arquitecturas: se ejecuta el código con el mismo comportamiento en varias arquitecturas, como x64, x86 y ARM.
- ✓ Herramientas de línea de comandos: incluye herramientas de línea de comandos fáciles de usar que se usan para el desarrollo local y escenarios de integración continua.
- ✓ Implementación flexible: se puede incluir en la aplicación o se puede instalar de forma paralela a nivel de usuario o de equipo. Se puede usar con contenedores de Docker.
- ✓ Compatible: .NET Core es compatible con .NET Framework, Xamarin y Mono, mediante .NET Standard.
- ✓ Código abierto: la plataforma .NET Core es de código abierto, con licencias de MIT y Apache 2. .NET Core es un proyecto de .NET Foundation.

- ✓ Compatible con Microsoft: .NET Core incluye compatibilidad con Microsoft, como se indica en .NET Core Support (Compatibilidad de .NET Core).

1.2. .NET Framework

.NET Framework es la implementación de .NET original que existe desde 2002. Es el mismo .NET Framework que los desarrolladores existentes de .NET han usado siempre. Las versiones 4.5 y posteriores implementan .NET Standard, de forma que el código que tiene como destino .NET Standard se puede ejecutar en esas versiones de .NET Framework.

Contiene API específicas de Windows adicionales, como API para el desarrollo de escritorio de Windows con Windows Forms y WPF. .NET Framework está optimizado para crear aplicaciones de escritorio de Windows.

El diseño de .NET Framework está enfocado a cumplir los objetivos siguientes:

- ✓ Proporcionar un entorno coherente de programación orientada a objetos, en el que el código de los objetos se pueda almacenar y ejecutar de forma local, distribuida en Internet o de forma remota.
- ✓ Proporcionar un entorno de ejecución de código que reduzca lo máximo posible la implementación de software y los conflictos de versiones.
- ✓ Ofrecer un entorno de ejecución de código que promueva la ejecución segura del mismo, incluso del creado por terceras personas desconocidas o que no son de plena confianza.
- ✓ Proporcionar un entorno de ejecución de código que elimine los problemas de rendimiento de los entornos en los que se utilizan scripts o intérpretes de comandos.
- ✓ Ofrecer al programador una experiencia coherente entre tipos de aplicaciones muy diferentes.
- ✓ Basar toda la comunicación en estándares del sector para asegurar que el código de .NET Framework se puede integrar con otros tipos de código.

1.3. Mono .NET

Mono es una implementación de .NET que se usa principalmente cuando se requiere un entorno de ejecución pequeño. Es el entorno de ejecución que activa las aplicaciones de **Xamarin** en Android, Mac, iOS, tvOS y watchOS, y se centra principalmente en una superficie pequeña. Mono también proporciona juegos creados con el motor de Unity.

Admite todas las versiones de .NET Standard publicadas actualmente. Históricamente, Mono implementaba la API de .NET Framework más grande y emulaba algunas de las funciones más populares en Unix. A veces, se usa para ejecutar aplicaciones de .NET que se basan en estas capacidades en Unix.

Mono se suele usar con un compilador Just-In-Time, pero también incluye un compilador estático completo (compilación Ahead Of Time) que se usa en plataformas como iOS.

1.4. Plataforma universal de Windows (UWP)

UWP es una implementación de .NET que se usa para compilar aplicaciones Windows modernas y táctiles y software para Internet de las cosas (IoT). Se ha diseñado para unificar los diferentes tipos de dispositivos de destino, incluidos equipos, tabletas, phablets, teléfonos e incluso la consola Xbox.

UWP proporciona muchos servicios, como una tienda de aplicaciones centralizada, un entorno de ejecución (AppContainer) y un conjunto de API de Windows para usar en lugar de Win32 (WinRT).

Las aplicaciones pueden escribirse en C++, C#, VB.NET y JavaScript. Al usar C# y VB.NET, .NET Core proporciona las API de .NET.

1.5. .NET Standard

.NET Standard es una especificación formal de las API de .NET que se prevé esté disponible en todas las implementaciones de .NET. La finalidad de .NET Standard es establecer una mayor uniformidad en el ecosistema de .NET.

.NET Standard habilita los escenarios clave siguientes:

- ✓ Define un conjunto uniforme de API de la Biblioteca de Clases Base (BCL) para todas las implementaciones de .NET que se van a implementar, independientemente de la carga de trabajo.
- ✓ Permite a los desarrolladores generar bibliotecas portátiles que se pueden usar en las implementaciones de .NET con este mismo conjunto de API.
- ✓ Reduce o incluso elimina la compilación condicional de código fuente compartido debido a las API de .NET, solo para API de sistema operativo.

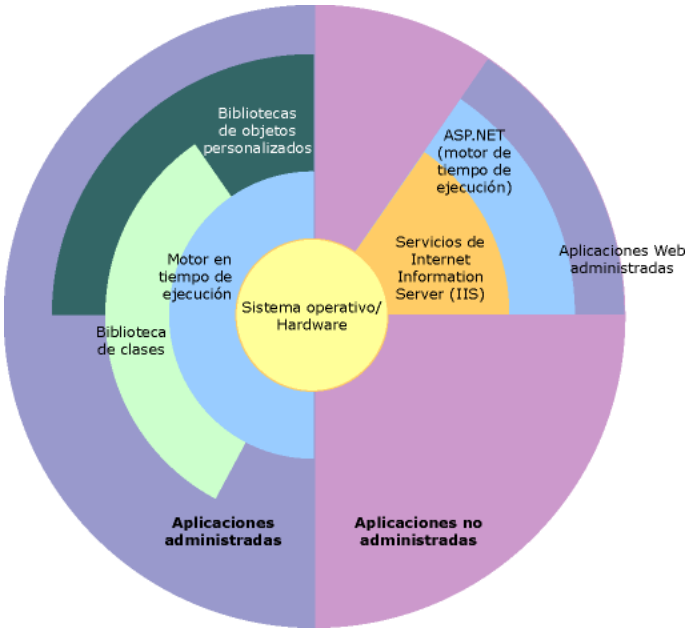
Las diversas implementaciones de .NET tienen como destino versiones concretas de .NET Standard. Cada implementación de .NET anuncia la última versión más alta de .NET Standard que admite, indicación de que también es compatible con versiones anteriores. Por ejemplo, .NET Framework 4.6 implementa .NET Standard 1.3, lo que significa que expone todas las API definidas en las versiones de .NET Standard de 1.0 a 1.3. De forma similar, .NET Framework 4.6.1 implementa .NET Standard 1.4, mientras que .NET Core 1.0 implementa .NET Standard 1.6.

2. Componentes de la plataforma .NET

.NET Framework es un entorno de ejecución administrado para Windows que proporciona diversos servicios a las aplicaciones en ejecución. Consta de dos componentes principales:

1. El Tiempo de Ejecución en Lenguaje Común o Common Language Runtime (CLR), que es el motor de ejecución que controla las aplicaciones en ejecución.
2. La Biblioteca de Clases Base de .NET Framework (BCL), que proporciona una biblioteca de código probado y reutilizable al que pueden llamar los desarrolladores desde sus propias aplicaciones.

En la figura siguiente se muestra la relación del CLR y la BCL con las aplicaciones y el sistema en su conjunto. En la figura se representa igualmente cómo funciona el código administrado dentro de una arquitectura mayor.



2.1. El Tiempo de Ejecución en Lenguaje Común (CLR)

El **Tiempo de Ejecución en Lenguaje Común o CLR** se puede considerar como un agente que administra el código en tiempo de ejecución y proporciona servicios centralizados, como la administración de memoria, la administración de subprocesos y la comunicación remota, al tiempo que aplica una seguridad de tipos estricta y otras formas de especificación del código que promueven su seguridad y solidez.

De hecho, el concepto de **administración de código** es un principio básico del motor en tiempo de ejecución. El código destinado al tiempo de ejecución se denomina código administrado, a diferencia del resto de código, que se conoce como código no administrado.

Para permitir al motor en tiempo de ejecución proporcionar servicios al código administrado, los compiladores de lenguajes deben emitir **metadatos** que describen los tipos, los miembros y las referencias del código. Los metadatos se almacenan con el código. El motor en tiempo de ejecución utiliza los metadatos para localizar y cargar clases, colocar instancias en memoria, resolver invocaciones a métodos, generar código nativo, exigir mecanismos de seguridad y establecer los límites del contexto en tiempo de ejecución.

El tiempo de ejecución controla automáticamente la disposición de los objetos y administra las referencias a éstos, liberándolos cuando ya no se utilizan. Los objetos cuya duración se administra de esta forma se denominan datos administrados. La recolección de elementos no utilizados elimina pérdidas de memoria así como otros errores habituales de programación.

Los compiladores y las herramientas exponen la funcionalidad del CLR y permiten escribir código con las ventajas que proporciona este entorno de ejecución administrado.

El CLR o motor en tiempo de ejecución ofrece las siguientes ventajas:

- ✓ Mejoras en el rendimiento.
- ✓ Capacidad para utilizar fácilmente componentes desarrollados en otros lenguajes.
- ✓ Tipos extensibles que proporciona una biblioteca de clases

- ✓ Características del lenguaje como herencia, interfaces y sobrecarga para la programación orientada a objetos.
- ✓ Compatibilidad con subprocesamiento libre explícito que permite la creación de aplicaciones multiprocesos escalables.
- ✓ Compatibilidad con el control de excepciones estructurado.
- ✓ Compatibilidad con atributos personalizados.
- ✓ Recolección de elementos no utilizados.
- ✓ Emplee delegados en lugar de punteros a funciones para mayor seguridad y protección de tipos.

2.2. El Código administrado

El **código administrado** es simplemente eso: código cuya ejecución está administrada mediante un tiempo de ejecución. En este caso, el tiempo de ejecución en cuestión es el Common Language Runtime o CLR, independientemente de la implementación (Mono, .NET Framework o .NET Core).

El CLR se encarga de tomar el código administrado, compilarlo en código máquina y, después, ejecutarlo. Además de eso, el tiempo de ejecución proporciona varios servicios importantes, como la administración de memoria automática, los límites de seguridad, la seguridad de los tipos, etc.

En un entorno no administrado, el programador se encarga prácticamente de todo. El programa real es, básicamente, un archivo binario que el sistema operativo (SO) carga en la memoria e inicia. Todo lo demás, desde la administración de memoria hasta las consideraciones de seguridad, es responsabilidad del programador.

El código administrado se escribe en uno de los lenguajes de alto nivel que se pueden ejecutar en la plataforma .NET. Cuando se compila código escrito en estos lenguajes con su respectivo compilador, no se obtiene código máquina, se obtiene código de **lenguaje intermedio** o **Intermediate Language (IL)** o **Lenguaje Intermedio Común (CIL)** o **Lenguaje Intermedio de Microsoft (MSIL)**, que más adelante el tiempo de ejecución o CLR compila y ejecuta.

Una vez que haya producido IL a partir del código fuente de alto nivel, lo más probable es que se quiera ejecutar. Aquí es donde el CLR se encarga del proceso e inicia la compilación Justo a Tiempo o Just-In-Time (JIT) del código IL en código máquina que se puede ejecutar en una CPU particular. De esta manera, el CLR sabe exactamente qué hace el código y puede administrarlo con eficacia.

2.3. El Sistema de Tipos Común (CTS)

Una implementación de .NET es independiente del lenguaje. Esto no sólo significa que un programador pueda escribir su código en cualquier lenguaje. Significa también que tiene que poder interactuar con código escrito en otros lenguajes que se pueden usar en una implementación de .NET.

Para hacer esto de forma transparente, debe haber una forma común de describir todos los tipos compatibles. De esto se encarga el **Sistema de Tipos Común o Common Type System (CTS)**. Se ha creado para realizar varias acciones:

- Establecer un marco para la ejecución de varios lenguajes.
- Proporcionar un modelo orientado a objetos para admitir la implementación de varios lenguajes en una implementación de .NET.

- Definir un conjunto de reglas que deben seguir todos los lenguajes al trabajar con tipos.
- Proporcionar una biblioteca que contenga los tipos de datos primitivos que se emplean en el desarrollo de aplicaciones (por ejemplo, Boolean, Byte, Char, etc.).

El CTS define dos tipos principales que deben ser compatibles:

- ✓ **Tipos de referencia**. Los objetos de los tipos de referencia se representan mediante una referencia al valor real del objeto, una referencia es similar a un puntero en C/C++. Simplemente, hace referencia a una ubicación de memoria donde están los valores de los objetos.
- ✓ **Tipos de valor**. Los objetos se representan mediante sus valores. Si asigna un tipo de valor a una variable, está copiando un valor del objeto.

El CTS define varias categorías de tipos, cada una con su semántica y uso específicos:

- Clases
- Estructuras
- Enumeraciones
- Interfaces
- Delegados

Además, el CTS también define todas las demás propiedades de los tipos, como modificadores de acceso, cuáles son los miembros de tipo válidos, cómo funcionan la herencia y la sobrecarga, etc.

2.4. La Especificación en Lenguaje Común (CLS)

Para habilitar escenarios de interoperabilidad completa, todos los objetos que se creen en el código deben basarse en algunas similitudes en los lenguajes que los usan. Puesto que hay numerosos lenguajes diferentes, .NET ha especificado las similitudes en lo que se denomina la **Especificación en Lenguaje Común o Common Language Specification (CLS)**.

El CLS define un conjunto de características comunes para diferentes lenguajes de programación. También proporciona una especie de receta para cualquier lenguaje que se implementa sobre .NET sobre qué necesita para ser compatible.

El CLS es un subconjunto del CTS. Esto significa que todas las reglas del CTS se aplican también al CLS, a menos que las reglas del CLS sean más estrictas.

2.5. La Biblioteca de Clases de .NET (BCL)

La plataforma .NET tiene un amplio **conjunto de bibliotecas estándar**, a las que se hace referencia como bibliotecas de clases base (conjunto básico) o bibliotecas de clases de framework o marco de trabajo (conjunto completo).

Hay tres tipos de bibliotecas de clases que puede usar:

1. Las **bibliotecas de clases específicas de la plataforma** tienen acceso a todas las API de una plataforma determinada (por ejemplo, .NET Framework, Xamarin iOS), pero solo las pueden usar las aplicaciones y bibliotecas destinadas a esa plataforma.
2. Las **bibliotecas de clases portables** tienen acceso a un subconjunto de API y las pueden usar las aplicaciones y bibliotecas que tienen como destino varias plataformas.

3. Las **bibliotecas de clases de .NET Standard** son una fusión del concepto de biblioteca específica de la plataforma y portable en un único modelo que ofrece lo mejor de ambas.

Los **tipos de .NET** usan un esquema de nomenclatura con sintaxis de punto lo que indica la existencia de una jerarquía. Esta técnica agrupa tipos relacionados en **espacios de nombres** para que se pueda buscar y hacer referencia a ellos más fácilmente. La primera parte del nombre completo, hasta el punto situado más a la derecha, es el nombre del espacio de nombres. La última parte es el nombre de tipo. Por ejemplo, System.Collections.Generic.List<T> representa el tipo List<T>, que pertenece al espacio de nombres System.Collections.Generic.

Este esquema de nomenclatura facilita a los programadores de bibliotecas la tarea de extender .NET Framework para poder crear grupos jerárquicos de tipos y asignarles nombre de forma coherente e ilustrativa. También permite identificar de forma inequívoca los tipos mediante su nombre completo, es decir, por su espacio de nombres y nombre de tipo. Esto evita que se produzcan conflictos entre los nombres de tipo.

El espacio de nombres **System** es el espacio de nombres de la raíz de los tipos fundamentales de .NET. Este espacio de nombres contiene clases que representan los tipos de datos base que se utilizan en todas las aplicaciones.

.NET incluye un conjunto de tipos primitivos, que se usan en todos los programas. Estos tipos contienen datos, como números, cadenas, bytes y objetos:

- System.Object (object): la clase base fundamental en el sistema de tipos de CLR. Es la raíz de la jerarquía de tipos.
- System.Int16 (short): tipo entero con signo de 16 bits. También existe el valor UInt16 sin signo.
- System.Int32 (int): tipo entero con signo de 32 bits. También existe el valor UInt32 sin firmar.
- System.Single (float): tipo de punto flotante de 32 bits.
- System.Decimal (decimal): tipo decimal de 128 bits.
- System.Byte (byte): entero sin signo de 8 bits que representa un byte de memoria.
- System.Boolean (bool): tipo booleano que representa true o false.
- System.Char (char): tipo numérico de 16 bits que representa un carácter Unicode.
- System.String (string): representa una serie de caracteres.

.NET incluye un conjunto de estructuras de datos que son la piedra angular de casi cualquier aplicación .NET:

- Array: representa una matriz de objetos fuertemente tipados a la que se puede obtener acceso por índice. Tiene un tamaño fijo debido a su construcción.
- List<T>: representa una lista de objetos fuertemente tipados a la que se puede obtener acceso por índice. Cambia automáticamente de tamaño según sea necesario.
- Dictionary<TKey,TValue>: representa una colección de valores que se indexan mediante una clave. Se puede obtener acceso a los valores a través de la clave. Cambia automáticamente de tamaño según sea necesario.

- Uri: proporciona una representación de objeto de un identificador uniforme de recursos (URI) y un acceso sencillo a las partes del identificador URI.
- DateTime: representa un instante de tiempo, normalmente expresado en forma de fecha y hora del día.

.NET incluye un conjunto de API de utilidades que proporcionan funcionalidad para muchas tareas importantes:

- HttpClient: una API para enviar solicitudes HTTP y recibir respuestas HTTP de un recurso identificado por un URI.
- XDocument: una API para cargar y consultar documentos XML con LINQ.
- StreamReader: una API para leer archivos. StringWriter: se puede usar para escribir archivos.

Además de los tipos de datos base, el espacio de nombres System contiene más de 100 clases, que comprenden desde las clases que controlan excepciones hasta las clases que tratan conceptos básicos en tiempo de ejecución, como los dominios de aplicación y el recolector de elementos no utilizados.