

# Programación Web .NET Core

Módulo 2

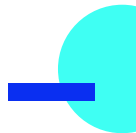
# Conversión de tipos

## Conversión de datos / Upcast / Downcast / Cast

Como a C# se le asignan tipos estáticos en tiempo de compilación, después de declarar una variable, no es posible volver a declararla ni servirá para almacenar valores de otro tipo, a menos que ese tipo pueda convertirse en el tipo de la variable. Por ejemplo, no existe conversión de un entero a una cadena arbitraria cualquiera.

Por lo tanto, después de declarar `i` como entero, no puede asignarle la cadena "Hello", como se muestra en el código:

```
inti; i = "Hello"; // Error: "Cannotimplicitlyconverttype  
'string' to 'int'"
```



Sin embargo, **en ocasiones puede que sea necesario copiar un valor en un parámetro de método o variable de otro tipo**. Por ejemplo, una variable de tipo entero que deba pasar a un método cuyo parámetro es de tipo `double`. O bien, quizás se necesite asignar una variable de clase a una variable de un tipo de interfaz. Estos tipos de operaciones se denominan **conversiones**.

En C#, se pueden realizar los tipos de conversiones que veremos a continuación:

- **Conversiones implícitas:** no se requiere una sintaxis especial porque la conversión se hace con seguridad de tipos y no se perderán datos. Por ejemplo, las conversiones de tipos enteros de menor a mayor y las conversiones de clases derivadas en clases base.
- **Conversiones explícitas (conversiones de tipos):** Requieren un operador de conversión. La conversión se requiere cuando es posible la pérdida de información en el proceso o cuando esta puede no realizarse correctamente por otras razones. Ejemplos habituales; se incluye la conversión en un tipo con menor precisión o un intervalo menor, y la conversión de una instancia de clase base en una clase derivada.

- **Conversiones definidas por el usuario:** Se realizan a través de métodos especiales que puede definir para habilitar las conversiones explícitas e implícitas entre tipos personalizados que no tienen una relación de clase base-clase derivada.
- **Conversiones con clases auxiliares:** para realizar conversiones entre tipos no compatibles, como los enteros u objetos `System.DateTime`, o bien cadenas hexadecimales y matrices de bytes, puede utilizar las clases llamadas `System.BitConverter` o `System.Convert` y los métodos `Parse` de los tipos numéricos integrados, como `Int32.Parse`



## Conversiones implícitas

En los tipos numéricos integrados, puede hacerse una conversión implícita cuando el valor que se va a almacenar puede ajustarse a la variable sin necesidad de truncamiento o redondeo. Por ejemplo, una variable de tipo long (entero de 8 bytes) puede almacenar cualquier valor que pueda almacenar a su vez un elemento int (4 bytes en un equipo de 32 bits).

En el ejemplo de arriba a la derecha, el compilador convierte de manera implícita el valor de la derecha en un tipo **long** antes de asignarlo a **bigNum**.



```
// Implicitconversion. numlong can  
// holdanyvalueanint can hold, and more!  
        intnum = 2147483647;  
longbigNum = num;
```

En los tipos de referencia, siempre existe una conversión implícita desde una clase a cualquiera de sus interfaces o clases base directas o indirectas. No se requiere una sintaxis especial, ya que una clase derivada siempre contiene todos los miembros de una clase base:

```
Derived d = new Derived();  
Base b = d; // Always OK.
```

## Conversiones explícitas

Sin embargo, si no se puede realizar una conversión sin riesgo de perder información, el compilador requiere que se haga una conversión explícita denominada **conversión de tipo**. Una conversión de tipo es una manera de informar al compilador de forma explícita que se pretende realizar la conversión y que se está al tanto de que puede producirse una pérdida de datos. Para hacer una conversión de tipo, se especifica entre paréntesis el tipo al que se va a aplicar dicha conversión delante del valor o la variable que se va a convertir. El programa siguiente convierte un tipo **double** a un tipo **int**. El programa no se compilará sin el operador de conversión de tipo.

```
class Test
{
    static void Main()
    {
        double x = 1234.7;
        int a;
        // Cast double to int.
        a = (int)x;
        System.Console.WriteLine(a);
    }
}
// Output: 1234
```



En los tipos de referencia, se requiere una conversión explícita si se debe convertir de un tipo base a un tipo derivado:

```
// Creación de un nuevo tipo derivado
Jirafa g = newJirafa();

// La conversión implícita al tipo base es segura
// Pasa implícitamente de tipo Jirafa a tipo Animal.
// La clase derivada Jirafa tiene herencia de la clase base Animal.
Animal a = g;

// Es necesaria una conversión explícita para volver al tipo de dato original
// es decir, pasar del tipo Animal al tipo Jirafa
// Nota: Esto compilará pero dará una excepción en tiempo de ejecución
// si el objeto de la derecha, la variable a internamente no es un objeto Jirafa
Jirafa g2 = (Jirafa) a;
```

Una operación de conversión entre tipos de referencia no cambia el tipo en tiempo de ejecución del objeto subyacente; solo cambia el tipo del valor que se utiliza como referencia para ese objeto.



## Excepciones de las conversiones de tipos en tiempo de ejecución

En algunas conversiones de tipos de referencia, el compilador no puede determinar si será válida una conversión de tipo. Es posible que una operación de conversión de tipo que se compila de forma correcta provoque un error en tiempo de ejecución.

Como veremos en el ejemplo de la siguiente diapositiva, una conversión de tipo que origine un error en tiempo de ejecución hará que se produzca una excepción **InvalidCastException**.

Veamos el ejemplo en la próxima pantalla.



```
using System;

class Animal
{
    public void Comer() { Console.WriteLine("Comiendo."); }
    public override string ToString()
    {
        return "Yo soy un animal.";
    }
}

class Reptil : Animal { }
class Mamifero : Animal { }
```

...

```
classUnsafeCast
{
    staticvoidMain()
    {
        Test(newMamifero());

        System.Console.WriteLine("Pulse una tecla para salir.");
        System.Console.ReadKey();
    }

    staticvoidTest(Animal a)
    {
        // Ocurre la excepción InvalidCastException en tiempo de ejecución
        // porque el objeto Mamifero no es convertible a tipo Reptil
        Reptil r = (Reptil)a;
    }
}
```

## Conversiones boxing y unboxing

La conversión **boxing** es el proceso de convertir un tipo de valor en el tipo `object` o en cualquier tipo de interfaz implementada por ese tipo de valor.

Cuando CLR aplica la conversión **boxing** a un tipo de valor, ajusta el valor dentro de una clase `System.Object` y lo almacena en el montón administrado.

La conversión **unboxing** extrae el tipo de valor del objeto. **La conversión boxing es implícita y la conversión unboxing es explícita.**

El concepto de conversión **boxing** y **unboxing** es la base de la vista unificada del sistema de tipos de C#, donde el que **un valor de cualquier tipo se puede tratar como objeto.**

En el ejemplo que vemos a continuación, se aplica la conversión **boxing** a la variable de entero **i** y esta se asigna al objeto **o**:

```
int i = 123;  
// The following line boxes i.  
object o = i;
```

Se puede aplicar la conversión **unboxing** al objeto **o** y asignarlo a la variable de entero **i**:

```
o = 123;  
i = (int)o; // unboxing
```

En los slides siguientes se muestra cómo usar la conversión **boxing** en C#:



```
// String.Concatexample.  
// String.Concat has manyversions. Restthe mouse pointer on  
// Concat in thefollowingstatementtoverifythattheversion//  
thatisusedheretakesthreeobjectarguments. Both 42 and// true must be boxed.  
Console.WriteLine(String.Concat("Answer", 42, true));
```

```
// Listexample.  
// Create a listofobjectstohold a heterogeneouscollection// ofelements.  
List<object>mixedList = newList<object>();
```

```
// Add a stringelementtothelist.  
mixedList.Add("FirstGroup:");
```

```
// AddsomeintegerstotheList.  
for (int j = 1; j < 5; j++)
```

```
    {  
    // Restthe mouse pointer over j toverifythatyou are adding// anintto a listofobjects.  
    Eachement j isboxedwhen// youadd j tomixedList.  
    mixedList.Add(j);  
    }  
  
    // Addanotherstring and more integers.  
    mixedList.Add("SecondGroup:");  
    for (int j = 5; j < 10; j++)  
    {  
    mixedList.Add(j);  
    }  
  
    // Displaytheelements in thelist. Declare theloop variable by// usingvar, so  
    thatthecompilerassignsitstype.  
    foreach (variteminmixedList)  
    {  
    // Restthe mouse pointer overitemtoverifythattheelements// ofmixedList are objects.  
    Console.WriteLine(item);  
    }
```

...

```
// The following loop sums the squares of the first group of boxed
// integers in mixedList. The list elements are objects, and cannot
// be multiplied or added to the sum until they are unboxed. The
// unboxing must be done explicitly.
var sum = 0;
for (var j = 1; j < 5; j++)
{
    // The following statement causes a compiler error: Operator '*' cannot be
    // applied to operand of type 'object' and 'object'.
    // sum += mixedList[j] * mixedList[j];

    // After the list elements are unboxed, the computation does not cause a compiler error.
    sum += (int)mixedList[j] * (int)mixedList[j];
}
```

...



...

```
// The sum displayed is 30, the sum of 1 + 4 + 9 + 16. Console.WriteLine("Sum: " + sum);

// Output:
// Answer42True// FirstGroup:
// 1
// 2
// 3
// 4
// SecondGroup:
// 5
// 6
// 7
// 8
// 9
// Sum: 30
```

## Rendimiento

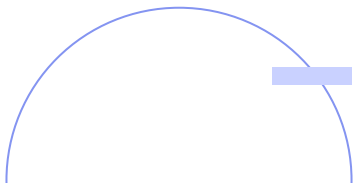
Con relación a las asignaciones simples, las conversiones **boxing** y **unboxing** son procesos que consumen muchos recursos. **Cuando se aplica la conversión boxing a un tipo de valor, se debe asignar y construir un objeto completamente nuevo.** En menor grado, la conversión de tipos requerida para aplicar la conversión unboxing también es costosa.

## Conversión boxing

Se utiliza para almacenar tipos de valor en el montón de recolección de elementos no usados. **Es una conversión implícita de un tipo de valor al tipo object o a cualquier tipo de interfaz implementada por este tipo de valor.** Al aplicar la conversión **boxing** a un tipo de valor se asigna una instancia de objeto en el montón y se copia el valor en el nuevo objeto.

Considere la siguiente declaración de una variable de tipo de valor:

```
int i = 123;
```



La siguiente instrucción aplica **implícitamente** la operación de conversión **boxing** en la variable **i**:

```
// Boxing copies the value of i into object o.  
object o = i;
```

El resultado de esta instrucción es crear una referencia de objeto **o** en la pila que hace referencia a un valor del tipo **int** en el montón. Este valor es una copia del tipo de valor asignado a la variable **i**. La diferencia entre las dos variables, **i** y **o**, se muestra en la figura de la derecha:

En la pila

En el montón

**i**



int i =123;

**o**



object o=i;

(i boxed)



También es posible realizar la conversión **boxing de manera explícita**, tal como se muestra en el ejemplo siguiente, pero esta nunca es necesaria:

```
int i = 123;  
object o = (object)i; // explicitboxing
```

Este ejemplo convierte una variable de entero **i** en un objeto **o** mediante la conversión **boxing**. A continuación, el valor almacenado en la variable **i** se cambia de **123** a **456**. El ejemplo muestra que el tipo de valor original y el objeto al que se ha aplicado la conversión **boxing** usan ubicaciones de memoria independientes y, por consiguiente, pueden almacenar valores diferentes.

Pasemos a la pantalla siguiente para ver el ejemplo mencionado.



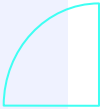
```
class TestBoxing
{
    static void Main()
    {
        int i = 123;

        // Boxing copies the value of i into object o.
        object o = i;

        // Change the value of i.
        i = 456;

        // The change in i does not effect the value stored in o.
        System.Console.WriteLine("The value-type value = {0}", i);
        System.Console.WriteLine("The object-type value = {0}", o);
    }
}

/* Output:
The value-type value = 456
The object-type value = 123
*/
```



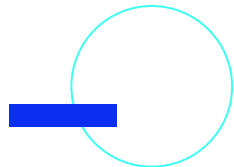
## Conversión Unboxing

La conversión **unboxing** es una **conversión explícita del tipo object a un tipo de valor o de un tipo de interfaz a un tipo de valor que implementa la interfaz**. Una operación de conversión unboxing consiste en lo siguiente:

- Comprobar la instancia de objeto para asegurarse de que se trata de un valor de conversión boxing del tipo de valor dado.
- Copiar el valor de la instancia en la variable de tipo de valor.

Las siguientes instrucciones muestran las operaciones de conversión **boxing** y **unboxing**:

```
int i = 123; // a valuetype  
object o = i; // boxing  
int j = (int)o; // unboxing
```



En la figura siguiente se muestra el resultado de las instrucciones anteriores:

En la pila

i



`int i =123;`

o



`object o=i;`

i



`int j =(int) o;`

En el montón

(i boxed)



Para que la conversión **unboxing** de tipos de valor sea correcta en tiempo de ejecución, el elemento al que se aplica debe ser una referencia a un objeto creado previamente mediante la conversión **boxing** de una instancia de ese tipo de valor. Si se intenta aplicar la conversión **unboxing** a **null**, se producirá una excepción llamada **NullReferenceException**. Si se intenta aplicar la conversión **unboxing** a una referencia de un tipo de valor incompatible se producirá una excepción **InvalidCastException**. **3.0.0.1**.

### Ejemplo:

El ejemplo siguiente muestra un caso de conversión **unboxing** no válida y la excepción **InvalidCastException** resultante. Si se utiliza **try** y **catch**, se muestra un mensaje de error cuando se produce el error.

[illegible]



Este programa produce el resultado siguiente:

```
Specifiedcastisnotvalid. Error: Incorrectunboxing.
```

Si cambia la instrucción:

```
int j = (short) o;
```

a:

```
int j = (int) o;
```

la conversión se realizará y se obtendrá el resultado:

```
Unboxing OK.
```

Fuente: [Docs.microsoft.com](https://docs.microsoft.com)



**¡Sigamos  
trabajando!**