**Cast e conversioni di tipi (Guida per programmatori C#)**

http://msdn.microsoft.com/it-it/library/ms173105.aspx

Poiché il codice C# viene tipizzato in modo statico in fase di compilazione, una variabile dichiarata non può essere nuovamente dichiarata o utilizzata per archiviare valori di un altro tipo a meno che il tipo non sia convertibile nel tipo della variabile. Ad esempio, non esiste alcuna conversione da un intero a una stringa arbitraria. Pertanto, dopo avere dichiarato i come intero, non è possibile assegnare a esso la stringa "Hello", come mostrato nel codice seguente.

int i;

i = "Hello"; // Error: "Cannot implicitly convert type 'string' to 'int'"

Tuttavia, copiare un valore in una variabile o in un parametro di metodo di un altro tipo potrebbe essere talvolta necessario. Ad esempio, è possibile che si disponga di una variabile integer che è necessario passare a un metodo il cui parametro è tipizzato come double. Questi tipi di operazioni sono detti conversioni di tipi. In C#, è possibile eseguire i tipi di conversioni seguenti:

* **Conversioni implicite**: non è richiesta alcuna sintassi speciale perché la conversione è indipendente dai tipi e nessun dato viene perso. Gli esempi includono le conversioni da tipi integrali più piccoli a più grandi
* **Conversioni esplicite (cast)**: le conversioni esplicite richiedono un operatore di cast. Il cast è obbligatoria quando le informazioni potrebbero andare perdute durante la conversione o quando la conversione potrebbe non riuscire per altri motivi. Gli esempi tipici includono la conversione numerica in un tipo con meno precisione o un intervallo più piccolo
* **Conversioni con le classi di supporto**: per eseguire la conversione tra tipi non compatibili, ad esempio numeri interi e oggetti System.DateTime o stringhe esadecimali e matrici di byte, è possibile utilizzare la classe System.Convert e i metodi Parse dei tipi numerici incorporati, ad esempio Int32.Parse.

**Conversioni implicite**

Per i tipi numerici incorporati, una conversione implicita può essere eseguita quando il valore da archiviare può essere adattato alla variabile senza essere troncato o arrotondato. Ad esempio, una variabile di tipo long (intero a 8 byte) è in grado di archiviare qualsiasi valore archiviabile in un tipo int (4 byte su un computer a 32 bit). Nell'esempio seguente, il compilatore converte implicitamente il valore a destra in un tipo long prima di assegnarlo a bigNum.

// Implicit conversion. num long can

// hold any value an int can hold, and more!

int num = 2147483647;

long bigNum = num;

Per un elenco completo delle conversioni numeriche implicite, vedere [Tabella delle conversioni numeriche implicite (Riferimenti per C#)](http://msdn.microsoft.com/it-it/library/y5b434w4.aspx).

**Conversioni esplicite**

Tuttavia, se una conversione non può essere eseguita senza un rischio di perdita di informazioni, il compilatore richiede che si esegua una conversione esplicita, chiamata cast. Il cast è un modo di informare in modo esplicito il compilatore che si intende eseguire la conversione e che si è consapevoli che potrebbe verificarsi una perdita di dati. Per eseguire un cast, specificare il tipo di destinazione del cast in parentesi davanti al valore o alla variabile da convertire. Il seguente programma esegue il cast di double su int. Se il cast non è presente, il programma non verrà compilato.

class Test

{

static void Main()

{

double x = 1234.7;

int a;

// Cast double to int.

a = (int)x;

System.Console.WriteLine(a);

}

}

// Output: 1234

Per un elenco delle conversioni numeriche esplicite consentite, vedere [Tabella delle conversioni numeriche esplicite (Riferimenti per C#)](http://msdn.microsoft.com/it-it/library/yht2cx7b.aspx).

# Metodo Int32.Parse

http://msdn.microsoft.com/it-it/library/f56zcx39(v=vs.110).aspx

# Metodo Int32.Parse (String, IFormatProvider)

Converte la rappresentazione in forma di stringa di un numero in un formato specifico delle impostazioni cultura nell'equivalente intero con segno a 32 bit.

public static int Parse(string s, IFormatProvider provider)

#### Parametri

S Stringa contenente un numero da convertire.

Provider Oggetto che fornisce informazioni di formattazione specifiche delle impostazioni cultura relativamente a s.

#### Valore restituito

Intero con segno a 32 bit equivalente al numero specificato in s.

**Eccezioni**

ArgumentNullException s è null.

FormatException s non è nel formato corretto.

OverflowException s rappresenta un numero minore di MinValue o maggiore di MaxValue.

**Note**

Questo overload del metodo Parse(String, IFormatProvider) viene utilizzato in genere per convertire un testo che può essere formattato in diversi modi in un valore Int32. Ad esempio, può essere utilizzato per convertire in un valore numerico il testo immesso da un utente in una casella di testo HTML.

Il parametro s contiene un numero nella forma: [sv][segno]cifre[sv]

Gli elementi tra parentesi quadre ([ e ]) sono facoltativi. Nella tabella riportata di seguito viene descritto ogni elemento.

svSpazio vuoto facoltativo.

signSegno facoltativo.

cifreSequenza di cifre comprese tra 0 e 9.

Il parametro s viene interpretato utilizzando lo stile NumberStyles.Integer. Oltre alle cifre decimali, sono consentiti solo gli spazi iniziali e finali con un segno iniziale. Per definire in modo esplicito gli elementi di stile che possono essere presenti in s, utilizzare il metodo Int32.Parse(String, NumberStyles, IFormatProvider).

Il parametro provider è un'implementazione di IFormatProvider, ad esempio un oggetto NumberFormatInfo o CultureInfo. Il parametro provider fornisce informazioni specifiche delle impostazioni cultura relative al formato di s. Se provider è null, viene utilizzato l'oggetto NumberFormatInfo per le impostazioni cultura correnti.

**Esempio**

string stringa = "123";

int Numero = Int32.Parse(stringa);

stringa = "1A2B";

Numero = Int32.Parse(stringa); //Errore!!!

# Metodo Char.Parse

http://msdn.microsoft.com/it-it/library/system.char.parse(v=vs.110).aspx

Converte il valore della stringa specificata nell'equivalente carattere Unicode.

public static char Parse(string s)

**Parametri**

S Stringa contenente un solo carattere oppure null.

**Valore restituito**

System.Char: Un carattere Unicode equivalente all'unico carattere in s altrimenti un’eccezione.

**Eccezioni:**

**Condizione Eccezione**

s è null. ArgumentNullException

La lunghezza di s non è 1. FormatException

**Esempi**

public class ParseSample {

public static void Main() {

Console.WriteLine(Char.Parse("A")); // Output: 'A'

}

}

**Metodo Char.TryParse**

http://msdn.microsoft.com/it-it/library/system.char.tryparse(v=vs.110).aspx

Converte il valore della stringa specificata nell'equivalente carattere Unicode.Un codice restituito indica se la conversione ha avuto esito positivo o negativo. Il metodo TryParse è come il metodo [Parse](http://msdn.microsoft.com/it-it/library/system.char.parse(v=vs.110).aspx), ma il metodo TryParse non genera un'eccezione se la conversione non riesce.

public static bool TryParse(string s, out char result)

**Parametri**

s Stringa contenente un solo carattere oppure null.

result Se la conversione ha esito positivo, il risultato restituito dal metodo contiene un carattere Unicode equivalente all'unico carattere contenuto nel parametro s; in caso contrario, il risultato contiene un valore non definito.La conversione non viene eseguita se il parametro s è null o se la lunghezza di s non è 1.Questo parametro viene passato non inizializzato.

**Valore restituito** System.Boolean true se il parametro s è stato convertito correttamente; in caso contrario, false.

**Esempi**

Nell'esempio di codice riportato di seguito sono illustrati gli overload del metodo TryParse per diversi tipi di base e il metodo per il metodo TryParseExact per il tipo di base DateTime.

using System;

using System.Globalization;

class Sample

{

public static void Main()

{

bool result;

CultureInfo ci;

string nl = Environment.NewLine;

string msg1 =

"This example demonstrates overloads of the TryParse method for{0}" +

"several base types, as well as the TryParseExact method for DateTime.{0}";

string msg2 = "Non-numeric types:{0}";

string msg3 = "{0}Numeric types:{0}";

// Non-numeric types.

Boolean booleanVal;

Char charVal;

DateTime datetimeVal;

// Numeric types.

Byte byteVal;

Int16 int16Val;

Int32 int32Val;

Int64 int64Val;

Decimal decimalVal;

Single singleVal;

Double doubleVal;

//

Console.WriteLine(msg1, nl);

// Non-numeric types:

Console.WriteLine(msg2, nl);

// DateTime

// TryParse:

// Assume current culture is en-US, and dates of the form: MMDDYYYY.

result = DateTime.TryParse("7/4/2004 12:34:56", out datetimeVal);

Show(result, "DateTime #1", datetimeVal.ToString());

// Use fr-FR culture, and dates of the form: DDMMYYYY.

ci = new CultureInfo("fr-FR");

result = DateTime.TryParse("4/7/2004 12:34:56",

ci, DateTimeStyles.None, out datetimeVal);

Show(result, "DateTime #2", datetimeVal.ToString());

// TryParseExact:

// Use fr-FR culture. The format, "G", is short date and long time.

result = DateTime.TryParseExact("04/07/2004 12:34:56", "G",

ci, DateTimeStyles.None, out datetimeVal);

Show(result, "DateTime #3", datetimeVal.ToString());

// Assume en-US culture.

string[] dateFormats = {"f", "F", "g", "G"};

result = DateTime.TryParseExact("7/4/2004 12:34:56 PM",

dateFormats, null, DateTimeStyles.None,

out datetimeVal);

Show(result, "DateTime #4", datetimeVal.ToString());

Console.WriteLine();

// Boolean

result = Boolean.TryParse("true", out booleanVal);

Show(result, "Boolean", booleanVal.ToString());

// Char

result = Char.TryParse("A", out charVal);

Show(result, "Char", charVal.ToString());

// Numeric types:

Console.WriteLine(msg3, nl);

// Byte

result = Byte.TryParse("1", NumberStyles.Integer, null, out byteVal);

Show(result, "Byte", byteVal.ToString());

// Int16

result = Int16.TryParse("-2", NumberStyles.Integer, null, out int16Val);

Show(result, "Int16", int16Val.ToString());

// Int32

result = Int32.TryParse("3", NumberStyles.Integer, null, out int32Val);

Show(result, "Int32", int32Val.ToString());

// Int64

result = Int64.TryParse("4", NumberStyles.Integer, null, out int64Val);

Show(result, "Int64", int64Val.ToString());

// Decimal

result = Decimal.TryParse("-5.5", NumberStyles.Number, null, out decimalVal);

Show(result, "Decimal", decimalVal.ToString());

// Single

result = Single.TryParse("6.6",

(NumberStyles.Float | NumberStyles.AllowThousands),

null, out singleVal);

Show(result, "Single", singleVal.ToString());

// Double

result = Double.TryParse("-7",

(NumberStyles.Float | NumberStyles.AllowThousands),

null, out doubleVal);

Show(result, "Double", doubleVal.ToString());

// Use the simple Double.TryParse overload, but specify an invalid value.

result = Double.TryParse("abc", out doubleVal);

Show(result, "Double #2", doubleVal.ToString());

//

}

protected static void Show(bool parseResult, string typeName,

string parseValue)

{

string msgSuccess = "Parse for {0} = {1}";

string msgFailure = "\*\* Parse for {0} failed. Invalid input.";

//

if (parseResult == true)

Console.WriteLine(msgSuccess, typeName, parseValue);

else

Console.WriteLine(msgFailure, typeName);

}

}

/\*

This example produces the following results:

This example demonstrates overloads of the TryParse method for

several base types, as well as the TryParseExact method for DateTime.

Non-numeric types:

Parse for DateTime #1 = 7/4/2004 12:34:56 PM

Parse for DateTime #2 = 7/4/2004 12:34:56 PM

Parse for DateTime #3 = 7/4/2004 12:34:56 PM

Parse for DateTime #4 = 7/4/2004 12:34:56 PM

Parse for Boolean = True

Parse for Char = A

Numeric types:

Parse for Byte = 1

Parse for Int16 = -2

Parse for Int32 = 3

Parse for Int64 = 4

Parse for Decimal = -5.5

Parse for Single = 6.6

Parse for Double = -7

\*\* Parse for Double #2 failed. Invalid input.

**Classe Convert**

http://msdn.microsoft.com/it-it/library/system.convert.aspx

Converte un tipo di dati di base in un altro tipo di dati di base.

**Metodi (Alcuni)**

|  |  |
| --- | --- |
| ToBoolean(String) | Converte una stringa che rappresenta un valore logico nell'equivalente valore Boolean. |
| ToChar(Int32) | Converte l'intero con segno a 32 bit specificato nell'equivalente carattere Unicode. |
| ToDateTime(String) | Converte la rappresentazione di stringa specificata di una data e un'ora in un valore di data e ora equivalente. |
| ToDouble(String) | Converte la rappresentazione di stringa specificata di un numero in un numero a virgola mobile e con precisione doppia equivalente. |
| ToInt32(String) | Converte la rappresentazione di stringa specificata di un numero in un intero con segno a 32 bit equivalente. |
| ToInt64(Int32) | Converte il valore dell'intero con segno a 32 bit specificato in un equivalente intero con segno a 64 bit.. |
| ToInt64(Int64) | Restituisce l'intero con segno a 64 bit specificato; non viene eseguita alcuna conversione effettiva. |
| ToString(DateTime) | Converte il valore dell'oggetto DateTime specificato nella relativa rappresentazione di stringa equivalente. |

**Note**

I metodi statici della classe Convert vengono utilizzati per supportare la conversione da e verso tipi di dati di base in .NET Framework. I tipi di base supportati sono Boolean, Char, SByte, Byte, Int16, Int32, Int64, UInt16, UInt32, UInt64, Single, Double, Decimal, DateTime e String.

**Conversioni da e verso tipi di base**

Esiste un metodo di conversione per convertire ogni tipo di base in ogni altro tipo di base. Tuttavia, la chiamata effettiva a un particolare metodo di conversione può produrre cinque risultati diversi, a seconda del valore del tipo di base in fase di esecuzione e del tipo di base di destinazione. Tali risultati sono:

Nessuna conversione. Si verifica quando si tenta di effettuare una conversione da un tipo nel tipo stesso, ad esempio chiamando Convert.ToInt32(Int32) con un argomento di tipo Int32. In tal caso, il metodo restituisce semplicemente un'istanza del tipo originale.

Oggetto InvalidCastException. Si verifica quando una particolare conversione non è supportata. InvalidCastException viene generato per le seguenti conversioni:

Conversioni da Char a Boolean, Single, Double, Decimal o DateTime.

Conversioni da Boolean, Single, Double, Decimal o DateTime a Char.

Conversioni da DateTime a qualsiasi altro tipo eccetto String.

Conversioni da qualsiasi altro tipo, eccetto String a DateTime.

Oggetto FormatException. Si verifica quando il tentativo di convertire un valore della stringa in un altro tipo di base non riesce perché la stringa non è nel formato appropriato. L'eccezione viene generata per le conversioni elencate di seguito.

Oggetto OverflowException. Si verifica quando una conversione verso un tipo di dati più piccolo comporta una perdita di dati.

**Esempio**

Nell'esempio riportato di seguito vengono illustrati alcuni metodi di conversione della classe Convert, tra cui ToInt32, ToBoolean e ToString.

double dNumber = 23.15;

try {

// Returns 23

int iNumber = System.Convert.ToInt32(dNumber);

}

catch (System.OverflowException) {

System.Console.WriteLine(

"Overflow in double to int conversion.");

}

// Returns True

bool bNumber = System.Convert.ToBoolean(dNumber);

// Returns "23.15"

string strNumber = System.Convert.ToString(dNumber);

try {

// Returns '2'

char chrNumber = System.Convert.ToChar(strNumber[0]);

}

catch (System.ArgumentNullException) {

System.Console.WriteLine("String is null");

}

catch (System.FormatException) {

System.Console.WriteLine("String length is greater than 1.");

}

// System.Console.ReadLine() returns a string and it

// must be converted.

int newInteger = 0;

try {

System.Console.WriteLine("Enter an integer:");

newInteger = System.Convert.ToInt32(

System.Console.ReadLine());

}

catch (System.ArgumentNullException) {

System.Console.WriteLine("String is null.");

}

catch (System.FormatException) {

System.Console.WriteLine("String does not consist of an " +

"optional sign followed by a series of digits.");

}

catch (System.OverflowException) {

System.Console.WriteLine(

"Overflow in string to int conversion.");

}

System.Console.WriteLine("Your integer as a double is {0}",

System.Convert.ToDouble(newInteger));