HASHTABLES

Representa una colección de pares de clave y valor que se organizan por código hash de la clave.

Ejemplos

El ejemplo siguiente muestra cómo crear, inicializar y realizar varias funciones para un Hashtable y cómo imprimir sus claves y valores.

```
using System;
using System.Collections;

class Example
{
    public static void Main()
    {
        // Create a new hash table.
        //
        Hashtable openWith = new Hashtable();

        // Add some elements to the hash table. There are no
        // duplicate keys, but some of the values are duplicates.
        openWith.Add("txt", "notepad.exe");
        openWith.Add("bmp", "paint.exe");
        openWith.Add("dib", "paint.exe");
        openWith.Add("rtf", "wordpad.exe");

        // The Add method throws an exception if the new key is
        // already in the hash table.
        try
        {
            openWith.Add("txt", "winword.exe");
        }
        catch
        {
            Console.WriteLine("An element with Key = \"txt\" already
exists.");
        }

        // The Item property is the default property, so you
        // can omit its name when accessing elements.
        Console.WriteLine("For key = \"rtf\", value = {0}.",
openWith["rtf"]);

        // The default Item property can be used to change the value
```

```
        // associated with a key.
        openWith["rtf"] = "winword.exe";
        Console.WriteLine("For key = \"rtf\", value = {0}.",
openWith["rtf"]);

        // If a key does not exist, setting the default Item property
        // for that key adds a new key/value pair.
        openWith["doc"] = "winword.exe";

        // ContainsKey can be used to test keys before inserting
        // them.
        if (!openWith.ContainsKey("ht"))
        {
            openWith.Add("ht", "hypertrm.exe");
            Console.WriteLine("Value added for key = \"ht\": {0}",
openWith["ht"]);
        }

        // When you use foreach to enumerate hash table elements,
        // the elements are retrieved as KeyValuePair objects.
        Console.WriteLine();
        foreach( DictionaryEntry de in openWith )
        {
            Console.WriteLine("Key = {0}, Value = {1}", de.Key, de.Value);
        }

        // To get the values alone, use the Values property.
        ICollection valueColl = openWith.Values;

        // The elements of the ValueCollection are strongly typed
        // with the type that was specified for hash table values.
        Console.WriteLine();
        foreach( string s in valueColl )
        {
            Console.WriteLine("Value = {0}", s);
        }

        // To get the keys alone, use the Keys property.
        ICollection keyColl = openWith.Keys;

        // The elements of the KeyCollection are strongly typed
        // with the type that was specified for hash table keys.
        Console.WriteLine();
        foreach( string s in keyColl )
        {
            Console.WriteLine("Key = {0}", s);
        }

        // Use the Remove method to remove a key/value pair.
```

```
        Console.WriteLine("\nRemove(\"doc\")");
        openWith.Remove("doc");

        if (!openWith.ContainsKey("doc"))
        {
            Console.WriteLine("Key \"doc\" is not found.");
        }
    }
}

/* This code example produces the following output:

An element with Key = "txt" already exists.
For key = "rtf", value = wordpad.exe.
For key = "rtf", value = winword.exe.
Value added for key = "ht": hypertrm.exe

Key = dib, Value = paint.exe
Key = txt, Value = notepad.exe
Key = ht, Value = hypertrm.exe
Key = bmp, Value = paint.exe
Key = rtf, Value = winword.exe
Key = doc, Value = winword.exe

Value = paint.exe
Value = notepad.exe
Value = hypertrm.exe
Value = paint.exe
Value = winword.exe
Value = winword.exe

Key = dib
Key = txt
Key = ht
Key = bmp
Key = rtf
Key = doc

Remove("doc")
Key "doc" is not found.
 */
```

PowerShellCopiar

```
# Create new hash table using PowerShell syntax
$OpenWith = @{}

# Add one element to the hash table using the Add method
$OpenWith.Add('txt', 'notepad.exe')

# Add three eleements using PowerShell syntax three different ways
```

```
$OpenWith.dib = 'paint.exe'

$KeyBMP = 'bmp'
$OpenWith[$KeyBMP] = 'paint.exe'

$OpenWith += @{'rtf' = 'wordpad.exe'}

# Display hash table
"There are {0} in the `$OpenWith hash table as follows:" -f $OpenWith.Count
''

# Display hashtable properties
'Count of items in the hashtable  : {0}' -f $OpenWith.Count
'Is hashtable fixed size?         : {0}' -f $OpenWith.IsFixedSize
'Is hashtable read-only?          : {0}' -f $OpenWith.IsReadonly
'Is hashtabale synchronised?      : {0}' -f $OpenWith.IsSynchronized
''
'Keys in hashtable:'
$OpenWith.Keys
''
'Values in hashtable:'
$OpenWith.Values
''

<#
This script produces the following output:

There are 4 in the $OpenWith hash table as follows:

Name                           Value
----                           -----
txt                            notepad.exe
dib                            paint.exe
bmp                            paint.exe
rtf                            wordpad.exe

Count of items in the hashtable  : 4
Is hashtable fixed size?         : False
Is hashtable read-only?          : False
Is hashtabale synchronised?      : False

Keys in hashtable:
txt
dib
bmp
rtf

Values in hashtable:
notepad.exe
```

```
paint.exe
paint.exe
wordpad.exe
#>
```

Comentarios

Cada elemento es un par clave/valor almacenado en
un DictionaryEntry objeto. No puede ser una clave null, pero puede ser un valor.

Importante

No se recomienda que utilice el Hashtable clase para el nuevo desarrollo. En su lugar, se recomienda que use el tipo genérico Dictionary<TKey,TValue> clase.

Objetos de clave deben ser inmutable, siempre se utilizan como claves en
el Hashtable.

Cuando se agrega un elemento a la Hashtable, el elemento se coloca en un depósito en función del código hash de la clave. Las búsquedas subsiguientes de la clave de usar el código hash de la clave para buscar en sólo un depósito determinado, lo que reduce sustancialmente el número de comparaciones claves necesarios para buscar un elemento.

El factor de carga de un Hashtable determina la relación máxima de elementos por sectores de almacenamiento. Factores de carga más pequeños provocar tiempos de más rápido promedio búsqueda a costa de consumo de memoria mayor. El factor de carga predeterminado de 1,0 suele ofrece el mejor equilibrio entre velocidad y tamaño. También puede ser un factor de carga diferentes que se especificó cuando la Hashtable se crea.

Cuando se agregan elementos a un Hashtable, el factor de carga real de
la Hashtable aumenta.Cuando el factor de carga real alcanza el factor de carga especificado, el número de depósitos en los Hashtable aumenta automáticamente hasta el número primo más pequeño que sea mayor que el doble del número actual de Hashtable depósitos.

Cada objeto de clave en el Hashtable debe proporcionar su propia función hash, que se puede acceder mediante una llamada a GetHash. Sin embargo, cualquier objeto que implemente IHashCodeProvider puede pasarse a
una Hashtable constructor y que se usa la función hash para todos los objetos en la tabla.

La capacidad de un Hashtable es el número de elementos de
la Hashtable puede contener. Cuando se agregan elementos a un Hashtable, automáticamente se aumenta la capacidad según sea necesario mediante la reasignación.