Nifty trick: Combining constructor with collection initializer

devblogs.microsoft.com/oldnewthing/20181127-00

November 27, 2018

Raymond Chen

C# provides a number of ways of initializing collections.

If a collection has a single-parameter `Add` method, you can add items into the collection as part of initialization:

```
var list = new List<int> { 1, 2, 3 };

// Equivalent to
var list = new List<int>();
list.Add(1);
list.Add(2);
list.Add(3);
```

Note that if you do not provide an argument list after the type, one will be provided for you, namely `()` .

If a collection has a multi-parameter `Add` method, you can add items into the collection as part of the initialization, but you need to put braces around each parameter list:

```
var dict = new Dictionary<int, int> {
    { 0, 1 },
    { 1, 2 },
};

// Equivalent to
var dict = new Dictionary<int, int>();
dict.Add(0, 1);
dict.Add(1, 2);
```

If a collection has an index setter, you can add items into the collection with indexer syntax:

```
var dict = new Dictionary<int, int> {
    [0] = 1,
    [1] = 2,
};

// Equivalent to
var dict = new Dictionary<int, int>();
dict[0] = 1;
dict[1] = 2;
```

You cannot combine these different initializer notations, however.

```
// Code in italics doesn't compile
var dict = new Dictionary<int, int> {
    {0, 1 },
    [1] = 2,
};
```

However, one thing that is sometimes interesting to do is combine the constructor with the collection initializer. This lets you clone a collection and then modify it.

```
// Resulting list is { 1, 2, 3, 4, 5 }
var list2 = new List<int>(list) { 4, 5 };

// Resulting list is { 4, 2, 3 }
var list3 = new List<int>(list) { [0] = 4 };

// Resulting dictionary is dict2[0] = 1, dict2[1] = 2,
// and dict[2] = 3
var dict2 = new Dictionary<int, int>(dict) { [2] = 3 };

// Resulting dictionary is dict2[0] = 4, dict2[1] = 2
var dict2 = new Dictionary<int, int>(dict) { [0] = 4 };
```

You can pass anything that is a valid constructor parameter. For example, `List<T>` permits construction from any enumerable, so you can do this:

```
string[] ab = new string[] { "a", "b" };
List<string> abcd = new List<string>(ab) { "c", "d" };
// abcd has four elements: "a" "b" "c" and "d"
```

This combination notation is useful if you want to clone an existing collection and then make some tweaks to it.

Raymond Chen

Follow