Mahesh Chand Oct 03 2018 Article

16 6 4.1k

At some point, your apps consume a list of data. Manipulating these lists is a key to your application's success. .NET provides powerful list manipulation in the template-based List object.

In this article and code examples, I will review some key features of the List object including adding List items, find an item in List, reverse List, and sort List items C# in .NET and apply it to common scenarios, so you can use it in your applications and avoid writing unnecessary code.

The C# List<T> Object

List<T> class in C# represents a strongly typed list of objects. List<T> provides functionality to create a list of objects, find list items, sort list, search list, and manipulate list items. In List<T>, T is the type of objects.

What is the 'T'?

List<T> class in represents a strongly typed list of objects. List<T> provides functionality to create a list of objects, find list items, sort list, search list, and manipulate list items. In List<T>, T is the type of objects.

Creating List<T> object

List<T> is a generic class and is defined in the System.Collections.Generic namespace. You must import this namespace in your project to access the List<T> class.

```
01. using System.Collections.Generic;
```

List<T> class constructor is used to create a List object of type T. It can either be empty or take an Integer value as an argument that defines the initial size of the list, also known as capacity. If there is no integer passed in the constructor, the size of the list is dynamic and grows every time an item is added to the array. You can also pass an initial collection of elements when initialize an object.

The code snippet in Listing 1 creates a List of Int16 and a list of string types. The last part of the code creates a List<T> object with an existing collection.

```
01. // List with default capacity
02. List<Int16> list = new List<Int16>();
03.
04. // List with capacity = 5
05. List<string> authors = new List<string>(5);
06.
07. string[] animals = { "Cow", "Camel", "Elephant" };
```

Listing 1

Adding Items to a List<T>

The Add method adds an element to a List. The code snippet in Listing 2 creates two List<T> objects and adds integer and string items to them respectively.

```
01. // Dynamic ArrayList with no size limit
02. List<int> numberList = new List<int>();
03. numberList.Add(32);
04. numberList.Add(21);
05. numberList.Add(45);
06. numberList.Add(11);
07. numberList.Add(89);
08.
09. // List of string
10. List<string> authors = new List<string>(5);
11. authors.Add("Mahesh Chand");
12. authors.Add("Chris Love");
13. authors.Add("Allen O'neill");
14. authors.Add("Naveen Sharma");
15. authors.Add("Monica Rathbun");
16. authors.Add("David McCarter");
```

Listing 2

We can also add a collection to a List. The AddRange method is used to add a collection to a List. The code snippet in Listing 3 adds an array of strings to a List.

```
01. // Collection of string
02. string[] animals = { "Cow", "Camel", "Elephant" };
03. // Create a List and add a collection
04. List<string> animalsList = new List<string>();
05. animalsList.AddRange(animals);
06. foreach (string a in animalsList)
07. Console.WriteLine(a);
```

Listing 3

Reading a List<T> Items

List<T> is a collection. We can use a foreach loop to loop through its items. The code snippet in Listing 6 reads all items of a List and displays on the console.

```
01. foreach (string a in authors)
02. Console.WriteLine(a);
```

Listing 4

The output of Listing 4 looks like Figure 1.

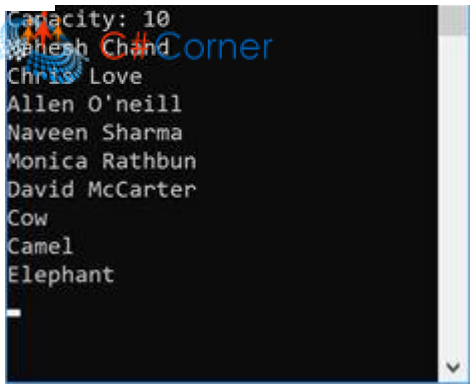

Figure 1.

To retrieve an item at a specific position in the List, we can use the collection's index. The following code snippet reads the 3rd items in the List.

```
01. Console.WriteLine(authors[2]);
```

List<T> Properties

List<T> class properties include the following:

- Capacity – Number of elements List<T> can contain. The default capacity of a List<T> is 0.
- Count – Number of elements in List<T>.

Code snippet in Listing 5 gets the value of these properties.

```
01. ArrayList authorsArray = new ArrayList();
02. authorsArray.Add("Mahesh Chand");
03. authorsArray.Add("Praveen Kumar");
04. authorsArray.Add("Raj Kumar");
05. authorsArray.Add("Dinesh Beniwal");
06. authorsArray.Add("David McCarter");
07.
08. Console.WriteLine("Count: " + authors.Count);
09. Console.WriteLine("Capacity: " + authors.Capacity);
```

Listing 5

Insert Items at a Position in a List<T>

The Insert method of List<T> class inserts an object at a given position. The first parameter of the method is the 0th based index in the List.
The InsertRange method can insert a collection at the given position.

The code snippet in Listing 6 inserts a string at the 3rd position and an array at the 2nd position of the List<T>.

```
01. authors.Insert(3, "Bill Author");
02.
03. // Collection of new authors
04. string[] newAuthors = { "New Author1", "New Author2", "New Author3" };
05.
06. // Insert array at position 2
```

cus

List<T> class also has Remove methods that can be u a

The Remove method removes the first occurrence of the given item in the List.

The following code snippet removes the first occurrence of 'New Author1'.

```
01. // Remove an item
02. authors.Remove("New Author1");
```

The RemoveAt method removes an item at the given position. The following code snippet removes the item at the 3rd position.

```
01. // Remove 3rd item
02. authors.RemoveAt(3);
```

The RemoveRange method removes a list of items from the starting index to the number of items. The following code snippet removes two items starting at 3rd position.

```
01. // Remove a range
02. authors.RemoveRange(3, 2);
```

The Clear method removes all items from a List<T>. The following code snippet removes all items from a List.

```
01. // Remove all items
02. authors.Clear();
```

Find an Item in a List<T>

The IndexOf method finds an item in a List. The IndexOf method returns -1 if there are no items found in the List.

The following code snippet finds a string and returns the matched position of the item.

```
01. int idx = authors.IndexOf("Naveen Sharma");
02. if (idx > 0)
03. Console.WriteLine($"Item index in List is: {idx}");
04. else
05. Console.WriteLine("Item not found");
```

We can also specify the position in a List where IndexOf method can start searching from.

For example, the following code snippet finds a string starting at the 3rd position in a String.

```
01. Console.WriteLine(authors.IndexOf("Naveen Sharma", 2));
```

The LastIndexOf method finds an item from the end of List.

The following code snippet looks for a string in the backward direction and returns the index of the item if found.

```
01. Console.WriteLine(authors.LastIndexOf("Mahesh Chand"));
```

```
01. // List of string
```

```
authors.Add("Allen O'neill");
authors.Add("Naveen Sharma");
authors.Add("Mahesh Chand");
authors.Add("Monica Rathbun");
authors.Add("David McCarter");

int idx = authors.IndexOf("Naveen Sharma");
if (idx > 0)
Console.WriteLine($"Item index in List is: {idx}");
else
Console.WriteLine("Item not found");

Console.WriteLine(authors.IndexOf("Naveen Sharma", 2));
Console.WriteLine(authors.LastIndexOf("Mahesh Chand"));
```

Listing 7.

Sort a List Items

The Sort method of List<T> sorts all items of the List using the QuickSort algorithm.

The following code example in Listing 8 sorts a List items and displays both original order and sorted order of the List items.

```
// List of string
List<string> authors = new List<string>(5);
authors.Add("Mahesh Chand");
authors.Add("Chris Love");
authors.Add("Allen O'neill");
authors.Add("Naveen Sharma");
authors.Add("Mahesh Chand");
authors.Add("Monica Rathbun");
authors.Add("David McCarter");

Console.WriteLine("Original List items");
Console.WriteLine("===============");
// Print original order
foreach (string a in authors)
Console.WriteLine(a);

// Sort list items
authors.Sort();

Console.WriteLine();
Console.WriteLine("Sorted List items");
Console.WriteLine("===============");
// Print sorted items
foreach (string a in authors)
Console.WriteLine(a);
```

Listing 8.

The output of Listing 8 looks like Figure 2.

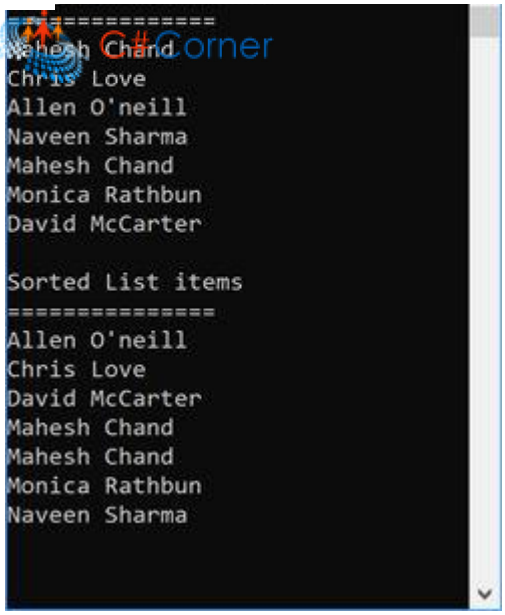

Figure 2.

Reverse an ArrayList

The Reverse method of List<T> reverses the order all items in in the List.

The following code snippet reverses a List.

```
01. // List of string
02. List<string> authors = new List<string>(5);
03. authors.Add("Mahesh Chand");
04. authors.Add("Chris Love");
05. authors.Add("Allen O'neill");
06. authors.Add("Naveen Sharma");
07. authors.Add("Mahesh Chand");
08. authors.Add("Monica Rathbun");
09. authors.Add("David McCarter");
10.
11. Console.WriteLine("Original List items");
12. Console.WriteLine("===============");
13. // Print original order
14. foreach (string a in authors)
15. Console.WriteLine(a);
16.
17. // Reverse list items
18. authors.Reverse();
19.
20. Console.WriteLine();
21. Console.WriteLine("Sorted List items");
22. Console.WriteLine("===============");
23. // Print reversed items
24. foreach (string a in authors)
25. Console.WriteLine(a);
```

Listing 8.

The output of Listing 8 looks like Figure 3.

Figure 3.

Search a List<T>

The BinarySearch method of List<T> searches a sorted list and returns the zero-based index of the found item. The List<T> must be sorted before this method can be used.

The following code snippet returns an index of a string in a List.

```
01. int bs = authors.BinarySearch("Mahesh Chand");
```

Summary

This article demonstrated how to use a List<T> class to work with a collection of objects. The article also demonstrated how to add, find, search, sort, and reverse items in a List.

.NET List Object | C# List | ListT | ListT Class

Mahesh Chand Admin

C# Corner Founder, Author, International Speaker, and Startup Adviser. My role as the CEO of Mindcracker Inc, I help businesses build their next generation digital platforms and help with their product innovation and gro... Read more

https://www.c-sharpcorner.com

134.3m **13**

16 **6**