Commonly used collection types

Article • 09/15/2021 • 2 minutes to read

Collection types represent different ways to collect data, such as hash tables, queues, stacks, bags, dictionaries, and lists.

All collections are based on the ICollection or ICollection<T> interfaces, either directly or indirectly. IList and IDictionary and their generic counterparts all derive from these two interfaces.

In collections based on IList or directly on ICollection, every element contains only a value. These types include:

- Array
- ArrayList
- List<T>
- Queue
- ConcurrentQueue<T>
- Stack
- ConcurrentStack<T>
- LinkedList<T>

In collections based on the IDictionary interface, every element contains both a key and a value. These types include:

- Hashtable
- SortedList
- SortedList<TKey,TValue>
- Dictionary<TKey,TValue>
- ConcurrentDictionary<TKey,TValue>

The KeyedCollection<TKey,TItem> class is unique because it is a list of values with keys embedded within the values. As a result, it behaves both like a list and like a dictionary.

When you need efficient multi-threaded collection access, use the generic collections in the

System.Collections.Concurrent namespace.

The Queue and Queue<T> classes provide first-in-first-out lists. The Stack and Stack<T> classes provide last-in-first-out lists.

Strong typing

Generic collections are the best solution to strong typing. For example, adding an element of any type other than an Int32 to a `List<Int32>` collection causes a compile-time error. However, if your language does not support generics, the System.Collections namespace includes abstract base classes that you can extend to create collection classes that are strongly typed. These base classes include:

- CollectionBase
- ReadOnlyCollectionBase
- DictionaryBase

How collections vary

Collections vary in how they store, sort, and compare elements, and how they perform searches.

The SortedList class and the SortedList<TKey,TValue> generic class provide sorted versions of the Hashtable class and the Dictionary<TKey,TValue> generic class.

All collections use zero-based indexes except Array, which allows arrays that are not zero-based.

You can access the elements of a SortedList or a KeyedCollection<TKey,TItem> by either the key or the element's index. You can only access the elements of a Hashtable or a Dictionary<TKey,TValue> by the element's key.

Use LINQ with collection types

The LINQ to Objects feature provides a common pattern for accessing in-memory objects of any type that implements IEnumerable or IEnumerable<T>. LINQ queries have several benefits over standard constructs like `foreach` loops:

- They are concise and easier to understand.
- They can filter, order, and group data.

- They can filter, order, and group data.
- They can improve performance.

For more information, see LINQ to Objects (C#), LINQ to Objects (Visual Basic), and Parallel LINQ (PLINQ).

Related topics

Title	Description
Collections and Data Structures	Discusses the various collection types available in .NET, including stacks, queues, lists, arrays, and dictionaries.
Hashtable and Dictionary Collection Types	Describes the features of generic and nongeneric hash-based dictionary types.
Sorted Collection Types	Describes classes that provide sorting functionality for lists and sets.
Generics	Describes the generics feature, including the generic collections, delegates, and interfaces provided by .NET. Provides links to feature documentation for C#, Visual Basic, and Visual C++, and to supporting technologies such as reflection.

Reference

System.Collections

System.Collections.Generic

System.Collections.ICollection

System.Collections.Generic.ICollection<T>

System.Collections.IList

System.Collections.Generic.IList<T>

System.Collections.IDictionary

System.Collections.Generic.IDictionary<TKey,TValue>