Selecting a Collection Class

Article • 09/15/2021 • 3 minutes to read

Be sure to choose your collection class carefully. Using the wrong type can restrict your use of the collection.

> **Important**
>
> Avoid using the types in the **System.Collections** namespace. The generic and concurrent versions of the collections are recommended because of their greater type safety and other improvements.

Consider the following questions:

- Do you need a sequential list where the element is typically discarded after its value is retrieved?
  - If yes, consider using the Queue class or the Queue<T> generic class if you need first-in, first-out (FIFO) behavior. Consider using the Stack class or the Stack<T> generic class if you need last-in, first-out (LIFO) behavior. For safe access from multiple threads, use the concurrent versions, ConcurrentQueue<T> and ConcurrentStack<T>. For immutability, consider the immutable versions, ImmutableQueue<T> and ImmutableStack<T>.
  - If not, consider using the other collections.
- Do you need to access the elements in a certain order, such as FIFO, LIFO, or random?
  - The Queue class, as well as the Queue<T>, ConcurrentQueue<T>, and ImmutableQueue<T> generic classes all offer FIFO access. For more information, see When to Use a Thread-Safe Collection.
  - The Stack class, as well as the Stack<T>, ConcurrentStack<T>, and ImmutableStack<T> generic classes all offer LIFO access. For more information, see When to Use a Thread-Safe Collection.
  - The LinkedList<T> generic class allows sequential access either from the head to

the tail, or from the tail to the head.

- Do you need to access each element by index?
  - The ArrayList and StringCollection classes and the List<T> generic class offer access to their elements by the zero-based index of the element. For immutability, consider the immutable generic versions, ImmutableArray<T> and ImmutableList<T>.
  - The Hashtable, SortedList, ListDictionary, and StringDictionary classes, and the Dictionary<TKey,TValue> and SortedDictionary<TKey,TValue> generic classes offer access to their elements by the key of the element. Additionally, there are immutable versions of several corresponding types: ImmutableHashSet<T>, ImmutableDictionary<TKey,TValue>, ImmutableSortedSet<T>, and ImmutableSortedDictionary<TKey,TValue>.
  - The NameObjectCollectionBase and NameValueCollection classes, and the KeyedCollection<TKey,TItem> and SortedList<TKey,TValue> generic classes offer access to their elements by either the zero-based index or the key of the element.
- Will each element contain one value, a combination of one key and one value, or a combination of one key and multiple values?
  - One value: Use any of the collections based on the IList interface or the IList<T> generic interface. For an immutable option, consider the IImmutableList<T> generic interface.
  - One key and one value: Use any of the collections based on the IDictionary interface or the IDictionary<TKey,TValue> generic interface. For an immutable option, consider the IImmutableSet<T> or IImmutableDictionary<TKey,TValue> generic interfaces.
  - One value with embedded key: Use the KeyedCollection<TKey,TItem> generic class.
  - One key and multiple values: Use the NameValueCollection class.
- Do you need to sort the elements differently from how they were entered?
  - The Hashtable class sorts its elements by their hash codes.
  - The SortedList class, and the SortedList<TKey,TValue> and SortedDictionary<TKey,TValue> generic classes sort their elements by the key. The sort order is based on the implementation of the IComparer interface for the

sort order is based on the implementation of the IComparer interface for the SortedList class and on the implementation of the IComparer<T> generic interface for the SortedList<TKey,TValue> and SortedDictionary<TKey,TValue> generic classes. Of the two generic types, SortedDictionary<TKey,TValue> offers better performance than SortedList<TKey,TValue>, while SortedList<TKey,TValue> consumes less memory.

    - ArrayList provides a Sort method that takes an IComparer implementation as a parameter. Its generic counterpart, the List<T> generic class, provides a Sort method that takes an implementation of the IComparer<T> generic interface as a parameter.

- Do you need fast searches and retrieval of information?
    - ListDictionary is faster than Hashtable for small collections (10 items or fewer). The Dictionary<TKey,TValue> generic class provides faster lookup than the SortedDictionary<TKey,TValue> generic class. The multi-threaded implementation is ConcurrentDictionary<TKey,TValue>. ConcurrentBag<T> provides fast multi-threaded insertion for unordered data. For more information about both multi-threaded types, see When to Use a Thread-Safe Collection.

- Do you need collections that accept only strings?

    - StringCollection (based on IList) and StringDictionary (based on IDictionary) are in the System.Collections.Specialized namespace.

    - In addition, you can use any of the generic collection classes in the System.Collections.Generic namespace as strongly typed string collections by specifying the String class for their generic type arguments. For example, you can declare a variable to be of type List<String> or Dictionary<String,String>.

LINQ to Objects and PLINQ

LINQ to Objects enables developers to use LINQ queries to access in-memory objects as long as the object type implements IEnumerable or IEnumerable<T>. LINQ queries provide a common pattern for accessing data, are typically more concise and readable than standard `foreach` loops, and provide filtering, ordering, and grouping capabilities. For more information, see LINQ to Objects (C#) and LINQ to Objects (Visual Basic).

PLINQ provides a parallel implementation of LINQ to Objects that can offer faster query execution in many scenarios, through more efficient use of multi-core computers. For more information, see Parallel LINQ (PLINQ).

information, see Parallel LINQ (PLINQ).

See also

- System.Collections
- System.Collections.Specialized
- System.Collections.Generic
- Thread-Safe Collections