Основы программирования

Коллекции

...

...

Массивы в языке C#, хранят наборы объектов однотипных объектов, однако работать с ними не всегда удобно.

Так массив хранит фиксированное количество объектов, но заранее не всегда известно, сколько элементов будет.

Для решения этой проблемы реализованы коллекции. Коллекция является классом.

Некоторые из коллекций реализует стандартные структуры данных, такие как стек, очередь, словарь …

Эти коллекции могут пригодиться для решения различных специальных задач.

Пространства имен коллекций

Большая часть классов коллекций содержится в пространствах имен:

- `System.Collections` -- простые необобщенные классы коллекций;
- `System.Collections.Generic` -- обобщенные или типизированные классы коллекций;
- `System.Collections.Concurrent` -- для обеспечения параллельного выполнения задач и многопоточного доступа.
- `System.Collections.Specialized` -- специальные классы коллекций.

Определение

Это совокупность объектов с расширенной функциональностью.

Использование коллекций упрощает решение многих задач, предлагая готовые решения для создания целого ряда типичных, но порой трудоемких для разработки структур данных.

Для универсальности коллекции хранят объекты типа `Object`.

Назначение коллекций

Стандартизируют обработку групп объектов.

Все коллекции разработаны на основе набора четко определенных интерфейсов.

Некоторые встроенные реализации таких интерфейсов, в том числе `ArrayList`, `Hashtable`, `Stack` и `Queue`, могут применяться в исходном виде и без каких-либо изменений.

Имеется также возможность реализовать собственную коллекцию.

Типы коллекций

- Обобщенные *)
- Необобщенные
- Специальные
- Параллельные
- Поразрядные

*) -- См. Обобщенные типы: https://metanit.com/sharp/tutorial/3.12.php

Обобщенные коллекции

В обобщенной коллекции могут храниться только такие элементы данных, которые совместимы по типу с данной коллекцией.

Классы обобщенных (типизированных) коллекций определены в пространстве имен `System.Collections.Generic`

Функционал по большей части описывается в обобщенных интерфейсах.

Если вам не надо хранить объекты разных типов, то предпочтительнее использовать обобщенные коллекции.

Интерфейсы обобщенных коллекций отличаются от необобщенных не только наличием универсального параметра `T`, но и самой функциональностью.

Обобщенные коллекции это

- списки,
- очереди,
- стэк,
- словари
- другие классы для добавления, удаления, сортировки элементов.

Примеры обобщенных коллекций

- `List<T>` – представляет последовательный список.

Реализует интерфейсы: `IList<T>`, `ICollection<T>`, `IEnumerable<T>`

- `LinkedList<T>` – двухсвязный список.

Реализует интерфейсы `ICollection<T>` и `IEnumerable<T>`

- `Dictionary<TKey, TValue>` – хранит наборы пар "ключ-значение".

Реализует интерфейсы `ICollection<T>`, `IEnumerable<T>`, `IDictionary<TKey, TValue>`

Примеры обобщенных коллекций

- `Queue<T>` – очередь объектов, работающая по алгоритму `FIFO` ("первый вошел-первый вышел").

Реализует интерфейсы `ICollection`, `IEnumerable<T>`

- `SortedSet<T>` – отсортированная коллекция однотипных объектов.

Реализует интерфейсы `ICollection<T>`, `ISet<T>`, `IEnumerable<T>`

- `SortedList<TKey, TValue>` – хранит наборы пар "ключ-значение", отсортированных по ключу.

Реализует интерфейсы `ICollection<T>`, `IEnumerable<T>`, `IDictionary<TKey, TValue>`

Примеры обобщенных коллекций

- `SortedDictionary<TKey, TValue>` – хранит наборы пар "ключ-значение", отсортированных по ключу.

В общем похож на класс `SortedList<TKey, TValue>`, основные отличия состоят лишь в использовании памяти и в скорости вставки и удаления.

- `Stack<T>` – стек однотипных объектов.

Реализует интерфейсы `ICollection<T>` и `IEnumerable<T>`

Основные интерфейсы обобщенных коллекций

- `IEnumerable<T>` – позволяет перебирать элементы коллекции с помощью цикла `foreach`, определяя метод `GetEnumerator()`, с помощью которого можно получать элементы любой коллекции (перечислитель).
- `IEnumerator<T>` – определяет методы, с помощью которых потом можно поэлементно получать содержимое коллекции.
- `ICollection<T>` – представляет ряд общих свойств и методов для всех обобщенных коллекций
  Например, методы `CopyTo`, `Add`, `Remove`, `Contains`, свойство `Count`.
- `IList<T>` – предоставляет функционал для создания последовательных списков.

Основные интерфейсы обобщенных коллекций

- `IComparer<T>` – определяет метод `Compare()` для сравнения двух однотипных объектов.
- `IEqualityComparer<T>` – определяет методы, с помощью которых два однотипных объекта сравниваются на предмет равенства.
- `IDictionary<TKey, TValue>` – определяет поведение коллекции, при котором она должна хранить объекты в виде пар “ключ-значение”
  (для каждого объекта определяется уникальный ключ типа, указанного в параметре `TKey`, и этому ключу соответствует определенное значение, имеющее тип, указанный в параметре `Tvalue)`.

Основные интерфейсы обобщенных коллекций

- `IDictionaryEnumerator` – определяет нумератор для коллекции реализующей интерфейс `IDictionary`.
- `IHashCodeProvider` – определяет хеш-функцию.

Необобщеные коллекции

Классы и интерфейсы необобщенных коллекций определены в пространстве имен `System.Collections`

- динамический массив,
- стек,
- очередь,
- словари.

Эти коллекции не типизированы и оперируют данными типа `object`.

Служат для хранения данных любого типа, причем в одной коллекции допускается наличие разнотипных данных.

Специализированные коллекции

Определены в пространстве имен `System.Collections.Specialized`

Существуют специальные коллекции символьных строк, а также специальные коллекции, в которых используется однонаправленный список.

Оперируют данными конкретного типа или же делают это каким-то особым образом.

Поразрядная коллекция

Определена только одна коллекция `BitArray` с поразрядной организацией.

Поддерживает поразрядные операции -- операции над двоичными разрядами, например И, ИЛИ, исключающее ИЛИ.

Существенно отличается своими возможностями от остальных типов коллекций.

Коллекция типа `BitArray` объявляется в пространстве имен `System.Collections`.

Параллельные коллекции

Поддерживают многопоточный доступ к коллекции.

Это обобщенные коллекции, определенные в пространстве имен `System.Collections.Concurrent`.

Коллекция ArrayList

Класс `ArrayList` представляет коллекцию объектов и позволяет хранить вместе объекты разных типов -- строки, числа и т.д.

Определен в пространстве имен `System.Collections`

```
…
ArrayList list = new ArrayList();
list.Add(2.3); // добавляем объект типа double
list.Add(55); // добавляем объект типа int
…
```

Подробнее: C# и .NET | Коллекции. ArrayList (metanit.com)
https://metanit.com/sharp/tutorial/4.3.php

Список List<T>

Класс `List<T>` представляет простейший список однотипных объектов.

```
…
List<int> numbers = new List<int>() { 1, 2, 3};
numbers.Add(4); // добавление элемента
…
```

Подробнее: C# и .NET | Список List (metanit.com)
https://metanit.com/sharp/tutorial/4.5.php

Двухсвязный список LinkedList<T>

Класс `LinkedList<T>` представляет двухсвязный список, в котором каждый элемент хранит ссылку одновременно на следующий и на предыдущий элемент.

В списке `LinkedList<T>` каждый узел представляет объект класса `LinkedListNode<T>`.

…

```
LinkedList<int> numbers = new LinkedList<int>();
numbers.AddLast(1); // вставляем «1» в начало?
numbers.AddFirst(2); // вставляем «2» в начало
```

…

Подробнее: C# и .NET | Двухсвязный список LinkedList (metanit.com)
https://metanit.com/sharp/tutorial/4.6.php

Очередь Queue<T>

Класс `Queue<T>` представляет обычную очередь, работающую по алгоритму `FIFO` ("первый вошел - первый вышел").

Методы:

- `Dequeue` - извлекает и возвращает первый элемент очереди;
- `Enqueue` - добавляет элемент в конец очереди;
- `Peek` - просто возвращает первый элемент из начала очереди без его удаления.

Очередь Queue<T>

```
…
Queue<int> numbers = new Queue<int>();
// добавляем элемент в конец очереди
numbers.Enqueue(3); // очередь 3
numbers.Enqueue(5); // очередь 3, 5
numbers.Enqueue(8); // очередь 3, 5, 8
// получаем первый элемент очереди
int queueElement = numbers.Dequeue();
                   //очередь 5, 8
…
```

Подробнее: C# и .NET | Коллекция Queue (metanit.com)
https://metanit.com/sharp/tutorial/4.7.php

Коллекция Stack<T>

Класс Stack<T> представляет коллекцию, которая использует алгоритм LIFO ("последний вошел - первый вышел").

При такой организации каждый следующий добавленный элемент помещается поверх предыдущего. Извлечение из коллекции происходит в обратном порядке - извлекается тот элемент, который находится выше всех в стеке.

Основные методы:

- `Push` - добавляет элемент в стек на первое место;
- `Pop` - извлекает и возвращает первый элемент из стека;
- `Peek` - просто возвращает первый элемент из стека без его удаления.

Коллекция Stack<T>

```
…
Stack<int> numbers = new Stack<int>();
numbers.Push(3); // в стеке 3
numbers.Push(5); // в стеке 5, 3
numbers.Push(8); // в стеке 8, 5, 3
// так как вверху стека будет находиться число 8,
// оно и извлекается
int stackElement = numbers.Pop(); // в стеке 5, 3
…
```

Подробнее: C# и .NET | Коллекция Stack (metanit.com)
https://metanit.com/sharp/tutorial/4.8.php

Коллекция Dictionary<T, V>

Словарь хранит объекты, которые представляют пару “ключ-значение”.

Каждый такой объект является объектом структуры KeyValuePair<TKey, TValue>.

Благодаря свойствам Key и Value, которые есть у данной структуры, мы можем получить ключ и значение элемента в словаре.

Подробнее: C# и .NET | Коллекция Dictionary (metanit.com)
https://metanit.com/sharp/tutorial/4.9.php

Коллекция Dictionary<T, V>. Пример

Интерфейс IList

Определяет поведение коллекции, доступ к элементам которой разрешен посредством индекса с отсчетом от нуля.

Наследует интерфейс `ICollection`.

Этот интерфейс определяет индексатор, а также способы вставки и удаления элементов в определенные позиции (методы `Insert()` и `Remove()`). `IList<T>` унаследованы от `ICollection<T>`)

Методы интерфейса IList

`int Add (object obj)` – добавляет объект `obj`. Возвращает индекс сохраненного объекта.

`void Clear()` – удаляет все элементы из вызывающей коллекции.

`bool Contains (object obj)` – возвращает значение `true` , если вызывающая коллекция содержит объект.

`int IndexOf (object obj)` – возвращает индекс объекта `obj`, если объект содержится в вызывающей коллекции, если нет, то метод возвращает `-1`.

...

`void Insert (int ind, object obj)` – вставляет в вызывающую коллекцию объект `obj` по индексу, заданному параметром `ind`.

`void Remove(object obj)` – удаляет из вызывающей коллекции объект, расположенный по индексу, заданному параметром `ind`.

`void RemoveAt(int ind)` – удаляет первое вхождение объекта `obj` из вызывающей коллекции.

Интерфейс IDictionary

Интерфейс `IDictionary` определяет поведение коллекции, которая устанавливает соответствие между уникальными ключами и значениями. Ключ – это объект, который используется для получения соответствующего ему значения.

Следовательно, коллекция, которая реализует интерфейс `IDictionary`, служит для хранения пар «ключ-значение».

Сохраненную однажды пару можно затем извлечь по заданному ключу.

Методы интерфейса IDictionary

`void Add (object к, object v)` – добавляет в коллекцию пару ключ/значение, заданную параметрами `к` и `v`. Ключ к не должен быть нулевым. Если ключ к уже есть в коллекции, генерируется исключение типа `ArgumentException`.

`void Clear()` – удаляет все пары.

`bool Contains (object к)` – возвращает значение `true`, если коллекция содержит объект `к` в качестве ключа. В противном случае возвращает значение `false`.

`GetEnumerator()` – возвращает нумератор для вызывающей коллекции.

`void Remove (object к)` – удаляет элемент, ключ которого равен значению `к`.

Свойства интерфейса IDictionary

`bool isFixedSize {get;}`- равно значению `true` , если словарь имеет фиксированный размер

`bool isReadOnly {get;}`- равно значению `true` , если словарь предназначен только для чтения

`ICollection Keys {get;}`- получает коллекцию ключей

`ICollection Values {get;}`- получает коллекцию значений

Индексатор интерфейса IDictionary

В интерфейсе `IDictionary` определен следующий индексатор:

```
object this[object key] { get; set; }
```

Этот индексатор можно использовать для получения или установки значения элемента.

Его можно также использовать для добавления в коллекцию нового элемента.

«Индекс» в данном случае не является обычным индексом, а ключом элемента.

Интерфейсы IEnumerable, IEnumerator и IDictionaryEnumerator

Интерфейс `IEnumerable` должен быть реализован в любом классе, если в нем предполагается поддержка нумераторов. Все классы коллекций реализуют интерфейс `IEnumerable`, поскольку он наследуется интерфейсом `ICollection`.

В интерфейсе `IEnumerable` определен единственный метод с именем `GetEnumerator`()

Он возвращает нумератор для коллекции.

Кроме того, реализация интерфейса `IEnumerable` позволяет получить доступ к содержимому коллекции с помощью цикла `foreach`.

...

Интерфейс `IEnumerator` определяет действие любого нумератора.

Используя его методы, можно циклически опрашивать содержимое коллекции.

Для коллекций, в которых хранятся пары ключ/значение (т.е. словари), метод `GetEnumerator()` возвращает объект типа `IDictionaryEnumerator`, а не типа `IEnumerator`.

Класс `IDictionaryEnumerator` является производным от класса `IEnumerator` и распространяет свои функциональные возможности на словари.

Интерфейс IComparer

В интерфейсе `IComparer` определен метод `Compare` (), который позволяет сравнивать два объекта:

```
int Compare(object v1, object v2)
```

Интерфейс IHashCodeProvider

Интерфейс `IHashCodeProvider` должен быть реализован коллекцией, если программисту необходимо определить собственную версию метода `GetHashCode ()`.

Вспомните, что все объекты (для получения хеш-кода) наследуют метод `Object.GetHashCode()`, который используется по умолчанию.

Посредством реализации интерфейса `IHashCodeProvider` можно определить альтернативный метод.

Структура DictionaryEntry

Коллекции, для хранения пар ключ/значение, используют объект типа `DictionaryEntry`.

В структуре `DictionaryEntry` определены два свойства:

```
public object Key { get; set; }
public object Value { get; set; }
```

Объект создается с помощью конструктора:

```
public DictionaryEntry(object к, object v)
```

Здесь параметр `к` принимает ключ, а параметр `v` — значение.

Источники

- C# и .NET | Введение в коллекции (metanit.com) https://metanit.com/sharp/tutorial/4.1.php
- Обобщенные коллекции | C# (metanit.com) https://metanit.com/sharp/tutorial/4.4.php
- C# и .NET | Обобщения (metanit.com) https://metanit.com/sharp/tutorial/3.12.php
- Необобщенные коллекции | C# (metanit.com) https://metanit.com/sharp/tutorial/4.2.php
- C# и .NET | Интерфейсы IEnumerable и IEnumerator (metanit.com) https://metanit.com/sharp/tutorial/4.11.php