**1. Перечислите стандартные коллекции NET Framework.**

В среде .NET Framework поддерживаются пять типов коллекций:

Необобщенные, обобщенные, параллельные, специальные и с поразрядной организацией.

**2. Поясните принцип работы коллекции:**

a) Stack<T>

Создает стек. Обеспечивает такие же функциональные возможности, как

необобщенный класс Stack

b) Queue<T>

Создает очередь. Обеспечивает такие же функциональные возможности, как и

необобщенный класс Queue

c) HashSet<T>

Сохраняет ряд уникальных значений, используя хештаблицу

d) List<T>

Создает динамический массив. Обеспечивает такие же функциональные

возможности, как и необобщенный класс ArrayList

e) Dictionary<Tkey, TValue>

Сохраняет пары "ключ-значение". Обеспечивает такие же функциональные

возможности, как и необобщенный класс Hashtable

f) LinkedList<T>

Сохраняет элементы в двунаправленном списке

g) SortedDictionary<TKey, TValue>

Создает отсортированный список из пар "ключ-значение"

h) SortedList<TKey, TValue>

Создает отсортированный список из пар "ключ-значение". Обеспечивает такие

же функциональные возможности, как и необобщенный класс SortedList

i) SortedSet<T>

Создает отсортированное множество

**3. Охарактеризуйте необобщенные, специальные, с поразрядной организацией, обобщенные и параллельные коллекции.**

Необобщенные коллекции

Реализуют ряд основных структур данных, включая динамический массив, стек, очередь, а также словари, в которых можно хранить пары "ключ-значение". Необобщенные коллекции оперируют данными типа object., а значит они могут служить для хранения данных любого типа, причем в одной коллекции допускается наличие разнотипных данных. Такие коллекции не типизированы, поскольку в них хранятся ссылки на данные типа object. Классы и интерфейсы необобщенных коллекций находятся в пространстве имен System.Collections.

Специальные коллекции

Оперируют данными конкретного типа или же делают это каким-то особым образом. Например, имеются специальные коллекции для символьных строк, а также специальные коллекции, в которых используется однонаправленный список. Специальные коллекции объявляются в пространстве

имен System.Collections.Specialized.

Поразрядная коллекция

В прикладном интерфейсе Collections API определена одна коллекция с поразрядной организацией — это BitArray. Коллекция типа BitArray поддерживает поразрядные операции, т.е. операции над отдельными двоичными разрядами, например И, ИЛИ, исключающее ИЛИ, а следовательно, она существенно отличается своими возможностями от остальных типов коллекций. Коллекция типа BitArray объявляется в пространстве имен System.Collections.

Обобщенные коллекции

Обеспечивают обобщенную реализацию нескольких стандартных структур данных, включая связные списки, стеки, очереди и словари. Эти коллекции - типизированы в силу их обобщенного характера, значит, что в ней могут храниться только такие элементы данных, которые совместимы по типу с данной коллекцией. Благодаря этому исключается случайное несовпадение типов. Они объявляются в пространстве имен System.Collections.Generic.

Параллельные коллекции

Поддерживают многопоточный доступ к коллекции. Это обобщенные коллекции, определенные в пространстве имен System.Collections.Concurrent.

В пространстве имен System.Collections.ObjectModel находится также ряд классов, поддерживающих создание пользователями собственных обобщенных коллекций.

**4. Какие интерфейсы используются в коллекциях C#?**

ICollection<T> Определяет основополагающие свойства обобщенных коллекци

IComparer<T> Определяет обобщенный метод Compare() для сравнения объектов, хранящихся

в коллекции

IDictionary<Tkey, TValue> Определяет обобщенную коллекцию, состоящую из пар "ключ-значение

IEnumerable<T> Определяет обобщенный метод GetEnumerator(), предоставляющий перечислитель для любого класса коллекции

Enumerator<T> Предоставляет методы, позволяющие получать содержимое коллекции по

очереди

IEqualityComparer<T> Сравнивает два объекта на предмет равенства

IList<T> Определяет обобщенную коллекцию, доступ к которой можно получить с

помощью индексатора

**5. Для чего используется интерфейс IComparable?**

Определяет обобщённый метод сравнения для типа, который реализуется типом значения или классом, чтобы упорядочить или отсортировать его экземпляры.

Содержит метод CompareTo(Object) сравнивает текущий экземпляр с другим объектом того же типа и возвращает целое число, которое показывает, расположен ли текущий экземпляр перед, после или на той же позиции в порядке сортировки что и другой объект.

**6. Что содержит интерфейс IEnumerator или обобщенный интерфейс IEnumerator<T>? Где и как его можно использовать?**

Определяет обобщенный метод GetEnumerator(), предоставляющий перечислитель для любого класса коллекции.

**7. Что такое наблюдаемая коллекция? Где и каким образом ее можно использовать?**

ObservableCollection<T>

· пользовательский интерфейс получает информацию об изменениях коллекции

· унаследован от Collection<T>, использует внутри себя List<T>

**1. На какие основные виды/типы делятся все коллекции .NET? Охарактеризуйте каждый из них.**

Необобщенные коллекции

Реализуют ряд основных структур данных, включая динамический массив, стек, очередь, а также словари, в которых можно хранить пары "ключ-значение". Необобщенные коллекции оперируют данными типа object., а значит они могут служить для хранения данных любого типа, причем в одной коллекции допускается наличие разнотипных данных. Такие коллекции не типизированы, поскольку в них хранятся ссылки на данные типа object. Классы и интерфейсы необобщенных коллекций находятся в пространстве имен System.Collections.

Специальные коллекции

Оперируют данными конкретного типа или же делают это каким-то особым образом. Например, имеются специальные коллекции для символьных строк, а также специальные коллекции, в которых используется однонаправленный список. Специальные коллекции объявляются в пространстве

имен System.Collections.Specialized.

Поразрядная коллекция

В прикладном интерфейсе Collections API определена одна коллекция с поразрядной организацией — это BitArray. Коллекция типа BitArray поддерживает поразрядные операции, т.е. операции над отдельными двоичными разрядами, например И, ИЛИ, исключающее ИЛИ, а следовательно, она существенно отличается своими возможностями от остальных типов коллекций. Коллекция типа BitArray объявляется в пространстве имен System.Collections.

Обобщенные коллекции

Обеспечивают обобщенную реализацию нескольких стандартных структур данных, включая связные списки, стеки, очереди и словари. Эти коллекции - типизированы в силу их обобщенного характера, значит, что в ней могут храниться только такие элементы данных, которые совместимы по типу с данной коллекцией. Благодаря этому исключается случайное несовпадение типов. Они объявляются в пространстве имен System.Collections.Generic.

Параллельные коллекции

Поддерживают многопоточный доступ к коллекции. Это обобщенные коллекции, определенные в пространстве имен System.Collections.Concurrent.

В пространстве имен System.Collections.ObjectModel находится также ряд классов, поддерживающих создание пользователями собственных обобщенных коллекций.

**2. Что такое generic-коллекции? Назовите примеры известных вам genericколлекций.**

Большая часть классов коллекций содержится в пространствах имен System.Collections (простые необобщенные классы коллекций), System.Collections.Generic (обобщенные или типизированные классы коллекций) и System.Collections.Specialized (специальные классы коллекций). Также для обеспечения параллельного выполнения задач и многопоточного доступа применяются классы коллекций из пространства имен System.Collections.Concurrent

Эти интерфейсы реализуются следующими классами коллекций в пространстве имен System.Collections.Generic:

* **List<T>**: класс, представляющий последовательный список. Реализует интерфейсы IList<T>, ICollection<T>, IEnumerable<T>
* **Dictionary<TKey, TValue>**: класс коллекции, хранящей наборы пар "ключ-значение". Реализует интерфейсы ICollection<T>, IEnumerable<T>, IDictionary<TKey, TValue>
* **LinkedList<T>**: класс двухсвязанного списка. Реализует интерфейсы ICollection<T> и IEnumerable<T>
* **Queue<T>**: класс очереди объектов, работающей по алгоритму FIFO("первый вошел -первый вышел"). Реализует интерфейсы ICollection, IEnumerable<T>
* **SortedSet<T>**: класс отсортированной коллекции однотипных объектов. Реализует интерфейсы ICollection<T>, ISet<T>, IEnumerable<T>
* **SortedList<TKey, TValue>**: класс коллекции, хранящей наборы пар "ключ-значение", отсортированных по ключу. Реализует интерфейсы ICollection<T>, IEnumerable<T>, IDictionary<TKey, TValue>
* **SortedDictionary<TKey, TValue>**: класс коллекции, хранящей 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>**: класс стека однотипных объектов. Реализует интерфейсы ICollection<T> и IEnumerable<T>

**3. В чем разница между ArrayList и Array?**

Это разные типы объектов. Они имеют разные возможности и хранят свои данные по-разному. Вы также можете спросить, в чем разница между десятичным и DateTime.

Массив (System.Array) имеет фиксированный размер после выделения. Вы не можете добавлять элементы к нему или удалять элементы из него. Также все элементы должны быть одного типа. В результате он безопасен по типу, а также является наиболее эффективным из трех с точки зрения памяти и производительности. Кроме того, System.Array поддерживает несколько измерений (т. Е. Имеет свойство [Rank](https://msdn.microsoft.com/en-us/library/system.array.rank(v=vs.110).aspx) ), а List и ArrayList - нет (хотя вы можете создать список списков или ArrayList из ArrayLists, если хотите).

ArrayList - это гибкий массив, который содержит список объектов. Вы можете добавлять и удалять элементы из него, и он автоматически занимается распределением пространства. Если вы храните типы значений в нем, они упакованы и распакованы, что может быть немного неэффективно. Кроме того, он не является типобезопасным.

A List<> использует дженерики; по сути, это типизированная версия ArrayList. Это означает, что нет никакой блокировки или распаковки (что повышает производительность), и если вы попытаетесь добавить элемент неправильного типа, это вызовет ошибку во время компиляции.

**4. Охарактеризуйте коллекции, которые вы использовали в своем варианте.**

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

Если в простом списке List<T> каждый элемент представляет объект типа T, то в LinkedList<T> каждый узел представляет объект класса LinkedListNode<T>. Этот класс имеет следующие свойства:

* **Value**: само значение узла, представленное типом T
* **Next**: ссылка на следующий элемент типа LinkedListNode<T> в списке. Если следующий элемент отсутствует, то имеет значение null
* **Previous**: ссылка на предыдущий элемент типа LinkedListNode<T> в списке. Если предыдущий элемент отсутствует, то имеет значение null

Используя методы класса LinkedList<T>, можно обращаться к различным элементам, как в конце, так и в начале списка:

* **AddAfter(LinkedListNode<T> node, LinkedListNode<T> newNode)**: вставляет узел newNode в список после узла node.
* **AddAfter(LinkedListNode<T> node, T value)**: вставляет в список новый узел со значением value после узла node.
* **AddBefore(LinkedListNode<T> node, LinkedListNode<T> newNode)**: вставляет в список узел newNode перед узлом node.
* **AddBefore(LinkedListNode<T> node, T value)**: вставляет в список новый узел со значением value перед узлом node.
* **AddFirst(LinkedListNode<T> node)**: вставляет новый узел в начало списка
* **AddFirst(T value)**: вставляет новый узел со значением value в начало списка
* **AddLast(LinkedListNode<T> node)**: вставляет новый узел в конец списка
* **AddLast(T value)**: вставляет новый узел со значением value в конец списка
* **RemoveFirst()**: удаляет первый узел из списка. После этого новым первым узлом становится узел, следующий за удаленным
* **RemoveLast()**: удаляет последний узел из списка

Класс List<T> из пространства имен System.Collections.Generic представляет простейший список однотипных объектов.

Среди его методов можно выделить следующие:

* **void Add(T item)**: добавление нового элемента в список
* **void AddRange(ICollection collection)**: добавление в список коллекции или массива
* **int BinarySearch(T item)**: бинарный поиск элемента в списке. Если элемент найден, то метод возвращает индекс этого элемента в коллекции. При этом список должен быть отсортирован.
* **int IndexOf(T item)**: возвращает индекс первого вхождения элемента в списке
* **void Insert(int index, T item)**: вставляет элемент item в списке на позицию index
* **bool Remove(T item)**: удаляет элемент item из списка, и если удаление прошло успешно, то возвращает true
* **void RemoveAt(int index)**: удаление элемента по указанному индексу index
* **void Sort()**: сортировка списка

**5. Чем отличаются коллекции, расположенные в пространстве имен System.Collections.Concurrent?**

Также для обеспечения параллельного выполнения задач и многопоточного доступа применяются классы коллекций из пространства имен System.Collections.Concurrent

**6. Какое пространство имен необходимо подключить в проект, чтобы иметь возможность использовать generic-коллекции?**

System.Collections.Generic

**7. Что такое наблюдаемая коллекция? Как ее можно использовать?**

ObservableCollection<T> Он по функциональности похож на список List за тем исключением, что позволяет известить внешние объекты о том, что коллекция была изменена.

· пользовательский интерфейс получает информацию об изменениях коллекции

· унаследован от Collection<T>, использует внутри себя List<T>

**8. Охарактеризуйте интерфейсы IEnumerator, IEnumerator. В чем отличие назначений интерфейсов IEnumerator и IEnumerable.**

Основой для создания всех коллекций является реализация интерфейсов **IEnumerator** и **IEnumerable** (и их обобщенных двойников **IEnumerator<T>** и **IEnumerable<T>**). Интерфейс IEnumerator представляет перечислитель, с помощью которого становится возможен последовательный перебор коллекции, например, в цикле **foreach**. А интерфейс IEnumerable через свой метод GetEnumerator предоставляет перечислитель всем классам, реализующим данный интерфейс. Поэтому интерфейс IEnumerable (IEnumerable<T>) является базовым для всех коллекций.

Интерфейс IEnumerable имеет метод, возвращающий ссылку на другой интерфейс - перечислитель:

|  |  |
| --- | --- |
| 1  2  3  4 | public interface IEnumerable  {      IEnumerator GetEnumerator();  } |

А интерфейс IEnumerator определяет функционал для перебора внутренних объектов в контейнере:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | public interface IEnumerator  {      bool MoveNext(); // перемещение на одну позицию вперед в контейнере элементов      object Current {get;}  // текущий элемент в контейнере      void Reset(); // перемещение в начало контейнера  } |

**9. Поясните принцип работы коллекций: a. LinkedList b. HashSet c. Dictionary d. ConcurrentBag e. Stack, Queue f. SortedList, SortedList.**

a) Stack<T>

Создает стек. Обеспечивает такие же функциональные возможности, как

необобщенный класс Stack

b) Queue<T>

Создает очередь. Обеспечивает такие же функциональные возможности, как и

необобщенный класс Queue

c) HashSet<T>

Сохраняет ряд уникальных значений, используя хештаблицу

d) List<T>

Создает динамический массив. Обеспечивает такие же функциональные

возможности, как и необобщенный класс ArrayList

e) Dictionary<Tkey, TValue>

Сохраняет пары "ключ-значение". Обеспечивает такие же функциональные

возможности, как и необобщенный класс Hashtable

f) LinkedList<T>

Сохраняет элементы в двунаправленном списке

g) SortedDictionary<TKey, TValue>

Создает отсортированный список из пар "ключ-значение"

h) SortedList<TKey, TValue>

Создает отсортированный список из пар "ключ-значение". Обеспечивает такие

же функциональные возможности, как и необобщенный класс SortedList

i) SortedSet<T>

Создает отсортированное множество