Необобщенные коллекции вошли в состав среды .NET Framework еще в версии 1.0.

Они определяются в пространстве имен System.Collections. Необобщенные коллекции

представляют собой структуры данных общего назначения, оперирующие ссылками

на объекты. Таким образом, они позволяют манипулировать объектом любого типа,

хотя и не типизированным способом. В этом состоит их преимущество и в то же время

недостаток. Благодаря тому, что необобщенные коллекции оперируют ссылками на

объекты, в них можно хранить разнотипные данные. Это удобно в тех случаях, когда требуется

манипулировать совокупностью разнотипных объектов или же когда типы хранящихся

в коллекции объектов заранее неизвестны. Но если коллекция предназначается

для хранения объекта конкретного типа, то необобщенные коллекции не обеспечивают

типовую безопасность, которую можно обнаружить в обобщенных коллекциях.

Необобщенные коллекции определены в ряде интерфейсов и классов, реализующих

эти интерфейсы. Все они рассматриваются далее по порядку.

**Интерфейсы необобщенных коллекций**

В пространстве имен System.Collections определен целый ряд интерфейсов

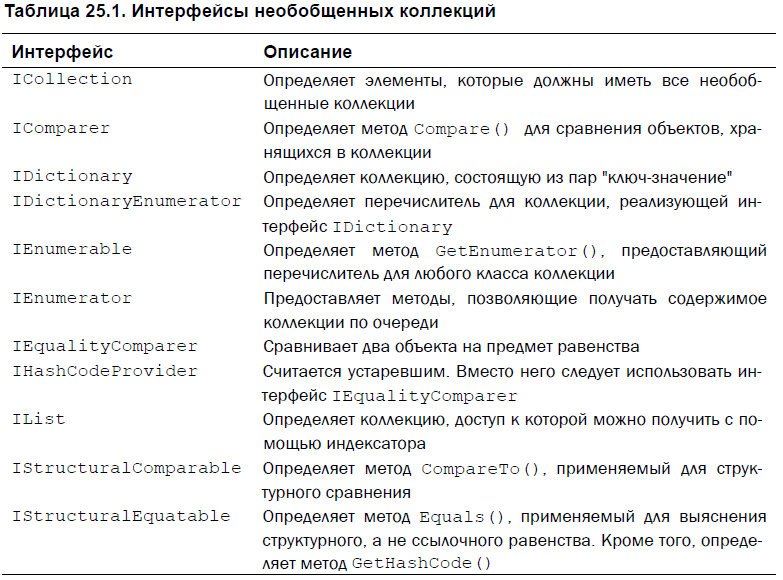
необобщенных коллекций. Начинать рассмотрение необобщенных коллекций следует

именно с интерфейсов, поскольку они определяют функциональные возможности,

которые являются общими для всех классов необобщенных коллекций. Интерфейсы,

служащие опорой для необобщенных коллекций, сведены в табл. 25.1. Каждый из этих

интерфейсов подробно описывается далее.



**Интерфейс Icollection**

Интерфейс ICollection служит основанием, на котором построены все необобщенные

коллекции. В нем объявляются основные методы и свойства для всех необобщенных

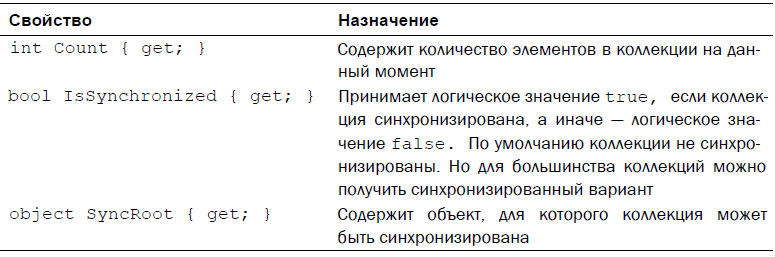
коллекций. Он также наследует от интерфейса IEnumerable.

В интерфейсе ICollection определяются перечисленные ниже свойства.

Свойство Count используется чаще всего, поскольку оно содержит количество элементов,

хранящихся в коллекции на данный момент. Если значение свойства Count

равно нулю, то коллекция считается пустой.



В интерфейсе ICollection определяется следующий метод.

void СоруТо(Array target, int startIdx)

Метод СоруТо() копирует содержимое коллекции в массив *target,* начиная с элемента,

указываемого по индексу *startIdx.* Следовательно, метод СоруТо() обеспечивает

в C# переход от коллекции к стандартному массиву.

Благодаря тому что интерфейс ICollection наследует от интерфейса IEnumerable,

в его состав входит также единственный метод, определенный в интерфейсе

IEnumerable. Это метод GetEnumerator(), объявляемый следующим образом.

IEnumerator GetEnumerator()

Он возвращает перечислитель для коллекции.

Вследствие того же наследования от интерфейса IEnumerable в интерфейсе

ICollection определяются также четыре следующих метода расширения:

AsParallel(), AsQueryable(), Cast() и OfType(). В частности, метод AsParallel()

объявляется в классе System.Linq.ParallelEnumerable, метод AsQueryable() —

в классе System.Linq.Queryable, а методы Cast() и OfType() — в классе System.

Linq.Enumerable. Эти методы предназначены главным образом для поддержки LINQ,

хотя их можно применять и в других целях.

**Интерфейс Ilist**

В интерфейсе IList объявляется такое поведение необобщенной коллекции, которое

позволяет осуществлять доступ к ее элементам по индексу с отсчетом от нуля.

Этот интерфейс наследует от интерфейсов ICollection и IEnumerable. Помимо

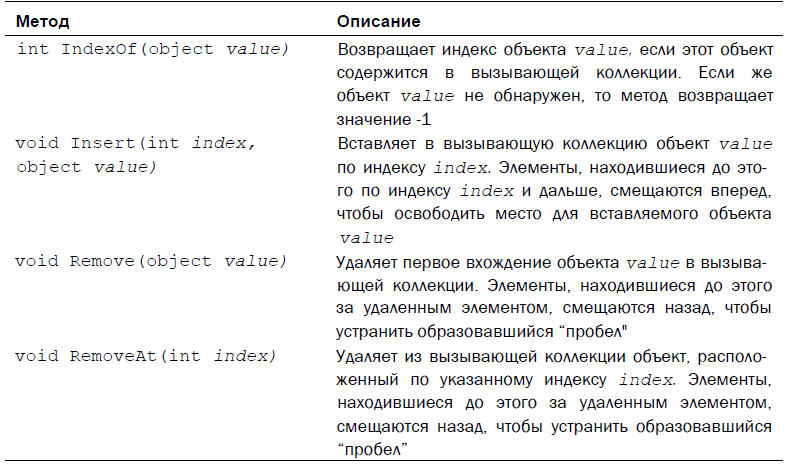
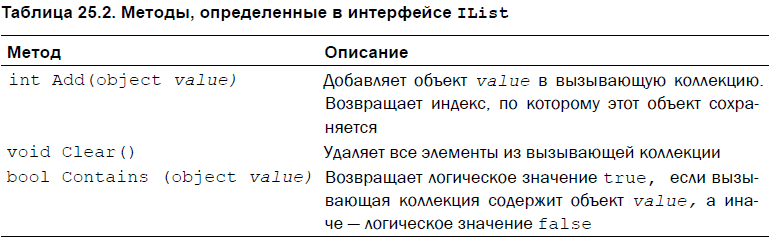
методов, определенных в этих интерфейсах, в интерфейсе IList определяется ряд

собственных методов. Все эти методы сведены в табл. 25.2. В некоторых из них предусматривается

модификация коллекции. Если же коллекция доступна только для чтения

или имеет фиксированный размер, то в этих методах генерируется исключение

NotSupportedException.



Объекты добавляются в коллекцию типа IList вызовом метода Add(). Обратите

внимание на то, что метод Add() принимает аргумент типа object. А поскольку

object является базовым классом для всех типов, то в необобщенной коллекции может

быть сохранен объект любого типа, включая и типы значений, в силу автоматической

упаковки и распаковки.

Для удаления элемента из коллекции служат методы Remove() и RemoveAt(). В частности,

метод Remove() удаляет указанный объект, а метод RemoveAt() удаляет объект

по указанному индексу. И для опорожнения коллекции вызывается метод Clear().

Для того чтобы выяснить, содержится ли в коллекции конкретный объект, вызывается

метод Contains(). Для получения индекса объекта вызывается метод IndexOf(),

а для вставки элемента в коллекцию по указанному индексу — метод Insert().

В интерфейсе IList определяются следующие свойства.

bool IsFixedSize { get; }

bool IsReadOnly { get; }

Если коллекция имеет фиксированный размер, то свойство IsFixedSize содержит

логическое значение true. Это означает, что в такую коллекцию нельзя ни вставлять

элементы, ни удалять их из нее. Если же коллекция доступна только для чтения, то

свойство IsReadOnly содержит логическое значение true. Это означает, что содержимое

такой коллекции не подлежит изменению.

Кроме того, в интерфейсе IList определяется следующий индексатор.

object this[int index] { get; set; }

Этот индексатор служит для получения и установки значения элемента коллекции.

Но его нельзя использовать для добавления в коллекцию нового элемента. С этой целью

обычно вызывается метод Add(). Как только элемент будет добавлен в коллекцию,

он станет доступным посредством индексатора.

**Интерфейс Idictionary**

В интерфейсе IDictionary определяется такое поведение необобщенной коллекции,

которое позволяет преобразовать уникальные ключи в соответствующие значения.

Ключ представляет собой объект, с помощью которого значение извлекается

впоследствии. Следовательно, в коллекции, реализующей интерфейс IDictionary,

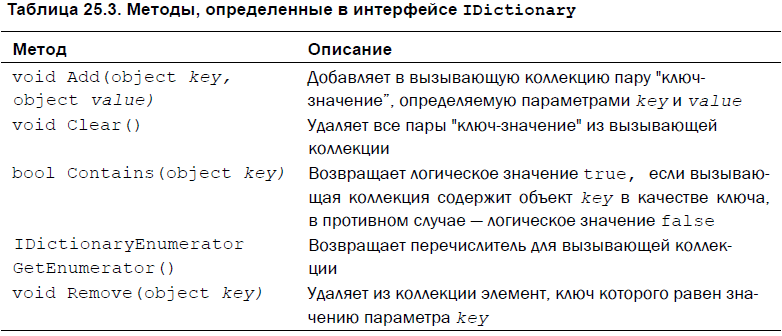
хранятся пары "ключ-значение". Как только подобная пара будет сохранена, ее можно

извлечь с помощью ключа. Интерфейс IDictionary наследует от интерфейсов

ICollection и IEnumerable. Методы, объявленные в интерфейсе IDictionary, сведены

в табл. 25.3. Некоторые из них генерируют исключение ArgumentNullException

при попытке указать пустой ключ, поскольку пустые ключи не допускаются.



Для добавления пары "ключ-значение" в коллекцию типа IDictionary служит

метод Add(). Обратите внимание на то, что ключ и его значение указываются отдельно.

А для удаления элемента из коллекции следует указать ключ этого объекта при

вызове метода Remove(). И для опорожнения коллекции вызывается метод Clear().

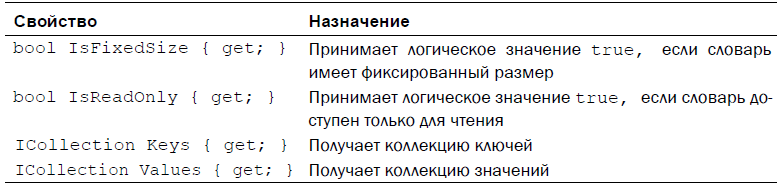
Для того чтобы выяснить, содержит ли коллекция конкретный объект, вызывается

метод Contains() с указанным ключом искомого элемента. С помощью метода

GetEnumerator() получается перечислитель, совместимый с коллекцией типа

IDictionary. Этот перечислитель оперирует парами "ключ-значение".

В интерфейсе IDictionary определяются перечисленные ниже свойства.



Следует иметь в виду, что ключи и значения, содержащиеся в коллекции, доступны

в отдельных списках с помощью свойств Keys и Values.

Кроме того, в интерфейсе IDictionary определяется следующий индексатор.

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

Этот индексатор служит для получения и установки значения элемента коллекции,

а также для добавления в коллекцию нового элемента. Но в качестве индекса в данном

случае служит ключ элемента, а не собственно индекс.

**Интерфейсы IEnumerable, IEnumerator и IdictionaryEnumerator**

Интерфейс IEnumerable является необобщенным, и поэтому он должен быть

реализован в классе для поддержки перечислителей. Как пояснялось выше, интерфейс

IEnumerable реализуется во всех классах необобщенных коллекций, поскольку

он наследуется интерфейсом ICollection. Ниже приведен единственный метод

GetEnumerator(), определяемый в интерфейсе IEnumerable.

IEnumerator GetEnumerator()

Он возвращает коллекцию. Благодаря реализации интерфейса IEnumerable можно

также получать содержимое коллекции в цикле foreach.

В интерфейсе IEnumerator определяются функции перечислителя. С помощью методов

этого интерфейса можно циклически обращаться к содержимому коллекции. Если

в коллекции содержатся пары "ключ-значение" (словари), то метод GetEnumerator()

возвращает объект типа IDictionaryEnumerator, а не типа IEnumerator. Интерфейс

IDictionaryEnumerator наследует от интерфейса IEnumerator и вводит дополнительные

функции, упрощающие перечисление словарей.

В интерфейсе IEnumerator определяются также методы MoveNext() и Reset()

и свойство Current. Способы их применения подробнее описываются далее в этой

главе. А до тех пор следует отметить, что свойство Current содержит элемент, получаемый

в текущий момент. Метод MoveNext() осуществляет переход к следующему

элементу коллекции, а метод Reset() возобновляет перечисление с самого начала.

**Интерфейсы IComparer и IequalityComparer**

В интерфейсе IComparer определяется метод Compare() для сравнения двух

объектов.

int Compare (object х, object у)

Он возвращает положительное значение, если значение объекта х больше, чем у

объекта у; отрицательное — если значение объекта х меньше, чем у объекта у; и нулевое

— если сравниваемые значения равны. Данный интерфейс можно использовать

для указания способа сортировки элементов коллекции.

В интерфейсе IEqualityComparer определяются два метода.

bool Equals(object х, object у)

int GetHashCode(object obj)

Метод Equals() возвращает логическое значение true, если значения объектов *х*

и у равны. А метод GetHashCode() возвращает хеш-код для объекта *obj.*

**Интерфейсы IStructuralComparable и IstructuralEquatable**

Оба интерфейса IStructuralComparable и IStructuralEquatable добавлены в

версию 4.0 среды .NET Framework. В интерфейсе IStructuralComparable определяется

метод CompareTo(), который задает способ структурного сравнения двух объектов

для целей сортировки. (Иными словами, Метод CompareTo() сравнивает содержимое

объектов, а не ссылки на них.) Ниже приведена форма объявления данного метода.

int CompareTo(object other, IComparer comparer)

Он должен возвращать *-1,* если вызывающий объект предшествует другому объекту

*other;* 1, если вызывающий объект следует после объекта *other;* и наконец, 0, если

значения обоих объектов одинаковы для целей сортировки. А само сравнение обеспечивает

объект, передаваемый через параметр *comparer.*

Интерфейс IStructuralEquatable служит для выяснения структурного равенства

путем сравнения содержимого двух объектов. В этом интерфейсе определены следующие

методы.

bool Equals(object other, IEqualityComparer comparer)

int GetHashCode(IEqualityComparer comparer)

Метод Equals() должен возвращать логическое значение true, если вызывающий

объект и другой объект *other* равны. А метод GetHashCode() должен возвращать

хеш-код для вызывающего объекта. Результаты, возвращаемые обоими методами,

должны быть совместимы. Само сравнение обеспечивает объект, передаваемый через

параметр *comparer.*

**Структура DictionaryEntry**

В пространстве имен System.Collections определена структура DictionaryEntry.

Необобщенные коллекции пар "ключ-значение" сохраняют эти пары в объекте типа

DictionaryEntry. В данной структуре определяются два следующих свойства.

public object Key { get; set; }

public object Value { get; set; }

Эти свойства служат для доступа к ключу или значению, связанному с элементом

коллекции. Объект типа DictionaryEntry может быть сконструирован с помощью

конструктора:

public DictionaryEntry(object key, object value)

где key обозначает ключ, a *value* — значение.

**Классы необобщенных коллекций**

А теперь, когда представлены интерфейсы необобщенных коллекций, можно перейти

к рассмотрению стандартных классов, в которых они реализуются. Ниже приведены

классы необобщенных коллекций, за исключением коллекции типа BitArray,

рассматриваемой далее в этой главе.

