Благодаря внедрению обобщений прикладной интерфейс Collections API значительно

расширился, в результате чего количество классов коллекций и интерфейсов

удвоилось. Обобщенные коллекции объявляются в пространстве имен System.

Collections.Generic. Как правило, классы обобщенных коллекций являются не более

чем обобщенными эквивалентами рассматривавшихся ранее классов необобщенных

коллекций, хотя это соответствие не является взаимно однозначным. Например,

в классе обобщенной коллекции LinkedList реализуется двунаправленный список,

тогда как в необобщенном эквиваленте его не существует. В некоторых случаях одни

и те же функции существуют параллельно в классах обобщенных и необобщенных коллекций,

хотя и под разными именами. Так, обобщенный вариант класса ArrayList

называется List, а обобщенный вариант класса HashTable — Dictionary. Кроме

того, конкретное содержимое различных интерфейсов и классов реорганизуется с минимальными

изменениями для переноса некоторых функций из одного интерфейса в

другой. Но в целом, имея ясное представление о необобщенных коллекциях, можно

без особого труда научиться применять и обобщенные коллекции.

Как правило, обобщенные коллекции действуют по тому же принципу, что и необобщенные,

за исключением того, что обобщенные коллекции типизированы. Это

означает, что в обобщенной коллекции можно хранить только те элементы, которые

совместимы по типу с ее аргументом. Так, если требуется коллекция для хранения несвязанных

друг с другом разнотипных данных, то для этой цели следует использовать

классы необобщенных коллекций. А во всех остальных случаях, когда в коллекции

должны храниться объекты только одного типа, выбор рекомендуется останавливать

на классах обобщенных коллекций.

Обобщенные коллекции определяются в ряде интерфейсов и классов, реализующих

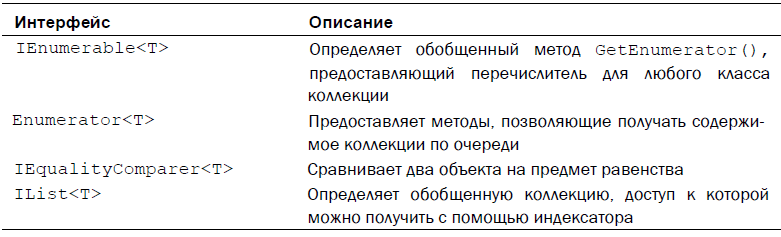
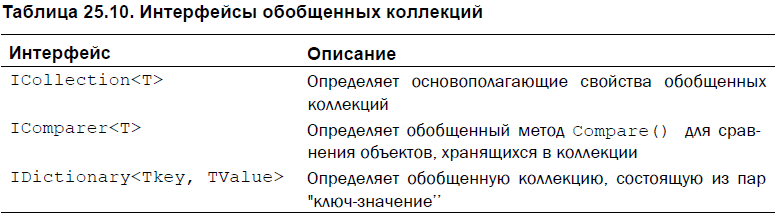
эти интерфейсы.

**Интерфейсы обобщенных коллекций**

В пространстве имен System.Collections.Generic определен целый ряд интерфейсов

обобщенных коллекций, имеющих соответствующие аналоги среди интерфейсов

необобщенных коллекций. Все эти интерфейсы сведены в табл. 25.10.



**Интерфейс ICollection<T>**

В интерфейсе ICollection<T> определен ряд свойств, которые являются общими

для всех обобщенных коллекций. Интерфейс ICollection<T> является обобщенным

вариантом необобщенного интерфейса ICollection, хотя между ними имеются некоторые

отличия.

Итак, в интерфейсе ICollection<T> определены следующие свойства.

int Count { get; }

bool IsReadOnly { get; }

Свойство Count содержит ряд элементов, хранящихся в данный момент в коллекции.

А свойство IsReadOnly имеет логическое значение true, если коллекция доступна

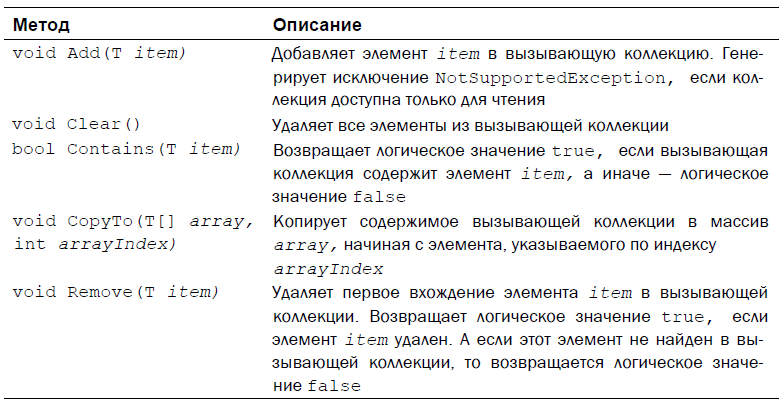
только для чтения. Если же коллекция доступна как для чтения, так и для записи, то

данное свойство имеет логическое значение false.

Кроме того, в интерфейсе ICollection<T> определены перечисленные ниже методы.

Обратите внимание на то, что в этом обобщенном интерфейсе определено несколько

большее количество методов, чем в его необобщенном аналоге.



Некоторые из перечисленных выше методов генерируют исключение

NotSupportedException, если коллекция доступна только для чтения.

А поскольку интерфейс ICollection<T> наследует от интерфейсов IEnumerable и

IEnumerable<T>, то он включает в себя также обобщенную и необобщенную формы

метода GetEnumerator().

Благодаря тому что в интерфейсе ICollection<T> реализуется интерфейс

IEnumerable<T>, в нем поддерживаются также методы расширения, определенные в

классе Enumerable. Несмотря на то что методы расширения предназначены главным

образом для поддержки LINQ, им можно найти и другое применение, в том числе и

в коллекциях.

**Интерфейс IList<T>**

В интерфейсе IList<T> определяется такое поведение обобщенной коллекции,

которое позволяет осуществлять доступ к ее элементам по индексу с отсчетом от

нуля. Этот интерфейс наследует от интерфейсов IEnumerable, IEnumerable<T> и

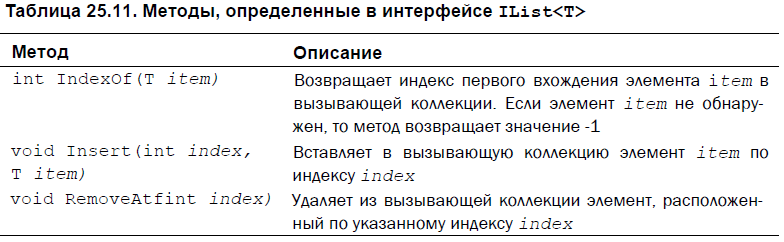
ICollection<T> и поэтому является обобщенным вариантом необобщенного интерфейса

IList. Методы, определенные в интерфейсе IList<T>, перечислены в табл.

25.11. В двух из этих методов предусматривается модификация коллекции. Если же

коллекция доступна только для чтения или имеет фиксированный размер, то методы

Insert() и RemoveAt() генерируют исключение NotSupportedException.



Кроме того, в интерфейсе IList<T> определяется индексатор

Т this[int index] { get; set; }

который устанавливает или возвращает значение элемента коллекции по указанному

индексу *index.*

**Интерфейс IDictionary<TKey, TValue>**

В интерфейсе IDictionary<TKey, TValue> определяется такое поведение обобщенной

коллекции, которое позволяет преобразовать уникальные ключи в соответствующие

значения. Это означает, что в данном интерфейсе определяется коллекция, в

которой хранятся пары "ключ-значение". Интерфейс IDictionary<TKey, TValue>

наследует от интерфейсов IEnumerable, IEnumerable<KeyValuePair<TKey,

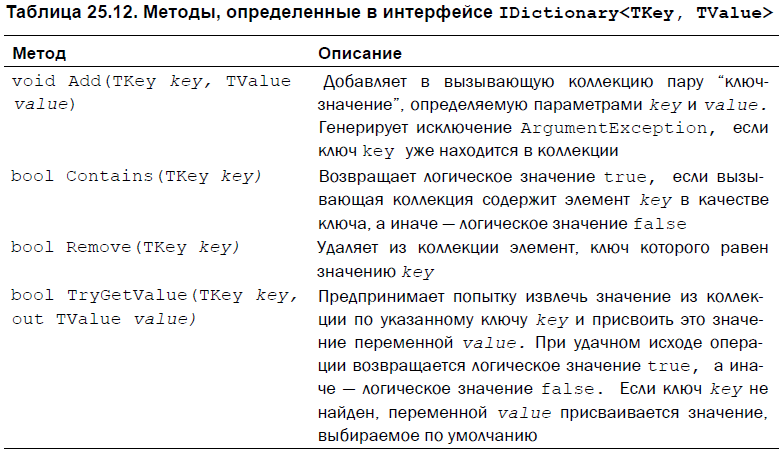
TValue>> и ICollection<KeyValuePair<TKey, TValue>> и поэтому является

обобщенным вариантом необобщенного интерфейса IDictionary. Методы, объявленные

в интерфейсе IDictionary<TKey, TValue>, приведены в табл. 25.12. Все эти

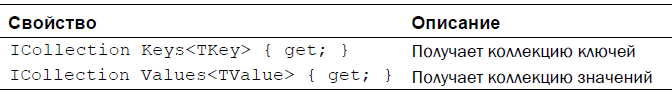
методы генерируют исключение ArgumentNullException при попытке указать пустой

ключ.



Кроме того, в интерфейсе IDictionary<TKey, TValue> определены перечисленные

ниже свойства.



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

отдельными списками с помощью свойств Keys и Values.

И наконец, в интерфейсе IDictionary<TKey, TValue> определяется следующий

индексатор.

TValue this[TKey key] { get; set; }

Этот индексатор служит для получения и установки значения элемента коллекции,

а также для добавления в коллекцию нового элемента. Следует, однако, иметь в виду,

что в качестве индекса в данном случае служит ключ элемента, а не сам индекс.

**Интерфейсы IEnumerable<T> и IEnumerator<T>**

Интерфейсы IEnumerable<T> и IEnumerator<T> являются обобщенными эквивалентами

рассмотренных ранее необобщенных интерфейсов IEnumerable и

IEnumerator. В них объявляются аналогичные методы и свойства, да и действуют они

по тому же принципу. Разумеется, обобщенные интерфейсы оперируют данными

только того типа, который указывается в аргументе типа.

В интерфейсе IEnumerable<T> метод GetEnumerator() объявляется следующим

образом.

IEnumerator<T> GetEnumerator()

Этот метод возвращает перечислитель типа Т для коллекции. А это означает, что

он возвращает типизированный перечислитель.

Кроме того, в интерфейсе IEnumerable<T> определяются два таких же метода, как

и в необобщенном его варианте: MoveNext() и Reset(). В этом интерфейсе объявляется

также обобщенный вариант свойства Current.

Т Current { get; }

Это свойство возвращает ссылку типа Т на следующий объект. А это означает, что

обобщенный вариант свойства Current является типизированным.

Но между интерфейсами IEnumerator и IEnumerator<T> имеется одно важное

различие: интерфейс IEnumerator<T> наследует от интерфейса IDisposable, тогда

как интерфейс IEnumerator не наследует от него. В интерфейсе IDisposable определяется

метод Dispose(), который служит для освобождения неуправляемых ресурсов.

**Интерфейс IComparer<T>**

Интерфейс IComparer<T> является обобщенным вариантом рассмотренного ранее

интерфейса IComparer. Главное отличие между ними заключается в том, что интерфейс

IComparer<T> обеспечивает типовую безопасность. В нем обобщенный вариант

метода Compare() объявляется следующим образом.

int Compare(Т х, Т у)

В этом методе сравниваются объекты х и 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 значение — если сравниваемые значения

равны.

**Интерфейс IEqualityComparer<T>**

Интерфейс IEqualityComparer<T> полностью соответствует своему необобщенному

аналогу EqualityComparer. В нем определяются два следующих метода.

bool Equals(Т х, Т у)

int GetHashCode(Т obj)

Метод Equals() должен возвратить логическое значение true, если значения

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

Если два сравниваемых объекта равны, то их хеш-коды также должны быть одинаковы.

**Интерфейс ISet<T>**

Интерфейс ISet<T> был добавлен в версию 4.0 среды .NET Framework. Он определяет

поведение обобщенной коллекции, реализующей ряд уникальных элементов.

Этот интерфейс наследует от интерфейсов IEnumerable, IEnumerable<T>, а также

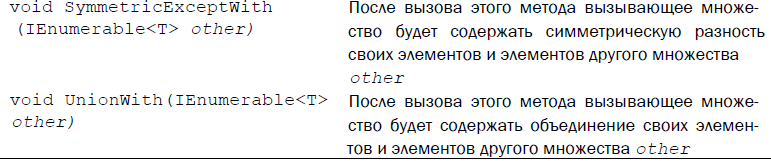
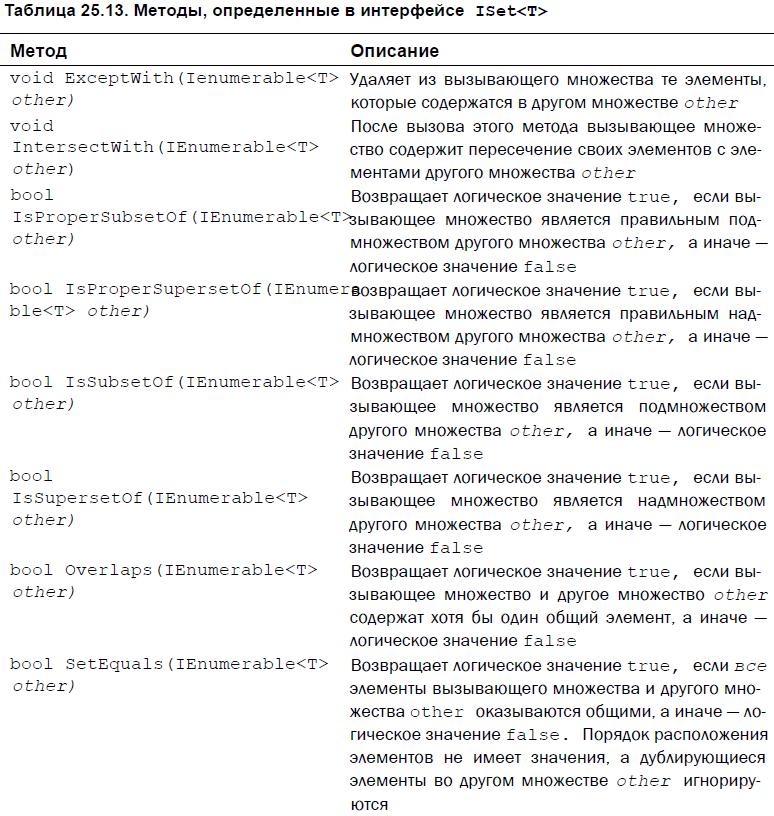
ICollection<T>. В интерфейсе ISet<T> определен ряд методов, перечисленных

в табл. 25.13. Обратите внимание на то, что параметры этих методов указываются как

относящиеся к типу IEnumerable<T>. Это означает, что в качестве второго аргумента

методу можно передать нечто, отличающееся от объектов типа ISet<T> Но чаще

всего оба аргумента оказываются экземплярами объектов типа ISet<T>



**Структура KeyValuePair<TKey, TValue>**

В пространстве имен System.Collections.Generic определена структура

KeyValuePair<TKey, TValue>. Она служит для хранения ключа и его значения

и применяется в классах обобщенных коллекций, в которых хранятся пары "ключ-

значение", как, например, в классе Dictionary<TKey, TValue>. В этой структуре

определяются два следующих свойства.

public TKey Key { get; };

public TValue Value { get; };

В этих свойствах хранятся ключ и значение соответствующего элемента коллекции.

Для построения объекта типа KeyValuePair<TKey, TValue> служит конструктор:

public KeyValuePair(TKey key, TValue value)

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

**Классы обобщенных коллекций**

Как упоминалось ранее, классы обобщенных коллекций по большей части соответствуют

своим необобщенным аналогам, хотя в некоторых случаях они носят другие

имена. Отличаются они также своей организацией и функциональными возможностями.

Классы обобщенных коллекций определяются в пространстве имен System.

Collections.Generic. В табл. 25.14 приведены классы, рассматриваемые в этой главе.

Эти классы составляют основу обобщенных коллекций.

