Операция ToDictionary

Операция ToDictionary создает Dictionary (../../../csharp/charp_theory/level12/12_10.php) типа <K,T>, или, возможно, <K, E>, если прототип имеет аргумент elementSelector, из входной последовательности типа T, где K — тип ключа, а T — тип хранимых значений. Или же, если Dictionary имеет тип <K, E>, то типом хранимых значений будет E, отличающийся от типа элементов в последовательности — T.

Если вы не знакомы с классом Dictionary коллекций C#, то знайте, что он позволяет хранить элементы, которые можно извлекать по ключу. Каждый ключ должен быть уникальным, и только один элемент может быть сохранен для одного ключа. Элементы в Dictionary индексируются по ключу для последующего их извлечения по соответствующим ключам. Для более подробного ознакомления с данным классом проследуйте по приведенной выше ссылке.

Операция ToDictionary имеет четыре прототипа, описанные ниже:

Первый прототип операции ToDictionary

C#

```
public static Dictionary<K, T> ToDictionary<T, K>(
        this IEnumerable<T> source,
        Func<T, K> keySelector);
```

В этом прототипе создается словарь Dictionary типа <K, T>, который возвращается при перечислении входной последовательности по имени source. Делегат метода keySelector вызывается для извлечения значения ключа из каждого входного элемента, и этот ключ становится ключом элемента в Dictionary. Эта версия операции дает в результате элементы в Dictionary того же типа, что и элементы входной последовательности.

Поскольку данный прототип не предусматривает указание объекта IEqualityComparer<K>, эта версия ToDictionary по умолчанию использует объект проверки эквивалентности EqualityComparer<K>.Default.

Второй прототип операции ToDictionary

Второй прототип ToDictionary подобен первому, за исключением того, что он позволяет указывать объект проверки эквивалентности IEqualityComparer<K>. Ниже показан второй прототип:

C#

```
public static Dictionary<K, T> ToDictionary<T, K>(
    this IEnumerable<T> source,
    Func<T, K> keySelector,
    vIEqualityComparer<K> comparer);
```

Этот прототип предоставляет возможность указать объект проверки эквивалентности IEqualityComparer<K>. Данный объект используется для сравнения элементов по значению ключа. Поэтому при добавлении или обращении к элементу в Dictionary он использует этот comparer для сравнения указанного ключа с ключами, содержащимися в Dictionary, чтобы определить их соответствие.

Реализация по умолчанию интерфейса IEqualityComparer<K> предоставляется EqualityComparer.Default. Однако если вы собираетесь использовать класс проверки эквивалентности по умолчанию, то нет причин указывать параметр comparer, потому что предыдущий прототип, где comparer не указан, использует эту установку по умолчанию. Класс StringComparer реализует в себе несколько классов проверки эквивалентности, включая класс, который игнорирует регистр символов. Таким образом, ключи "Joe" и "joe" оцениваются как эквивалентные.

Третий прототип ToDictionary

Третий прототип ToDictionary подобен первому, за исключением того, что позволяет указывать селектор элемента, поэтому тип данных элементов, хранимых в Dictionary, может отличаться от типа элементов входной последовательности:

C#

```
public static Dictionary<K, E> ToDictionary<T, K, E>(
      this IEnumerable<T> source,
      Func<T, K> keySelector,
      Func<T, E> elementSelector);
```

С помощью аргумента elementSelector можно задать делегат метода, возвращающий часть входного элемента или вновь созданный объект совершенно другого типа, — который необходимо сохранить в Dictionary.

Четвертый прототип операции ToDictionary

Четвертый прототип операции ToDictionary предлагает лучшее из всех предыдущих. Он представляет собой комбинацию второго и третьего прототипов, а это означает, что можно указывать объекты elementSelector и comparer.

C#

```
public static Dictionary<K, E> ToDictionary<T, K, E> (
         this IEnumerable<T> source,
         Func<T, K> keySelector,
         Func<T, E> elementSelector,
         IEqualityComparer<K> comparer) ;
```

Этот прототип позволяет указывать объекты elementSelector и comparer.

Если аргумент source, keySelector или elementSelector равен null, либо ключ, возвращенный keySelector, равен null, генерируется *исключение ArgumentNullException*. Если keySelector возвращает одинаковый ключ для двух элементов, генерируется *исключение ArgumentException*.

В следующем примере вместо типичного массива cars, применяемого ранее в примерах, используется общий класс Employee (../level2/2_0.php). Будет создан словарь типа Dictionary<int, Employee>, где ключ типа int — это член id класса Employee, и сам объект Employee представляет хранящийся элемент.

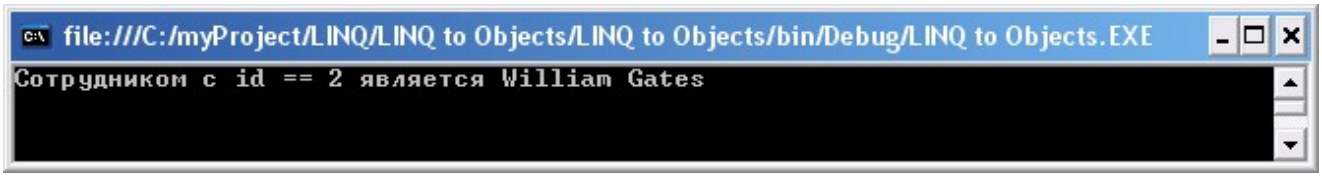
Ниже приведен пример вызова операции ToDictionary с использованием класса Employee:

C#

```
Dictionary<int, Employee> eDictionary =
       Employee.GetEmployeesArray().ToDictionary(k => k.id);

Employee e = eDictionary[2];
Console.WriteLine("Сотрудником с id == 2 является " + e.firstName + " " + e.lastName);
```

Словарь Dictionary объявляется с целочисленным типом ключа, т.к. для ключа выбрано поле Employee.id. Поскольку прототип операции ToDictionary позволяет хранить только входной элемент целиком, которым является объект Employee, типом элемента Dictionary также является Employee. Словарь Dictionary<int, Employee> затем позволяет искать сотрудников по их идентификатору id, предлагая производительность и удобство класса Dictionary. Вот результат работы приведенного выше кода:



Так как предназначение второго прототипа — позволить указывать объект, проверяющий эквивалентность типа IEqualityComparer<T>, для написания примера понадобится ситуация, в которой будет полезен класс, проверяющий эквивалентность.

Это ситуация, когда ключи, которые литерально могут быть не эквивалентны, должны трактоваться таковыми посредством специального класса проверки эквивалентности. Для этой цели в качестве ключа используется числовое значение в строковом формате, такое как "1". Поскольку иногда числовые значения в строковом формате имеют ведущие нули, также может оказаться, что ключ для одного и того же элемента данных имеет вид "1", "01" или даже "00001. Поскольку эти строковые значения не эквивалентны, понадобится класс проверки эквивалентности, который будет знать, что все эти значения следует трактовать как эквивалентные.

Однако сначала понадобится класс с ключом типа string. Для этого в используемый выше общий класс Employee вносятся небольшие изменения. Показанный ниже класс Employee2 во всем идентичен Employee, за исключением того, что типом члена id будет string вместо int:

C#

Пройди тесты

C# тест (легкий) (https://professorweb.ru/test/c-sharp-test.html)

.NET тест (средний) (https://professorweb.ru/test/asp-test.html)

```
public class Employee2
    {
        public string id;
        public string firstName;
        public string lastName;

        public static ArrayList GetEmployeesArrayList()
        {
            ArrayList al = new ArrayList();

            al.Add(new Employee2 { id = "1", firstName = "Joe", lastName = "Rattz" });
            al.Add(new Employee2 { id = "2", firstName = "William", lastName = "Gates" });
            al.Add(new Employee2 {id = "3", firstName = "Anders", lastName = "Hejlsberg"
 });
            al.Add(new Employee2 { id = "4", firstName = "David", lastName = "Lightman" });
            al.Add(new Employee2 { id = "101", firstName = "Kevin", lastName = "Flynn" });
            return (al);
        }

        public static Employee2[] GetEmployeesArray()
        {
            return ((Employee2[])GetEmployeesArrayList().ToArray(typeof(Employee2)));
        }
    }
```

Тип ключа изменен на string, чтобы продемонстрировать, как можно использован класс, проверяющий эквивалентность ключей, даже в случае, когда они литерально не эквивалентны. Поскольку ключи теперь string, в этом примере задействован общий класс MyStringfiedNumberComparer, которому известно, что ключ "02" эквивалентен ключу "2":

C#

```
public class MyStringifiedNumberComparer : IEqualityComparer<string>
  {
    public bool Equals(string x, string y)
    {
      return (Int32.Parse(x) == Int32.Parse(y));
    }

    public int GetHashCode(string obj)
    {
      return Int32.Parse(obj).ToString().GetHashCode();
    }
  }
```

Пройди тесты

Теперь рассмотрим код, в котором применяется класс Employee2 и реализация IEqualityComparer:

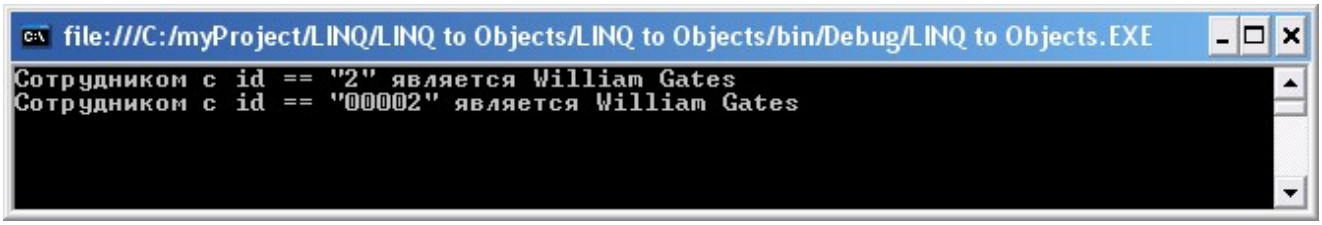
C#

```
Dictionary<int, Employee> eDictionary =
      Employee.GetEmployeesArray().ToDictionary(k => k.id);
Employee e = eDictionary[2];
//Console.WriteLine("Сотрудником с id == 2 является " + e.firstName + " " + e.lastName);

Dictionary<string, Employee2> e2Dictionary =
      Employee2.GetEmployeesArray().ToDictionary(k => k.id, new MyStringifiedNumberComparer
());
Employee2 e2 = e2Dictionary["2"];
Console.WriteLine("Сотрудником с id == \"2\" является " + e2.firstName + " " + e2.lastNam
e);

e2 = e2Dictionary["00002"];
Console.WriteLine("Сотрудником с id == \"00002\" является " + e2.firstName + " " + e2.lastN
ame);
```

В примере производится попытка обратиться к элементам в Dictionary со значениями ключей "2" и "000002". Если класс, проверяющий эквивалентность, работает правильно, в обоих случаях из Dictionary должен быть получен один и тот же элемент. Вот результат:



Как видите, из Dictionary получен один и тот же элемент, независимо от того, какой строковый ключ применялся для доступа. Это справедливо до тех пор, пока каждое переданное строковое значение представляет одно и то же целое число.

Третий прототип позволяет сохранять в словаре элемент, тип которого отличается от типа элементов входной последовательности. Для примера третьего прототипа используется тот же самый класс Employee, который применялся в коде примера первого прототипа ToDictionary. Ниже содержится код примера вызова третьего прототипа ToDictionary:

C#

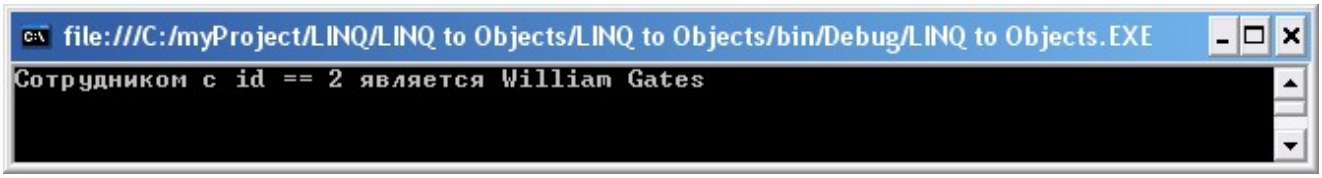
Пройди тесты

C# тест (легкий) (https://professorweb.ru/test/c-sharp-test.html)

.NET тест (средний) (https://professorweb.ru/test/asp-test.html)

```
Dictionary<int, string> eDictionary =
     Employee.GetEmployeesArray().ToDictionary(k => k.id,
                                               i => string.Format("{0} {1}",
                                                                  i.firstName, i.lastName));
string name = eDictionary[2];
Console.WriteLine("Сотрудником с id == 2 является " + name);
```

В данном коде написано лямбда-выражение, соединяющее firstName и lastName в одну строку. Эта объединенная строка становится значением, хранимым в Dictionary. Таким образом, хотя типом входного элемента является Employee, тип данных элемента, хранимого в словаре — string. Вот результаты этого запроса:



Для демонстрации четвертого прототипа ToDictionary применяются классы Employee2 и MyStringifiedNumberComparer. Код примера представлен ниже:

C#

```
Dictionary eDictionary =
                Employee2.GetEmployeesArray().ToDictionary(k => k.id,
                                                           i => string.Format("{0} {1}",
                                                               i.firstName, i.lastName),
                                                           new MyStringifiedNumberComparer
());
string name = eDictionary["2"];
Console.WriteLine("Сотрудником с id == \"2\" является " + name);

name = eDictionary["00002"];
Console.WriteLine("Сотрудником с id == \"00002\" является " + name);
```

В коде создается elementSelector, который указывает единственную строку в качестве значения, сохраняемого в Dictionary, а также пользовательский объект проверки эквивалентности. В результате для извлечения элемента из Dictionary можно использовать "2" или "000002", благодаря специальному классу проверки эквивалентности. То, что теперь получается из Dictionary — это просто строка, в которой содержатся имя и фамилия сотрудника, соединенные вместе. Результат аналогичен примеру со вторым прототипом.

Пройди тесты

C# тест (легкий) (https://professorweb.ru/test/c-sharp-test.html)

.NET тест (средний) (https://professorweb.ru/test/asp-test.html)

Назад (3_1.php)	1	2	3	Вперед (3_3.php)

Пройди тесты

C# тест (легкий) (https://professorweb.ru/test/c-sharp-test.html)

.NET тест (средний) (https://professorweb.ru/test/asp-test.html)