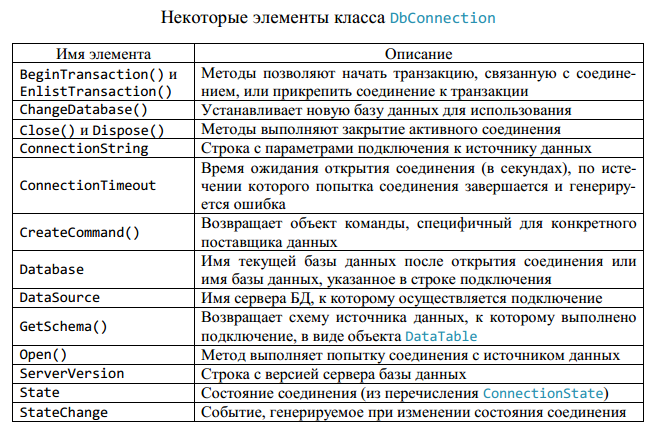
В ADO.NET любое действие с источником данных можно выполнить при наличии соединения с этим источником. Каждый поставщик данных содержит класс, ответственный за описание соединений. Например, поставщик данных для SQL Server использует класс SqlConnection. Класс соединения реализует интерфейс System.Data. IDbConnection. и на-  
следуется от класса System.Data. Common. DbConnection. Наиболее важные элементы класса DbConnection перечислены в табл. 2.1. 

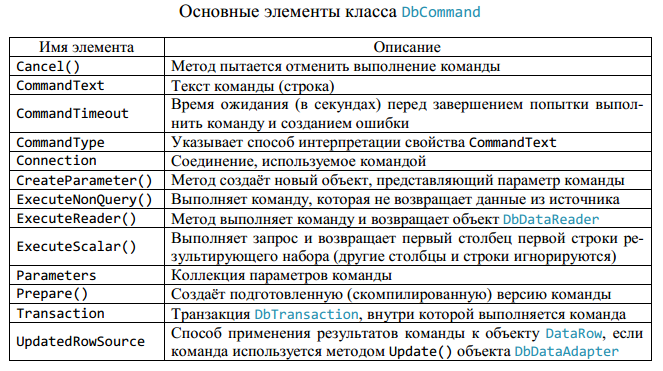
Конкретные поставщики данных могут добавлять или переопределять элементы DbConnection. Класс SqlConnection имеет свойство PacketSize (размер пакета обмена с сервером в байтах), свойство StatisticsEnabled (управление сбором статистики для текущего соединения), свойство WorkstationId (строковый идентификатор подключившегося клиента). Также SqlConnection поддерживает событие InfoMessage, генерируемое, если сервер БД послал клиенту строку с предупреждением или ошибкой.  
Для задания параметров подключения объект соединения использует *строку подключения*, в которой через точку с запятой перечислены пары вида имя параметра=значение. Каждый поставщик данных может использовать специфичные имена параметров.

Для поставщика SQL Server обязательным является задание в строке подключения параметров Data Source и Initial Catalog, а также данных аутентификации. Можно применить два вида аутентификации. Если база использует встроенную аутентификацию, то передаётся имя пользователя и пароль. Сервер SQL Server может применять интегрированную Windows-аутентификацию. Тогда в строке подключения указывается Integrated Security=SSPI. Задать строку подключения можно, либо указав её как параметр конструктора класса соединения, либо при помощи свойства ConnectionString. Естественно, строка подключения задаётся до вызова у соединения метода Open().

Серверы баз данных устанавливают лимит на количество одновременных подключений. Поэтому открытые соединения следует обязательно закрывать после использования. Для этого применяется метод Close() или метод Dispose(), являющийся реализацией интерфейса IDisposable:

// использование using гарантирует вызов connection.Dispose()  
using(var connection = new SqlConnection(builder.ConnectionString))  
{  
connection.Open();  
// работа с соединением  
}  
Для увеличения производительности поставщики данных могут поддерживать *пул соединений* (*connection pool*). Сущность пула заключается в следующем. При вызове методов Close() или Dispose() соединение с базой не разрывается, а помещается в буфер. Если приложение захочет открыть соединение, аналогичное существующему в буфере, то система возвращает открытое  
подключение из пула1. Какие подключения считаются «аналогичными», зависит от поставщика данных. Например, поставщик для SQL Server требует буквального совпадения строк подключения с точностью до символа. Настройка пула соединений выполняется при помощи параметров строки подключения (Pooling, Max\Min Pool Size, Connection Lifetime).

Работа с источником данных основана на выполнении запросов и команд. Для поддержки команд каждый поставщик данных предоставляет класс, который реализует интерфейс IDbCommand и наследуется от класса DbCommand. Основные элементы класса DbCommand приведены в табл. 3.1.



Рассмотрим использование класса команды SqlCommand, описанного в поставщике данных для SQL Server. Класс SqlCommand содержит несколько перегруженных конструкторов. В частности, можно создать команду, указав текст команды и соединение, с которым связана команда:

// конструктор без параметров  
var cmd1 = new SqlCommand();  
// задаём текст команды  
var cmd2 = new SqlCommand("SELECT \* FROM Book");  
// указываем текст команды и соединение  
var cmd3 = new SqlCommand("SELECT \* FROM Book", connection);  
Объект команды также может быть создан при помощи фабрики поставщика или на основе объекта-соединения (в последнем случае команда автоматически ассоциируется с соединением):  
DbCommand cmd4 = SqlClientFactory.Instance.CreateCommand();  
SqlConnection connection = new SqlConnection();  
SqlCommand cmd5 = connection.CreateCommand();

Свойство команды CommandTimeout задаёт время, в течение которого ожидается начало выполнения команды (при использовании SqlCommand значение этого свойства по умолчанию – 30 с). Следует учитывать, что после начала выполнения команды данное свойство никакой роли не играет. Выполнение команды не прервётся, даже если она будет получать данные из базы на протяжении, например, одной минуты. Перед выполнением любая команда должна быть связана с соединением (свойство Connection), и это соединение должно быть открыто. В ADO.NET существует несколько способов выполнения команд, которые отличаются лишь  
информацией, возвращаемой из БД. Ниже перечислены методы выполнения команд, поддерживаемые всеми поставщиками:  
– ExecuteNonQuery(). Метод применяется для запросов, не возвращающих данные. Метод возвращает суммарное число строк, добавленных, изменённых или удалённых в результате выполнения команды;  
– ExecuteScalar(). Метод выполняет команду и возвращает первый столбец первой строки первого результирующего набора данных. Метод может быть полезен при выполнении запросов или хранимых процедур, возвращающих единственный результат;  
– ExecuteReader(). Метод выполняет команду и возвращает *DataReader*

В дополнение к основным методам выполнения команд класс SqlCommand поддерживает метод ExecuteXmlReader(), возвращающий объект XmlReader. Метод работает с SQL Server 2000 или более поздней версией этой СУБД и требует, чтобы получаемые из базы данные имели формат XML. Кроме этого, класс SqlCommand реализует возможность асинхронных операций с базой. У методов ExecuteNonQuery(), ExecuteReader() и ExecuteXmlReader() имеются парные асинхронные аналоги (см. пример кода ниже). Для включения поддержки  
асинхронных операций необходимо задать в строке соединения параметр Asynchronous Processing=true.