В пространстве имён System.IO доступно несколько классов для работы с объектами файловой системы ‑ дисками, каталогами, файлами. Класс DriveInfo инкапсулирует информацию о диске. Он имеет статический метод GetDrives() для получения массива объектов DriveInfo, соответствующих дискам операционной системы. В примере демонстрируется работа с элементами класса DriveInfo.

Классы Directory, File, DirectoryInfo и FileInfo предназначены для работы с каталогами и файлами. Первые два класса выполняют операции при помощи статических методов, вторые два – при помощи экземплярных методов.

Элементы класса FileSystemInfo

|  |  |
| --- | --- |
| **Имя элемента** | **Описание** |
| Attributes | Свойство позволяет получить или установить атрибуты объекта файловой системы (тип – перечисление FileAttributes) |
| CreationTime | Время создания объекта файловой системы |
| Exists | Свойство для чтения, проверка существования объекта файловой системы |
| Extension | Свойство для чтения, расширение файла |
| FullName | Свойство для чтения, полное имя объекта файловой системы |
| LastAccessTime | Время последнего доступа к объекту файловой системы |
| LastWriteTime | Времени последней записи для объекта файловой системы |
| Name | Свойство для чтения; имя файла или каталога |
| Delete() | Метод удаляет объект файловой системы |
| Refresh() | Метод обновляет информацию об объекте файловой системы |

Конструктор класса DirectoryInfo принимает в качестве параметра строку с именем того каталога, с которым будет производиться работа. Для указания текущего каталога используется строка ".". При попытке работать с данными несуществующего каталога генерируется исключение.

Класс FileInfo описывает файл на диске и позволяет производить операции с этим файлом.

Элементы класса FileInfo

|  |  |
| --- | --- |
| **Имя элемента** | **Описание** |
| AppendText() | Создает объект StreamWriter[[1]](#footnote-1) для добавления текста к файлу |
| CopyTo() | Копирует существующий файл в новый файл |
| Create() | Создает файл и возвращает объект FileStream для работы |
| CreateText() | Создает объект StreamWriter для записи текста в новый файл |
| Decrypt() | Дешифрует файл зашифрованный методом Encrypt() |
| Directory | Свойство для чтения, каталог файла |
| DirectoryName | Свойство для чтения, полный путь к файлу |
| Encrypt() | Шифрует файл с учётом данных текущего пользователя |
| IsReadOnly | Булево свойство; является ли файл файлом только для чтения |
| Length | Свойство для чтения, размер файла в байтах |
| MoveTo() | Перемещает файл (возможно, с переименованием) |
| Open() | Открывает файл с указанными правами доступа на чтение, запись или совместное использование |
| OpenRead() | Создает объект FileStream, доступный только для чтения |
| OpenText() | Создает объект StreamReader для чтения информации из существующего текстового файла |
| OpenWrite() | Создает объект FileStream, доступный для чтения и записи |

Как правило, код, работающий с данными файла, вначале вызывает метод Open(). Рассмотрим перегруженную версию метода Open(), которая принимает три параметра. Первый параметр определяет режим запроса на открытие файла. Для него используются значения из перечисления FileMode:

* Append – открывает файл, если он существует, и ищет конец файла. Если файл не существует, то он создается. Этот режим может использоваться только с доступом FileAccess.Write;
* Create – указывает на создание нового файла. Если файл существует, он будет перезаписан;
* CreateNew – указывает на создание нового файла. Если файл существует, генерирует исключение IOException;
* Open – операционная система должна открыть существующий файл;
* OpenOrCreate – операционная система должна открыть существующий файл или создать новый, если файл не существует;
* Truncate – система должна открыть существующий файл и обрезать его до нулевой длины.

Второй параметр метода Open() определяет тип доступа к данным файла. Для него используются элементы перечисления FileAccess:

* + Read – файл будет открыт только для чтения;
  + ReadWrite – файл будет открыт и для чтения, и для записи;
  + Write – файл открывается только для записи, то есть добавления данных.

Третий параметр задаёт возможность совместной работы с открытым файлом и представлен значениями перечисления FileShare:

* None – совместное использование запрещено, на любой запрос на открытие файла будет возвращено сообщение об ошибке;
* Read – файл могут открыть и другие пользователи, но только для чтения;
* ReadWrite – другие пользователи могут открыть файл и для чтения, и для записи;
* Write – файл может быть открыт другими пользователями для записи.

Кроме методов класса FileInfo, статический класс File обладает методами, позволяющими легко прочитать и записать информацию, содержащуюся в файле определенного типа:

* File.ReadAllText() – читает содержимое текстового файла как строку;
* File.ReadAllLines() – читает текстовый файл как массив строк;
* File.ReadAllBytes() – возвращает содержимое файла как массив байт;
* File.WriteAllText();
* File.WriteAllLines();
* File.WriteAllBytes();
* File.AppendAllText() – добавляет строку к текстовому файлу.

Начиная с платформы .NET версии 2.0, доступен статический класс Path для работы с именами файлов и путями в файловой системе. Методы этого класса позволяют выделить имя файла из полного пути, скомбинировать для получения пути имя файла и имя каталога. Также класс Path обладает методами, генерирующими имя для временного файла или каталога.

Платформа .NET содержит развитый набор типов для поддержки операций ввода/вывода информации. Типы для поддержки ввода/вывода можно разбить на две категории: *типы для представления потоков данных* и *адаптеры потоков*. *Поток данных* – это абстрактное представление данных в виде последовательности байт. Поток либо ассоциируется с неким физическим хранилищем (файлами на диске, памятью, сетью), либо декорирует (обрамляет) другой поток, преобразуя данные тем или иным образом. Адаптеры потоков служат оболочкой потока, преобразуя информацию определённого формата в набор байт[[2]](#footnote-2).

**1. Абстрактный класс System.IO.Stream** ‑ базовый класс для других классов, представляющих потоки.

**2. Классы для работы с потоками, связанными с хранилищами.**

* + FileStream – класс для работы с файлами, как с потоками (пространство имён System.IO).
  + MemoryStream – класс для представления потока в памяти (пространство имён System.IO).
  + NetworkStream – работа с сокетами, как с потокам (пространство имён System.Net.Sockets).
  + PipeStream ‑ абстрактный класс из пространства имён System.IO.Pipes, базовый для классов-потоков, которые позволяют передавать данные между процессами системы.

**3. Декораторы потоков.**

* + DeflateStream и GZipStream – классы (пространство имён System.IO.Compression) для потоков со сжатием данных.
  + CryptoStream – поток зашифрованных данных (пространство имён System.Security.Cryptography).
  + BufferedStream – поток с поддержкой буферизации данных (пространство имён System.IO).

**4. Адаптеры потоков.**

* + BinaryReader и BinaryWriter – классы для ввода/вывода примитивных типов в двоичном формате.
  + StreamReader и StreamWriter – классы для ввода/вывода информации в строковом представлении.
  + XmlReader и XmlWriter – абстрактные классы для ввода/вывода XML.

Элементы абстрактного класса Stream

|  |  |
| --- | --- |
| **Категория** | **Элементы** |
| Чтение данных | bool CanRead { get; } |
| int Read(byte[] buffer, int offset, int count) |
| int ReadByte() |
| Запись данных | bool CanWrite { get; } |
| void Write(byte[] buffer, int offset, int count) |
| void WriteByte(byte value) |
| Перемещение | bool CanSeek { get; } |
| long Position { get; set; } |
| void SetLength(long value) |
| long Length { get; } |
| long Seek(long offset, SeekOrigin origin) |
| Закрытие потока | void Close() |
| void Dispose() |
| void Flush() |
| Таймауты | bool CanTimeout { get; } |
| int ReadTimeout { get; set; } |
| int WriteTimeout { get; set; } |
| Другие элементы | static readonly Stream Null |
| static Stream Synchronized(Stream stream) |

Класс Stream вводит поддержку асинхронного ввода/вывода. Для этого служат методы BeginRead() и BeginWrite(). Уведомление о завершении асинхронной операции возможно двумя способами: или при помощи делегата типа AsyncCallback, передаваемого как параметр методов BeginRead() и BeginWrite(), или при помощи вызова методов EndRead() и EndWrite(), которые ожидают до окончания асинхронной операции.

Статический метод Synchronized() возвращает оболочку для потока, которая обеспечивает безопасность при совместной работе с потоком нескольких нитей выполнения.

Класс MemoryStream даёт возможность организовать поток в оперативной памяти. Свойство Capacity этого класса позволяет получить или установить количество байтов, выделенных под поток. Метод ToArray() записывает все содержимое потока в массив байт. Метод WriteTo() переносит содержимое потока из памяти в другой поток, производный от класса Stream.

Класс BufferedStream – это декоратор потока для повышения производительности путём буферизации данных. В примере кода BufferedStream работает с FileStream, предоставляя 20.000 байт буфера. То есть, второе физическое обращение к файлу произойдет только при чтении 20.001-го байта.

Классы DeflateStream и GZipStream являются декораторами потока, реализующими по алгоритму, аналогичному формату ZIP. Они различаются тем, что GZipStream записывает дополнительные данные о протоколе сжатия в начало и конец потока.

Перейдём к рассмотрению классов-адаптеров для потоков. Классы BinaryReader и BinaryWriter позволяют при помощи своих методов читать и записывать в поток данные примитивных типов и массивов байт или символов. Вся информация записывается в поток в двоичном представлении.

Абстрактные классы TextReader и TextWriter дают возможность читать и записывать данные в поток в строковом представлении. От этих классов наследуются классы StreamReader и StreamWriter.

1. [↑](#footnote-ref-1)
2. Сами адаптеры потоками не являются. [↑](#footnote-ref-2)