Основы программирования

Средства ввода-вывода

Схема потоков данных

Схема потоков

Передача данных (ввод/вывод) осуществляется по определенной схеме – схеме потоков.

Данные -- абстрактный линейный поток, передаваемого от источника к приемнику (получателю).

При этом:

- источник не знает, кто является получателем данных, и работает только с потоком;
- получатель также не имеет информации об источнике, и работает только с потоком данных.

Схема потоков

Абстракция потока позволяет сделать универсальной:

- передачу данных независимо от типа получателя;
- прием данных, независимо от источника.

При этом и источник и получатель работают с абстракцией потока.

Концепция потока не ограничена файловым вводом-выводом -- библиотеки .NET предоставляют потоковый доступ к сетям, областям памяти и прочим абстракциям.

Типы потоков

Потоки делятся на потоки:

- ввода (потоки, с которыми работает получатель) и
- вывода (потоки, с которыми работает источник).

Таким образом, ввод и вывод логически отделены друг от друга и (по большому счету) никак не связаны.

По типу передаваемых данных потоки бывают:

- байтовыми
- текстовыми.

Типы потоков

Выделим четыре большие группы потоков:

- байтовые ввода;
- байтовые вывода;
- текстовые ввода (потоки чтения);
- текстовые вывода (потоки записи).

Каждая группа имеет базовый абстрактный класс, где описана основная функциональность.

Наследники этих классов реализуют ввод или вывод для конкретных хранилищ данных.

Окончив работу с потоком его нужно закрыть.

Байтовые потоки данных

Объединены байтовый поток ввода и байтовый поток вывода.

Работа ведется не с потоком ввода и с потоком вывода по отдельности, а с байтовым потоком ввода/вывода.

Поток поддерживает произвольный доступ к своему содержимому. В связи с этим возникает необходимость управления курсором в потоке.

Базовая функциональность описана в абстрактном классе `Stream`.

Класс Stream

Базовый для всех остальных классов потоков. Из пространства имен `System.IO`.

Является абстрактным классом, а это означает, что создать экземпляр класса `Stream` нельзя.

В абстрактном классе определен набор членов, которые обеспечивают поддержку:

- синхронного и
- асинхронного

взаимодействия с хранилищем (например, файлом или областью памяти).

Класс Stream

- Потомки класса `Stream` представляют данные, как низкоуровневые потоки байт, а непосредственная работа с низкоуровневыми потоками может оказаться довольно неоднозначной.
- Некоторые типы, унаследованные от `Stream`, поддерживают поиск (`seeking`), что означает возможность получения и изменения текущей позиции в потоке.

Класс Stream. Свойства

`CanRead` - определяет, поддерживает ли текущий поток чтение.

`CanSeek` - определяет, поддерживает ли текущий поток поиск.

`CanWrite` - определяет, поддерживает ли текущий поток запись.

`Length` - возвращает длину потока в байтах.

`Position` - определяет текущую позицию в потоке.

Класс Stream. Методы

`Close()` – закрывает текущий поток и освобождает все ресурсы, связанные с текущим потоком.

`Flush()` – записывает данные в связанный с потоком источник данных и очищает внутренний буфер потока. Если буфер не реализуется, то метод не делает ничего.

`Read()` – читает последовательность байт в массив и перемещает курсор на количество считанных байтов.

`ReadByte()` – читает одиночный байт из текущего потока и перемещает текущую позицию потока на один байт.

Класс Stream. Методы

`Seek()` – устанавливает позицию в текущем потоке.

`SetLength()` – устанавливает длину текущего потока.

`Write()` – записывает последовательность байтов в поток и переместить курсор на количество записанных байтов.

`WriteByte()` – записывает байт в поток и перемещает курсор на один байт.

Класс FileStream

Является наследником класса `Stream` и реализует всю его функциональность для взаимодействия с файлами.

Файл при этом интерпретируется как линейный поток байтов, каждый байт которого может быть прочитан или записан.

При этом реализуется возможность управления курсором в файле.

Для задания режимов работы с файлами используются стандартные перечисления `FileMode`, `FileAccess` и `FileShare`.

Класс FileStream

Класс `FileStream` также позволяет создать новый файл на диске, или открыть существующий.

Применяется для чтения и записи данных в любой файл.

Взаимодействовать с членами типа `FileStream` придется нечасто, так как чаще используются оболочки потоков, которые облегчают работу с текстовыми данными или типами `.NET`.

Создание экземпляра FileStream

Для создания экземпляра требуется указать:

- ***файл***, к которому должен получаться доступ.
- ***режим открытия файла*** - создать или открыть существующий? Должно ли перезаписываться его содержимое, или новые данные должны добавляться в конец файла?
- ***вид доступа к файлу*** - нужен ли доступ на чтение или запись либо то и другое вместе?
- ***общий доступ*** - должен ли доступ к файлу быть эксклюзивным или должна быть возможность доступа со стороны других потоков одновременно. Разрешено ли им чтение, запись либо то и другое?

Перечисления FileStream

FileMode: Append,
Create,
CreateNew,
Open,
OpenOrCreate,
Truncate

FileAccess: Read,
ReadWrite,
Write

FileShare: Delete,
Inheritable,
None,
Read,
ReadWrite,
Write

Задание режима FileMode

Если запрашивается режим, не соответствующий существующему состоянию файла, может быть сгенерировано исключение.

Значения `Append`, `Open` и `Truncate` будут приводить к генерации исключения, если файл не существует, а значение `CreateNew` — наоборот, если он уже существует.

Значения `Create` и `OpenOrCreate` подходят в обоих сценариях, но `Create` приводит к удалению любого существующего файла и замене его новым, изначально пустым.

FileAccess и FileShare

Перечисления `FileAccess` и `FileShare` являются битовыми флагами, поэтому их значения могут комбинироваться с помощью битовой операции "`ИЛИ`", то есть "|".

Конструкторы FileStream

Создает файл с доступом для чтения и записи и позволяет другим потокам получать к нему доступ для чтения

```
FileStream fs = new FileStream(@"C:\C#
Projects\Project.doc", FileMode.Create);
```

+ другие потоки имеют доступ к файлу для записи

```
FileStream fs2 = new FileStream(@"C:\C#
Projects\Project2.doc", FileMode.Create,
FileAccess.Write);
```

+ другие потоки не имеют доступа к файлу до тех пор, пока файл `fs3` открыт

```
FileStream fs3 = new FileStream(@"C:\C#
Projects\Project3.doc", FileMode.Create,
FileAccess.Write, FileShare.None);
```

Конструкторы FileStream

Перегруженные версии конструкторов способны подставлять стандартные значения `FileAccess.ReadWrite` и `FileShare.Read` на месте третьего и четвертого параметров в зависимости от значения `FileMode`.

Закрытие потока

После окончания работы поток нужно закрыть:

```
fs.Close();
```

Закрытие потока приводит к освобождению всех связанных с ним ресурсов и позволяет другим приложениям запускать потоки для работы с тем же файлом.

Кроме того, это действие приводит к очистке буфера.

Методы ReadByte()

Представляет собой самый простой способ для чтения данных.

Он берет один байт из потока и приводит результат к типу `int` со значением в диапазоне от `0` до `255`. В случае достижения конца потока он возвращает `-1`.

Метод Read()

Метод `Read()` - считывает заданное количество байт из файла в массив.

Принимает три параметра и возвращает количество успешно считанных байтов.

Принимаемые параметры:

- `array` - массив байтов, куда будут помещены считываемые из файла данные;
- `offset` представляет смещение в байтах в массиве `array`, в который считанные байты будут помещены;
- `count` - максимальное число байтов, предназначенных для чтения.

Метод Read()

Метод `Read()` возвращает действительное количество прочитанных байтов; если возвращается значение `0`, значит, был достигнут конец потока.

Пример чтения данных в массив байтов `ByteArray`:

```
int nBytesRead = fs.Read(ByteArray,
                                0, nBytes);
```

`0` -- смещения, которое позволяет указать, что массив должен заполняться, начиная не с первого, а с какого-то другого элемента;

`nBytes` -- сколько байт должно читаться в массив.

Метод WriteByte()

Позволяет записывать по одному байту в поток:

```
byte NextByte = 4;
fs.WriteByte(NextByte);
```

Метод Write()

Метод `Write()` записывает в файл данные из массива байтов.

Принимает три параметра:

- `array` - массив байтов, откуда данные будут записываться в файла;
- `offset` - смещение в байтах в массиве `array`, откуда начинается запись байтов в поток;
- `count` - максимальное число байтов, предназначенных для записи.

Пример

```
FileStream fs = new FileStream(”test.dat”,
FileMode.OpenOrCreate, FileAccess.ReadWrite,
FileShare.None);
byte[] x = new byte[10];
for (int i = 0; i < 10, i++) {
     x[i] = i;                    }
fs.Write(x);
     //в поток по-байтно записывается массив байт
fs.Write(x,0,5);
     //пишем 5  элементов массива, начиная с 0-го
```

Пример

```
for (byte i = 0; i < 256; i++) {
fs.WriteByte(i);                }
//записываются числа от 0 до 255 в виде байтов
fs.Position = 0;
//перемещаем курсор в начало файлового потока
byte[] y = new byte[15];
fs.Read(y,0,15);
//из потока считывается 15 элементов в массив
byte z = fs.ReadByte();
//из потока считывается 1 байт
fs.Close();
```

Метод Seek()

Метод `Seek()` используется для установки курсора в потоке на байт с определенным номером (еще один способ управления свойством `Position`).

Этот метод имеет два параметра – количество байт, которые нужно отступить и точку отсчета, от которой нужно отступить указанное число байт.

Точкой отсчета являются константы из перечисления `SeekOrigin`:

- `Begin` – от начала файлового потока:
- `Current` – от текущей позиции курсора в потоке;
- `End` – от конца потока.

Исключения

При работе с файлами возможен выброс следующих исключений:

- `FileNotFoundException` – попытка открыть несуществующий файл;
- `DirectoryNotFoundException` – обращение к несуществующей директории;
- `ArgumetException` – неверно задан режим открытия потока;
- `IOException` – ошибка ввода/вывода.

Эти исключения нужно отлавливать.

Рекомендуемые источники

1.C# и .NET | FileStream. Чтение и запись файла (metanit.com)
https://metanit.com/sharp/tutorial/5.4.php