Лекция 15 Потоковые адаптеры и анонимные каналы

15.1 Понятие потокового адаптера

Базовым классом всех потоков является класс Stream, который оперирует только с байтами данных.

Однако, существует множество задач, в которых предпочтительнее является работа не с байтами данных, а с более сложными типами, например, строками или XML-документами.

Поэтому в языке C# были разработаны специальные классы, включающие специализированные методы для работы со сложными типами данных. Эти классы не являются потоками, а предоставляют потокам специализированные методы, выполняющие роль преобразователей типов данных в потоках, как бы расширяют возможности потоков.

Потоковые адаптеры это специальные классы, предназначенные для работы с потоками и содержащие специальные методы для работы со сложными типами данных.

В настоящее время платформа .NET, точнее ее библиотека Framework, содержит три группы потоковых адаптеров:

- текстовые адаптеры для строковых и символьных типов данных;
- двоичные адаптеры для работы со стандартными базовыми типами, такими как bool, byte, char, int, float, double и т.д.
- XML-адаптеры, позволяют значительно упростить работу потоков с XML-документами.

В этой лекции мы познакомимся с текстовыми и двоичными потоковыми адаптерами.

15.2 Текстовые и двоичные потоковые адаптеры

Текстовые потоковые адаптеры платформы .NET базируются на абстрактных классах TextReader и TextWriter.

Класс TextReader, в свою очередь, является абстрактным базовым классом для классов StreamReader и StringReader, которые обеспечивают чтение символов либо из потока, либо из обычной строки.

Класс TextWriter, так же является абстрактным базовым классом для классов StreamWriter и StringWriter, которые обеспечивают запись символов либо в поток, либо в обычную строку.

Таким образом, эти классы обеспечивают передачу данных в символьном виде как из потока в строку, так из строки в поток.

Каждый из перечисленных классов имеет богатый набор методов, свойств и полей, в дополнение к которым обычно около 10 перегружаемых конструкторов. Копия экрана справки для StreamReader приведена в конце лекции.

В предыдущей лекции для чтения текстовых сообщений из именованного канала использован потоковый адаптер `StreamReader reader`

`= new StreamReader(pipestream);`, который позволил читать данные из потока в строковом виде `ss = reader.ReadLine();` и сразу отображать их в консольном окне экрана монитора `Console.WriteLine("Получено сообщение от сервера: " + ss);`.

Для записи текстовых сообщений в именованный канал передачи данных, в предыдущей лекции использован потоковый адаптер `StreamWriter writer = new StreamWriter(pipestream);`, который позволил записывать данные из строковой переменной в поток. Например,

```
Console.WriteLine("Сообщение от клиента ?");
ss = Console.ReadLine();
writer.WriteLine(ss);
writer.Flush();
```

Метод writer.Flush(); очищает содержимое буфера обмена.

Двоичные потоковые адаптеры представлены классами BinaryReader и BinaryWriter, которые читают и, соответственно, записывают данные базовых типов в поток. Из первого семестра вы знаете, что к базовым типам относятся все значимые типы языка C#, а это bool, byte, int, float, double и т.д.

В отличие от текстовых адаптеров двоичные адаптеры хранят данные базовых типов в том виде, в каком они представлены в памяти компьютера – под данные целого типа отводится 4 байта, а под данные типа double – 8 байт.

15.3 Понятие анонимного канала передачи данных

Как мы уже отмечали, анонимный канал предназначен для передачи данных между процессами, работающими на одном компьютере.

Классы, обеспечивающие работу анонимного канала, имеют названия AnonymousPipeServerStream и AnonymousPipeClientStream.

Анонимный канал может иметь только однонаправленный поток – только чтение или только запись. Поэтому для обмена данными между процессами необходимо создавать два анонимных канала.

Скорость работы анонимного канала значительно быстрее, чем у именованного канала, так как он не выходит за границы операционной системы компьютера (говорят, что он не работает с транспортными протоколами сети).

При создании сервером анонимного канала операционная система присваивает каналу «закрытый дескриптор», идентификатор которого необходимо передавать клиенту. Обычно это осуществляется с помощью аргументов метода Main программы клиента. Получив идентификатор дескриптора анонимного канала сервера, клиент может создать свой анонимный канал для передачи данных, но с противоположным направлением – если сервер настраивается на чтение данных, то клиент должен записывать данные в канал и наоборот.

В общем случае взаимодействия сервера и клиента должны осуществляться в следующей очередности:

- сервер создает анонимный канал передачи данных, определяя направление передачи (In или Out);
- с помощью метода GetClientHandleAsString() сервер получает идентификатор созданного анонимного канала;
- сервер запускает процесс клиента и передает ему в виде аргумента метода Main полученный идентификатор;
- клиент создает экземпляр анонимного канала для полученного идентификатора, но с противоположным направлением передачи данных;
- сервер освобождает «закрытый дескриптор» полученный при создании анонимного канала;
- сервер и клиент обмениваются данными.

Рассмотрим работу анонимного канала передачи данных.

15.4 Анонимный канал передачи вещественных данных

Рассмотрим чисто учебный пример передачи вещественного чисел от сервера клиенту и целого числа от клиента серверу.

Исходный код сервера:

```
using System;
using System.Diagnostics;
using System.IO;
using System.IO.Pipes;

namespace ConsoleApplication1
{
 class Program
 {
  static void Main(string[] args)
  {
   string clientExe = "ConsoleApplication2.exe";
   AnonymousPipeServerStream PipeServer1 = new
AnonymousPipeServerStream(PipeDirection.Out,
HandleInheritability.Inheritable);
   AnonymousPipeServerStream PipeServer2 = new
AnonymousPipeServerStream(PipeDirection.In,
HandleInheritability.Inheritable);
   string txID = PipeServer1.GetClientHandleAsString();
   string rxID = PipeServer2.GetClientHandleAsString();
   var startInfo = new ProcessStartInfo(clientExe, txID + " " +
rxID);
   startInfo.UseShellExecute = false;//в одно консольное окно
   Process p = Process.Start(startInfo);
   PipeServer1.DisposeLocalCopyOfClientHandle();
   PipeServer2.DisposeLocalCopyOfClientHandle();
   BinaryWriter dd = new BinaryWriter(PipeServer1);
   dd.Write(15.5);
   BinaryReader ii = new BinaryReader(PipeServer2);
```

```
   Console.WriteLine("сервер получил сообщение:  " +
ii.ReadInt32());
   p.WaitForExit();
   Console.WriteLine("сервер - сеанс закончен");
   Console.ReadLine();
  }
 }
}
```

Исходный код клиента:

```
using System;
using System.IO;
using System.IO.Pipes;

namespace ConsoleApplication2
{
 class Program
 {
  static void Main(string[] args)
  {
   string rxID = args[0];
   string txID = args[1];
   using (var rx = new
AnonymousPipeClientStream(PipeDirection.In, rxID))
   using (var tx = new
AnonymousPipeClientStream(PipeDirection.Out, txID))
   {
    BinaryReader dd = new BinaryReader(rx);
    Console.WriteLine(" клиент получил сообщение: " +
dd.ReadDouble());
    BinaryWriter ii = new BinaryWriter(tx);
    ii.Write(1234);
   }
   Console.WriteLine("клиент - сеанс закончен");
   Console.ReadLine();
  }
 }
}
```

Для нормальной работы программ файл `ConsoleApplication2.exe`, должен находиться в папке с файлом `ConsoleApplication1.exe`. Рассматривая работу программы необходимо отметить, что были созданы два анонимных канала сервером `PipeServer1` и `PipeServer2`, для которых были получены идентификаторы каналов txID и rxID. Эти идентификаторы были переданы программе клиент (при ее запуске).

В программе сервер были использованы два двоичных потоковых адаптера – для передачи вещественного числа и для приема целого числа.

Программа клиент с помощью аргументов метода Main приняла идентификаторы созданных каналов и сформировала два анонимных канала rx и tx для чтения и записи данных в канал. Одновременно программа клиент использовала два двоичных потоковых адаптера – для приема вещественного числа и для отправки целого числа серверу.

Работа программ отображена на рисунке 15.1.

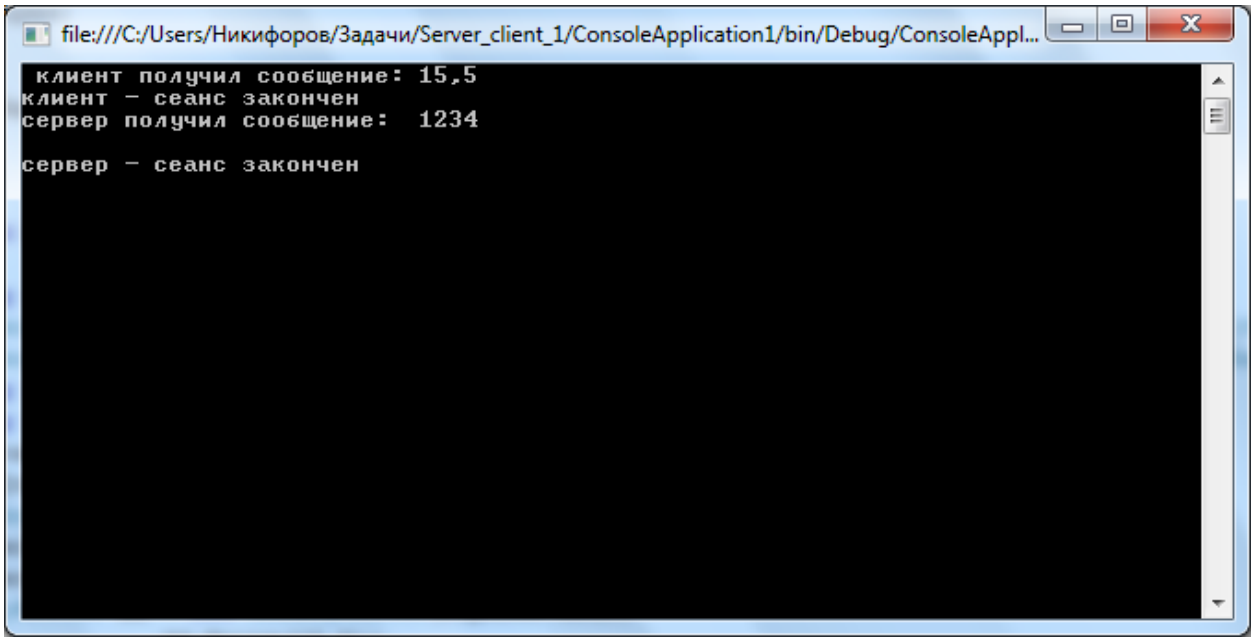


Рисунок 15.1 – Работа анонимного канала

15.4 Анонимный канал передачи текстовых данных

В чисто учебных целях рассмотрим работу анонимного канала передачи текстовых данных от сервера к клиенту и от клиента к серверу. Естественно и сервер и клиент должны создавать по два анонимных канала передачи данных и дважды использовать текстовые потоковые адаптеры. Перед запуском программ, но после их компиляции файл `ConsoleApplication2.exe` необходимо разместить в корневом каталоге диска с.

Исходный код программы сервера:

```
using System;
using System.Diagnostics;
using System.IO;
using System.IO.Pipes;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;

namespace ConsoleApplication1
{
 class Program
 {
  static void Main(string[] args)
  {
   string clientExe = "ConsoleApplication2.exe";
```

```
   AnonymousPipeServerStream PipeServer1 = new
AnonymousPipeServerStream(PipeDirection.Out,
HandleInheritability.Inheritable);
   AnonymousPipeServerStream PipeServer2 = new
AnonymousPipeServerStream(PipeDirection.In,
HandleInheritability.Inheritable);
   string txID = PipeServer1.GetClientHandleAsString();
   string rxID = PipeServer2.GetClientHandleAsString();
   var startInfo = new ProcessStartInfo(clientExe, txID + " " +
rxID);
   startInfo.UseShellExecute = false;
   Process p = Process.Start(startInfo);
   StreamReader reader = new StreamReader(PipeServer2);
   StreamWriter writer = new StreamWriter(PipeServer1);
   PipeServer1.DisposeLocalCopyOfClientHandle();
   PipeServer2.DisposeLocalCopyOfClientHandle();
   Console.WriteLine("Ваше сообщение ?");
   string ss = Console.ReadLine();
   writer.WriteLine(ss);
   writer.Flush();
   Console.WriteLine("сервер получил сообщение:  " +
reader.ReadLine());
   p.WaitForExit();
   Console.WriteLine("сервер - сеанс закончен");
   Console.ReadLine();
  }
 }
}
```

Исходный код программы клиента:

```
using System;
using System.Diagnostics;
using System.IO;
using System.IO.Pipes;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication2
{
 class Program
 {
  static void Main(string[] args)
  {
   string rxID = args[0];
   string txID = args[1];
   using (var rx = new
AnonymousPipeClientStream(PipeDirection.In, rxID))
   using (var tx = new
AnonymousPipeClientStream(PipeDirection.Out, txID))
   {
    StreamReader reader = new StreamReader(rx);
    StreamWriter writer = new StreamWriter(tx);
```

```
    Console.WriteLine(" клиент получил сообщение: " +
reader.ReadLine());
    Console.WriteLine("Ваше сообщение ?");
    string ss = Console.ReadLine();
    writer.WriteLine(ss);
    writer.Flush();
   }
   Console.WriteLine("клиент - сеанс закончен");
   Console.ReadLine();
  }
 }
}
```

Для нормальной работы программ файл `ConsoleApplication2.exe`, должен находиться в папке с файлом `ConsoleApplication1.exe`. Работа программы представлена копией экрана на рисунке 15.2

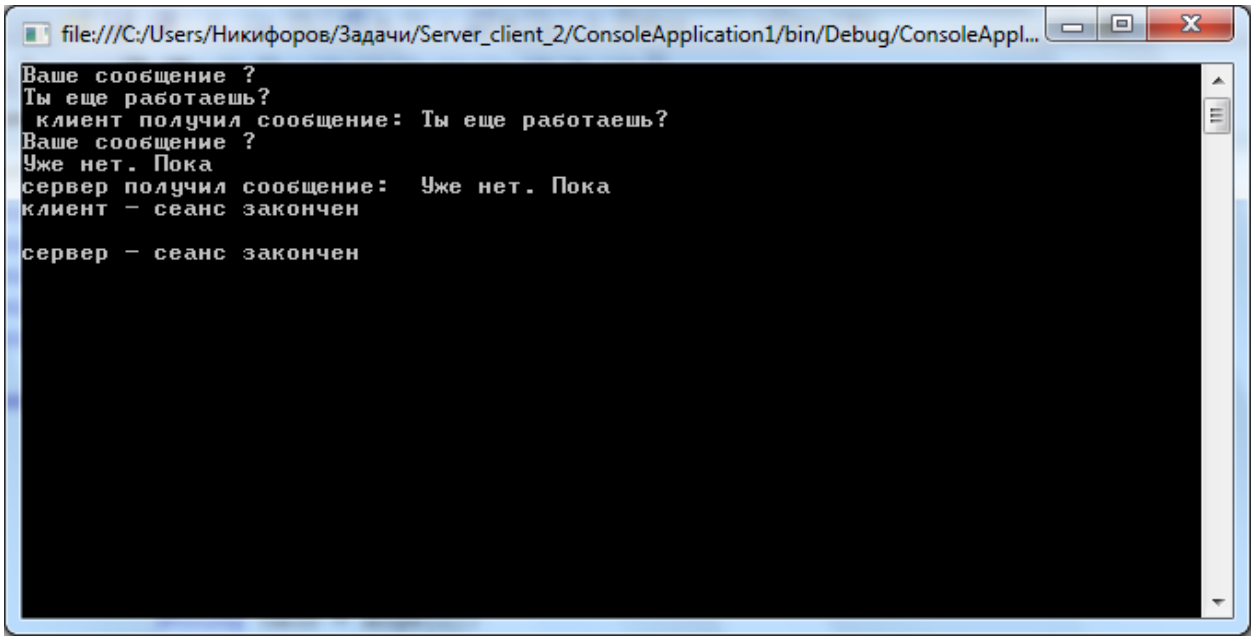


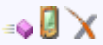
Рисунок 15.2 – Работа анонимного канала при передаче текстовых данных

Приложение к лекции 15

Реализует TextReader, который считывает символы из потока байтов в определенной кодировке.

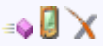
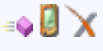
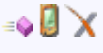
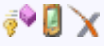
Универсальный тип StreamReader предоставляет следующий члены.

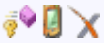
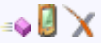
Конструкторы

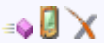
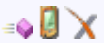
	Имя	Описание
	StreamReader	Перегружен. Инициализирует новый экземпляр класса StreamReader для указанного потока.

В начало страницы

Методы

	Имя	Описание
	Close	Закрывает объект StreamReader и основной поток и освобождает все системные ресурсы, связанные с устройством чтения. (Переопределяет TextReader.Close().)
	CreateObjRef	Создает объект, который содержит всю необходимую информацию для создания прокси-сервера, используемого для взаимодействия с удаленным объектом. (Унаследовано от MarshalByRefObject.)
	DiscardBufferedData	Позволяет объекту StreamReader удалять текущие данные.
	Dispose	Перегружен.
	Equals	Определяет, равен ли заданный объект Object текущему объекту Object. (Унаследовано от Object.)
	Finalize	Позволяет объекту Object попытаться освободить ресурсы и

		выполнить другие операции очистки, перед тем как объект Object будет утилизирован в процессе сборки мусора. (Унаследовано от Object.)
	GetHashCode	Играет роль хэш-функции для определенного типа. (Унаследовано от Object.)
	GetLifetimeService	Извлекает объект обслуживания во время существования, который управляет политикой времени существования данного экземпляра. (Унаследовано от MarshalByRefObject.)
	GetType	Возвращает объект Type для текущего экземпляра. (Унаследовано от Object.)
	InitializeLifetimeService	Возвращает объект обслуживания во время существования для управления политикой времени существования данного экземпляра. (Унаследовано от MarshalByRefObject.)
	MemberwiseClone	Перегружен.
	Peek	Возвращает следующий доступный символ, но не использует его. (Переопределяет TextReader.Peek().)
	Read	Перегружен. Считывает следующий символ или следующий набор символов из входного потока.
	ReadBlock	Выполняет чтение максимального количества символов *count* из текущего потока и записывает данные в буфер *buffer*, начиная с *index*. (Унаследовано от

		TextReader.)
	ReadLine	Выполняет чтение строки символов из текущего потока и возвращает данные в виде строки. (Переопределяет TextReader.ReadLine().)
	ReadToEnd	Считывает поток от текущего положения до конца. (Переопределяет TextReader.ReadToEnd().)
	ToString	Возвращает объект String, который представляет текущий объект Object. (Унаследовано от Object.)

В начало страницы

Поля

	Имя	Описание
	Null	Объект StreamReader для пустого потока.

В начало страницы

Свойства

	Имя	Описание
	BaseStream	Возвращает основной поток.
	CurrentEncoding	Получает текущую кодировку символов, используемую текущим объектом StreamReader.
	EndOfStream	Получает значение, определяющее, находится ли позиция текущего потока в конце потока.

В начало страницы

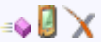
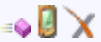
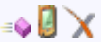
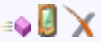
Явные реализации интерфейса

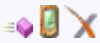
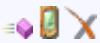
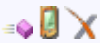
	Имя	Описание
	IDisposable.Dispose	Описание этого члена см. в описании свойства IDisposable.Dispose.

		(Унаследовано от TextReader.)

В начало страницы

Список перегрузки

	Имя	Описание
	StreamReader(Stream)	Инициализирует новый экземпляр класса StreamReader для указанного потока.
	StreamReader(String)	Инициализирует новый экземпляр класса StreamReader для указанного имени файла.
	StreamReader(Stream, Boolean)	Инициализирует новый экземпляр класса StreamReader для указанного потока с заданным параметром обнаружения метки порядка следования байтов.
	StreamReader(Stream, Encoding)	Инициализирует новый экземпляр класса StreamReader для указанного потока с заданной кодировкой символов.
	StreamReader(String, Boolean)	Инициализирует новый экземпляр класса StreamReader для указанного имени файла с заданным параметром обнаружения метки порядка следования байтов.
	StreamReader(String, Encoding)	Инициализирует новый экземпляр класса StreamReader для указанного имени файла с заданной кодировкой символов.
	StreamReader(Stream, Encoding, Boolean)	Инициализирует новый экземпляр класса StreamReader для указанного потока с заданной кодировкой символов и параметром обнаружения метки порядка следования

		байтов.
	StreamReader(String, Encoding, Boolean)	Инициализирует новый экземпляр класса StreamReader для указанного имени файла с заданной кодировкой символов и параметром обнаружения метки порядка следования байтов.
	StreamReader(Stream, Encoding, Boolean, Int32)	Инициализирует новый экземпляр класса StreamReader для указанного потока с заданной кодировкой символов, параметром обнаружения метки порядка следования байтов и размером буфера.
	StreamReader(String, Encoding, Boolean, Int32)	Инициализирует новый экземпляр класса StreamReader для указанного имени файла с заданной кодировкой символов, параметром обнаружения метки порядка следования байтов и размером буфера.