Работа с файловой системой

В пространстве имен System.IO предусмотрено четыре класса, которые предназначены для работы с файловой системой компьютера, т.е для создания, удаления переноса и т.д. файлов и каталогов.

Первые два типа — Directory и Fi1e реализуют свои возможности с помощью статических методов, поэтому данные классы можно использовать без создания соответствующих объектов (экземпляров классов).

Следующие типы – DirectoryInfo и FileInfo обладают схожими функциональными возможностями с Directory и Fi1e, но порождены от класса FileSystemInfo и поэтому реализуются путем создания соответствующих экземпляров классов.

Работа с каталогами

Абстрактный класс FileSystemInfo

Значительная часть членов FileSystemInfo предназначена для работы с общими характеристиками файла или каталога (метками времени, атрибутами и т. п.). Рассмотрим некоторые свойства FileSystemInfo:

Свойство	***Описание***
Attributes	Позволяет получить или установить атрибуты для данного объекта файловой системы. Для этого свойства используются значения и перечисления FileAttributes
CreationTime	Позволяет получить или установить время создания объекта файловой системы
Exists	Может быть использовано для того, чтобы определить, существует ли данный объект файловой системы
Extension	Позволяет получить расширение для файла
FullName	Возвращает имя файла или каталога с указанием пути к нему в файловой системе
LastAccessTime	Позволяет получить или установить время последнего обращения к объекту файловой системы
LastWriteTime	Позволяет получить или установить время последнего внесения изменений в объект файловой системы
Name	Возвращает имя указанного файла. Это свойство доступно только для чтения. Для каталогов возвращает имя последнего каталога в иерархии, если это возможно. Если нет, возвращает полностью определенное имя

В FileSystemInfo предусмотрено и несколько методов. Например, метод Delete() - позволяет удалить объект файловой системы с жесткого диска, а Refresh() — обновить информацию об объекте файловой системы.

Класс DirectoryInfo

Данный класс наследует члены класса FileSystemInfo и содержит дополнительный набор членов, которые предназначены для создания, перемещения, удаления, получения информации о каталогах и подкаталогах в файловой системе. Наиболее важные члены класса содержатся в следующей таблице:

Член	Описание
Create() CreateSubDirectory()	Создают каталог (или подкаталог) по указанному пути в файловой системе
Delete()	Удаляет пустой каталог
GetDirectories()	Позволяет получить доступ к подкаталогам текущего каталога (в виде массива объектов DirectoryInfo)
GetFiles()	Позволяет получить доступ к файлам текущего каталога (в виде массива объектов FileInfo)
MoveTo()	Перемещает каталог и все его содержимое на новый адрес в файловой системе
Parent	Возвращает родительский каталог в иерархии файловой системы

Работа с типом DirectoryInfo начинается с того, что мы создаем экземпляр класса (объект), указывая при вызове конструктора в качестве параметра путь к нужному каталогу. Если мы хотим обратиться к текущему каталогу (то есть каталогу, в котором в настоящее время производится выполнение приложения), вместо параметра используется обозначение ".". Например:

```
// Создаем объект DirectoryInfo, которому будет обращаться к текущему каталогу
DirectoryInfo dir1 = new DirectoryInfo(".");
// Создаем объект DirectoryInfo, которому будет обращаться к каталогу d:\prim
DirectoryInfo dir2 = new DirectoryInfo(@"d:\prim");
```

Если мы попытаемся создать объект DirectoryInfo, связав его с несуществующим каталогом, то будет сгенерировано исключение System.IO.DirectoryNotFoundException. Если же все нормально, то мы сможем получить доступ к данному каталогу. В примере, который приведен ниже, мы создаем объект DlrectoryInfo, который связан с каталогом d:\prim, и выводим информацию о данном каталоге:

```
using System;
using System.Text;
using System.IO;

namespace MyProgram
{
  class Program
  {
    static void Main(string[] args)
    {
        DirectoryInfo dir = new DirectoryInfo(@"d:\prim");
        Console.WriteLine("***** "+dir.Name+" *****");
        Console.WriteLine("FullName: {0}", dir.FullName);
        Console.WriteLine("Name: {0}", dir.Name);
        Console.WriteLine("Parent: {0}", dir.Parent);
        Console.WriteLine("Creation: {0}", dir.CreationTime);
        Console.WriteLine("Attributes: {0}", dir.Attributes.ToString());
        Console.WriteLine("Root: {0}", dir.Root);
      }
  }
}
```

Свойство Attributes позволяет получить информацию об атрибутах объекта файловой системы. Возможные значения данного свойства приведены в следующей таблице:

Значение	**Описание**
Archive	Этот атрибут используется приложениями при проведении резервного копирования, а в некоторых случаях — удаления старых файлов
Compressed	Определяет, что файл является сжатым
Directory	Определяет, что объект файловой системы является каталогом
Encrypted	Определяет, что файл является зашифрованным
Hidden	Определяет, что файл является скрытым (такой файл не будет выводиться при обычном просмотре каталога)
Normal	Определяет, что файл находится в обычном состоянии и для него установлены любые другие атрибуты. Этот атрибут не может использоваться с другими атрибутами
Offline	Файл (расположенный на сервере) кэширован в хранилище off-line на клиентском компьютере. Возможно, что данные этого файла уже устарели

Readonly	Файл доступен только для чтения
System	Файл является системным (то есть файл является частью операционной системы или используется исключительно операционной системой)

Через *DirectoryInfo* можно не только получать доступ к информации о текущем каталоге, но получить доступ к информации о его подкаталогах:

```
class Program
{
  static void printDirect( DirectoryInfo dir)
  {
    Console.WriteLine("***** "+dir.Name+" *****");
    Console.WriteLine("FullName: {0}", dir.FullName);
    Console.WriteLine("Name: {0}", dir.Name);
    Console.WriteLine("Parent: {0}", dir.Parent);
    Console.WriteLine("Creation: {0}", dir.CreationTime);
    Console.WriteLine("Attributes: {0}", dir.Attributes.ToString());
    Console.WriteLine("Root: {0}", dir.Root);
  }
  static void Main(string[] args)
  {
    DirectoryInfo dir = new DirectoryInfo(@"d:\prim");
    printDirect(dir);
    DirectoryInfo[] subDirects = dir.GetDirectories();
    Console.WriteLine("Найдено {0} подкаталогов", subDirects.Length);
    foreach (DirectoryInfo d in subDirects)
    {
        printDirect(d);
    }
   }
}
```

Задание. Преобразуйте метод printDirect в рекурсивный метод таким образом, чтобы можно было просмотреть информацию о всех подкаталогах текущего каталога, независимо от уровня вложенности.

Метод CreateSubdirectory() позволяет создать в выбранном каталоге как единственный подкаталог, так и множество подкаталогов (в том числе, и вложенных друг в друга). Создадим в каталоге несколько дополнительных подкаталогов:

```
DirectoryInfo dir = new DirectoryInfo(@"d:\prim");
dir.CreateSubdirectory("doc"); //создали подкаталог
dir.CreateSubdirectory(@"book\2008"); //создали вложенный подкаталог
```

Задание. Создайте вложенный подкаталог best\2008 для своего каталога

Метод MoveTo() позволяет переместить текущий каталог по заданному в качестве параметра адресу. При этом возможно произвести переименование каталога. Например:

```
DirectoryInfo dir = new DirectoryInfo(@"d:\prim\bmp");
dir.(@"d:\prim\letter\bmp");
```

В данном случае каталог bmp перемещается в по адресу d:\prim\letter\bmp. Так как имя перемещаемого каталога совпадает с крайним правым именем в адресе нового местоположения

каталога, то переименования не происходит. Следующий пример позволит нам переименовать текущий каталог:

```
DirectoryInfo dir = new DirectoryInfo(@"d:\prim\letter");
dir.MoveTo(@"d:\prim\archive");
```

Класс Directory

Работать с каталогами файловой системы компьютера можно и при помощи класса Directory, функциональные возможности которого во многом совпадают с возможностями DirectoryInfo. Но члены данного класса реализованы статически, поэтому для их использования нет необходимости создавать объект.

Рассмотрим работу с методами данного класса на примерах.

Замечание. Удалите с диска d измененную папку prim. И еще раз скопируйте ее исходную версию из раздела 12 данного электронного учебника.

```
Directory.CreateDirectory(@"d:\prim\2008");//создали подкаталог 2008
Directory.Move(@"d:\prim\bmp", @"d:\prim\2008\bmp");//перенесли каталог bmp в каталог 2008
Directory.Move(@"d:\prim\letter", @"d:\prim\archives");//переименовали каталог letter в archives
```

Замечания.

1. Удаление каталога возможно только тогда, когда он пуст.
2. На практике комбинируют использование классов Directory и DirectoryInfo.

Работа с файлами

Класс FileInfo

Класс FileInfo предназначен для организации доступа к физическому файлу, который содержится на жестком диске компьютера. Он позволяет получать информацию об этом файле (например, о времени его создания, размере, атрибутах и т. п.), а также производить различные операции, например, по созданию файла или его удалению. Класс FileInfo наследует члены класса FileSystemInfo и содержит дополнительный набор членов, который приведен в следующей таблице:

Член	***Описание***
AppendText()	Создает объект StreamWriter для добавления текста к файлу
CopyTo()	Копирует уже существующий файл в новый файл
Create()	Создает новый файл и возвращает объект FileStream для взаимодействия с этим файлом
CreateText()	Создает объект StreamWriter для записи текстовых данных в новый файл
Delete()	Удаляет файл, которому соответствует объект FileInfo
Directory	Возвращает каталог, в котором расположен данный файл
DirectoryName	Возвращает полный путь к данному файлу в файловой системе
Length	Возвращает размер файла
MoveTo()	Перемещает файл в указанное пользователем место (этот метод позволяет одновременно переименовать данный файл)
Name	Позволяет получить имя файла
Open()	Открывает файл с указанными пользователем правами доступа на чтение, запись или совместное использование с другими пользователями
OpenRead()	Создает объект FileStream, доступный только для чтения
OpenText()	Создает объект StreamReader (о нем также будет рассказано ниже), который позволяет считывать информацию из существующего текстового файла
OpenWrite()	Создает объект FileStream, доступный для чтения и записи

Как мы видим, большинство методов FileInfo возвращает объекты (FIleStream, StreamWriter, StreamReader и т. п.), которые позволяют различным образом взаимодействовать с

файлом, например, производить чтение или запись в него. Приемы работы с данными потоками нам уже известны. Поэтому рассмотрим другие возможности класса FileInfo.

```
using System;
using System.Text;
using System.IO;       //для работы с файловым вводом-выводом
using System.Text.RegularExpressions;

namespace MyProgram
{
  class Program
  {
  static void Main()
  {
      //создаем новый файл и связываем с ним строковый поток
      FileInfo f = new FileInfo("text.txt");
      StreamWriter fOut = new StreamWriter(f.Create());
      //записываем в файл данные и закрываем строковый поток,
                        // при этом связь с физическим файлом для f не рвется
      fOut.WriteLine("ОДИН ДВА ТРИ...");
      fOut.Close();
      //получаем информацию о файле
      Console.WriteLine("*******"+f.Name File Inf+"**************");
      Console.WriteLine("File size: {0}", f.Length);
      Console.WriteLine("Creation: {0}", f.CreationTime);
      Console.WriteLine("Attributes: {0}", f.Attributes.ToString());
    }
  }
}
```

Рассмотрим следующий пример:

```
    static void Main()
    {
        FileInfo f = new FileInfo(@"d:\prim\letter\letter1.txt");
        f.CopyTo(@"d:\prim\bmp\letter.txt");
        Directory.CreateDirectory(@"d:\prim\archives");
        f.MoveTo(@"d:\prim\archives\letter1.txt");
        f = new FileInfo(@"d:\prim\letter\letter2.txt");
        f.Delete();
    }
```

Рассмотрим еще один пример по удалению файлов:

```
static void printFile( FileInfo file)
{
  Console.WriteLine("***** "+file.Name+" *****");
  Console.WriteLine("File size: {0}", file.Length);
  Console.WriteLine("Creation: {0}", file.CreationTime);
  Console.WriteLine("Attributes: {0}", file.Attributes.ToString());
}
```

```
static void Main(string[] args)
{
  DirectoryInfo dir = new DirectoryInfo(@"d:\prim\bmp");
  FileInfo[] files = dir.GetFiles();
  if (files.Length!=0)
    {
      Console.WriteLine("Найдено {0} файла", files.Length);
      foreach (FileInfo f in files)
      {
        printFile(f);
        f.Delete();
      }
      Console.WriteLine("\nТеперь в каталоге содержится {0} файлов и можно его удалить",
      dir.GetFiles().Length);
      dir.Delete();
    }
}
```

Класс File

Доступ к физическим файлам можно получать и через статические методы класса File. Большинство методов объекта Fileinfo представляют в этом смысле зеркальное отражение методов объекта File.

```
static void Main(string[] args)
{
    File.Copy(@"d:\prim\letter\letter1.txt",@"d:\prim\bmp\letter1.txt");
    Directory.CreateDirectory(@"d:\prim\archives");
    File.Move(@"d:\prim\letter\letter1.txt",@"d:\prim\archives\letter1.txt");
    File.Delete(@"d:\prim\letter\letter2.txt");
    Directory.Delete(@"d:\prim\letter");
}
```

Имеет прямой смысл использовать статический класс File, когда требуется осуществить единственный вызов метода на объект. В этом случае вызов будет выполнен быстрее, поскольку .NET Framework не придется проходить через процедуру создания экземпляра нового объекта с последующим вызовом метода. Однако если приложение осуществляет несколько операций над файлом, то более разумным представляется создать экземпляр объекта FileInfo и использовать его методы. Это позволит сэкономить определенное время, поскольку объект будет заранее настроен на нужный файл в файловой системе, в то время как статическому классу придется каждый раз осуществлять его поиск заново.

Аналогичное правило действует и при выборе между классами Directory и DirectoryInfo.