ПОТОКОВЕ ВВЕДЕННЯ-ВИВЕДЕННЯ

Введення-виведення в програмах на C # здійснюється за допомогою потоків. Потік - це якась абстракція виробництва або споживання інформації. З фізичним пристроєм потік пов'язує система введення-виведення. Всі потоки діють однаково - навіть якщо вони пов'язані з різними фізичними пристроями. Тому класи і методи введення-виведення можуть застосовуватися до самих різних типів пристроїв. Наприклад, методами виведення на консоль можна користуватися і для виведення в файл на диску.

На найнижчому рівні введення-виведення в C # здійснюється байтами. І робиться це тому, що багато пристроїв орієнтовані на операції введення-виведення окремими байтами. Але людині більше властиво спілкуватися символами. Нагадаємо, що в C # тип char є 16-розрядним, а тип byte - 8-розрядних. Так, якщо в цілях введення-виведення використовується набір символів в коді ASCII, то для перетворення типу char в тип byte досить відкинути старший байт значення типу char. Але це не годиться для набору символів в УНІКОД (Unicode), де символи потрібно представляти двома, а то й більше байтами. Отже, байтові потоки не зовсім підходять для організації введення-виведення окремими символами. З метою вирішити це складне становище в середовищі .NET Framework визначено кілька класів, виконують перетворення байтового потоку в символьний з автоматичним перетворенням типу byte в тип char і назад.

Класи бібліотеки .Net для роботи з потоками

клас FileStream

У середовищі .NET Framework передбачені класи для організації введення-виведення в файли. Безумовно, це в основному файли дискового типу. На рівні операційної системи файли мають байтовую організацію. І, як слід було очікувати, для введення і виведення байтів в файли є відповідні методи. Тому введення і виведення в файли байтовими потоками досить поширений. Крім того, байтовий потік введення

або виведення в файл може бути укладений в відповідний об'єкт символьного потоку. Операції символьного введення-виведення в файл знаходять застосування при обробці тексту. Про символьних потоках мова піде далі в цій главі, а тут розглядається байтовий введення-виведення.

Для створення байтового потоку, прив'язаного до файлу, служить клас FileStream. Цей клас є похідним від класу Stream і успадковує все його функції. Нагадаємо, що класи потоків, в тому числі і FileStream, визначені в просторі імен System.IO. Тому на самому початку будь-якої використовує їх програми зазвичай вводиться наступний рядок коду.

Клас FileStream представляє можливості з зчитування з файлу і запису в файл. Він дозволяє працювати як з текстовими файлами, так і з бінарними.

Розглянемо найбільш важливі його властивості та методи:

• Властивість Length: повертає довжину потоку в байтах

• Властивість Position: повертає поточну позицію в потоці

• Метод Read: зчитує дані з файлу в масив байтів. Приймає три параметра: int Read (byte [] array, int offset, int count) і повертає кількість успішно лічених байтів. Тут використовуються такі параметри:

o array - масив байтів, куди будуть поміщені зчитувальні з файлу дані

o offset представляє зміщення в байтах в масиві array, в який лічені байти будуть поміщені

o count - максимальне число байтів, призначених для читання. Якщо у файлі знаходиться менша кількість байтів, то всі вони будуть прочитані.

• Метод long Seek (long offset, SeekOrigin origin): встановлює позицію в потоці зі зміщенням на кількість байт, зазначених в параметрі offset.

• Метод Write: записує в файл дані з масиву байтів. Приймає три параметра: Write (byte [] array, int offset, int count)

o array - масив байтів, звідки дані будуть записуватися в файлу

o offset - зміщення в байтах в масиві array, звідки починається запис байтів в потік

o count - максимальне число байтів, призначених для запису

FileStream представляє доступ до файлів на рівні байтів, тому, наприклад, якщо вам треба вважати або записати одну або кілька рядків в текстовий файл, то масив байтів треба перетворити в рядки, використовуючи спеціальні методи. Тому для роботи з текстовими файлами застосовуються інші класи.

У той же час при роботі з різними бінарними файлами, що мають певну структуру FileStream може бути дуже навіть корисний для вилучення певних порцій інформації та її обробки.

Відкриття та закриття файлу

Для формування байтового потоку, прив'язаного до файлу, створюється об'єкт класу FileStream. В цьому класі межах кілька конструкторів. Нижче наведено чи не найпоширеніший серед них:

FileStream (string шлях, FileMode режим)

де шлях позначає ім'я файлу, включаючи повний шлях до нього;

а режим-порядок відкриття файлу. В останньому випадку вказується одне зі значень, що визначаються в перерахуванні FileMode.

Як правило, цей конструктор відкриває файл для доступу з метою читання або запису. Винятком з цього правила є відкриття файлу в режимі FileMode.Append, коли файл стає доступним тільки для запису.

Якщо спроба відкрити файл виявляється невдалою, то генерується ісключеніе.Значенія з перерахування FileMode Як згадувалося вище, конструктор класу FileStream відкриває файл, доступний для читання або запису. Якщо ж потрібно обмежити доступ до файлу тільки для читання або ж тільки для запису, то в такому випадку слід використовувати такий конструктор.FileStream (string шлях, FileMode режим, FileAccess доступ) Як і раніше, шлях позначає ім'я файлу, включаючи і повний шлях до нього, а режим - порядок відкриття файлу. У той же час доступ позначає конкретнийспособ доступу до файлу. В останньому випадку вказується одне зі значень, що визначаються в перерахуванні FileAccess і наведених ніже.FileAccess.Read FileAccess.Write FileAccess.ReadWrite Наприклад, в наступному прикладі коду файл test.dat відкривається тільки для чтенія.FileStream

fin = new FileStream ("test.dat ", FileMode.Open, FileAccess.Read); По завершенні роботи з файлом його слід закрити, викликавши метод Close (). Нижче наведена загальна форма звернення до цього методу.void Close () Читання байтів з потоку файлового введення-виводаВ класі FileStream визначені два методу для читання байтів з файлу: ReadByte () і Read (). Так, для читання одного байта з файлу використовується метод ReadByte (), загальна форма якого наведена ніже.int ReadByte () Всякий раз, коли цей метод викликається, з файлу зчитується один байт, який потім повертається у вигляді цілого значення. До числа ймовірних винятків, які генеруються при цьому, відносяться NotSupportedException (потік не відкритий для введення) і ObjectDisposedException (потік закритий) .Для читання блоку байтів з файлу служить метод Read (), загальна форма якого виглядає так.int Read (byte [] array, int offset, int count) у методі Read () робиться спроба вважати кількість count байтів в масив array, починаючи з елемента array [offset]. Він повертає кількість байтів, успішно лічених з файлу. Якщо ж виникає помилка введення-виведення, то генерується виключення IOException. До числа інших ймовірних винятків, які генеруються при цьому, відноситься NotSupportedException. Це виняток генерується в тому випадку, якщо читання з файлу не підтримується в потоке.Запісь в файлДля записи байта в файл служить метод WriteByte (). Нижче наведена його найпростіша форма.void WriteByte (byte value) Цей метод виконує запис в файл байта, що позначається параметром value. Якщо базовий потік не відкривається для виводу, то генерується ісключеніеNotSupportedException. А якщо потік закритий, то генерується виключення ObjectDisposedException.Для записи в файл цілого масиву байтів може бути викликаний метод Write (). Нижче наведена його загальна форма.void Write (byte [] array, int offset, int count) У методі Write () робиться спроба записати в файл кількість count байтів з масиву array, починаючи з елемента array [offset]. Він повертає кількість байтів, успішно записаних в файл. Якщо під час запису виникає помилка, то генерується виключення IOException. А якщо базовий потік не відкривається для виводу, то генерується виключення NotSupportedException. Крім того, може битьсгенерірован ряд інших винятків. При виведенні в файл виводяться

дані часто записуються на конкретному фізичному пристрої не відразу. Замість цього вони буферизуются на рівні операційної системи до тих пір, поки не нагромадиться достаточнийоб'ем даних, щоб записати їх відразу одним блоком. Завдяки цьому підвищується ефективність системи. Так, на диску файли організовані за секторами величиною від 128 байтів і більше. Тому що виводяться дані зазвичай буферизуются до тих пір, поки не з'явиться можливість записати на диск відразу весь сектор. Але якщо дані потрібно записати на фізичний пристрій без попереднього накопичення в буфері, то для цієї мети можна викликати метод Flush.void Flush () При невдалому результаті даної операції генерується виключення IOException. Якщо ж потік закритий, то генерується виключення ObjectDisposedException. По завершенні виведення в файл слід закрити його за допомогою методу Close (). Цим гарантується, що будь-які виведені дані, що залишилися в дисковому буфері, будуть записані на диск. У цьому випадку відпадає необхідність викликати метод Flush () перед закриттям файла.Прімер. Вважати записати рядок в текстовий файл. private void button1_Click (object sender, EventArgs e) {string text = textBox1.Text; // Запис в файлusing (FileStream fstream = new FileStream (@ "C: \ note.txt", FileMode.OpenOrCreate)) {// перетворимо рядок в байти byte [] array = System.Text.Encoding.Default.GetBytes (text); // Запис масиву байтів в файл fstream.Write (array, 0, array.Length); MessageBox.Show ("Текст записаний в файл");}} private void button2_Click (object sender, EventArgs e) {// читання з файлу using (FileStream fstream = File.OpenRead (@ "C: \ note.txt")) {// перетворимо рядок в байти byte [] array = new byte [fstream.Length]; // Зчитуємо дані fstream.Read (array, 0, array.Length); // Декодуємо байти в рядок string textFromFile = System.Text.Encoding.Default.GetString (array); label2.Text = "Текст з файлу:" + textFr omFile; }} Розберемо цей приклад. І при читанні, і при запису використовується оператор using. Не треба плутати даний оператор з директивою using, яка підключає простору імен на початку файлу коду. Оператор using дозволяє створювати об'єкт в блоці коду, по завершенню якого викликається метод Dispose у цього об'єкта, і, таким чином, об'єкт знищується. В даному випадку в якості такого об'єкта служить змінна fstream.Об'єкт fstream створюється двома

різними способами: через конструктор і через один з статичних методів класу File.Проізвольний доступ до файламНередко бінарні файли представляють певну стрктуру. І, знаючи цю структуру, ми можемо взяти з файлу потрібну порцію інформації або наоброт записати в певному місці файлу певний набір байтів. Наприклад, в wav-файлах безпосередньо звукові дані починаються з 44 байта, а до 44 байта йдуть різні метадані - кількість каналів аудіо, частота дискретизації і т.д. З допомогою методу Seek () ми можемо управляти положенням курсору потоку, починаючи з якого виробляється зчитування або запис в файл. Цей метод приймає два параметри: offset (зміщення) і позиція в файлі. Позиція в файлі описується трьома значеннями: • SeekOrigin.Begin: початок файлу • SeekOrigin.End: кінець файлу • SeekOrigin.Current: поточна файлаКурсор потоку, з якого починається читання або запис, зміщується вперед на значення offset щодо позиції, зазначеної в якості другого параметра . Зсув може негативним, тоді курсор зсувається назад, якщо позитивне - то вперед.Чтеніе і запис текстових файлів. StreamReader і StreamWriterКласс FileStream не дуже зручно застосовувати для роботи з текстовими файлами. До того ж для цього в просторі System.IO визначені спеціальні класи: StreamReader і StreamWriter.Чтеніе з файлу і StreamReaderКласс StreamReader дозволяє нам легко зчитувати весь текст або окремі рядки з текстового файлу. Серед його методів можна виділити наступні: • Close: закриває зчитування файл і звільняє всі ресурси • Peek: повертає наступний доступний символ, якщо символів більше немає, то повертає -1 • Read: зчитує і повертає наступний символ в чисельному поданні. Має перевантажену версію: Read (char [] array, int index, int count), де array - масив, куди зчитуються символи, index - індекс в масиві array, починаючи з якого записуються зчитувальні символи, і count - максимальна кількість зчитувальних символів • ReadLine : зчитує один рядок в файлі • ReadToEnd: зчитує весь текст з файлаЗапісь в файл і StreamWriterДля записи в текстовий файл використовується клас StreamWriter. Свою функціональність він реалізує через такі методи: • Close: закриває записується файл і звільняє всі ресурси • Flush: записує в файл залишилися в буфері дані і очищає буфер. • Write: записує в файл дані найпростіших типів, як int, double, char, string і

т.д. • WriteLine: також записує дані, тільки після запису додає в файл символ закінчення строкіПрімер. Створити клас, для роботи з одновимірним масивом, а саме: 1. Читання масиву з файла2. Записи масиву в файл3. Розрахунків відповідно до варіанту: Перенести в початок масиву всі негативні елементи кратні трем.4.Запісі результатів розрахунків в файл Класccclass File_Arrays {int [] a; int [] b; int length; public bool f = false; public File_Arrays () {a = new int [100]; b = new int [100]; length = 100; } Void show (TextBox t, int [] aa) {t.Text = ""; for (int i = 0; i <length; i ++) {t.AppendText (aa [i] + ""); }} Public void ct (string s, TextBox t) {int i = 0; using (StreamReader sr = new StreamReader (s, System.Text.Encoding.Default)) {string line; while ((line = sr.ReadLine ())! = null) {a [i] = Convert.ToInt16 (line); i ++; }} Length = i; show (t, a); } Public void zap () {string s = @ "C: \ rez.txt"; using (StreamWriter sw = new StreamWriter (s, false, System.Text.Encoding.Default)) {for (int i = 0; i <length; i ++) {sw.WriteLine (b [i] .ToString ()); }}} Public void ras (TextBox t) {int i = 0, j = 0, i1 = 0; int [] b1, b2; b1 = new int [100]; b2 = new int [100]; for (int ii = 0; ii <length; ii ++) {if ((a [ii] <0) && (a [ii]% 3 == 0)) {b1 [i] = a [ii]; i ++; } Else {b2 [j] = a [ii]; j ++; }} For (int ii = 0; ii <i; ii ++) {b [i1] = b1 [ii]; i1 ++; } For (int ii = 0; ii <j; ii ++) {b [i1] = b2 [ii]; i1 ++; } Show (t, b); }} Основна программаprivate void button1_Click (object sender, EventArgs e) {openFileDialog1.Title = "Виберіть файл"; // Вихід, якщо була натиснута кнопка Скасування та інші (крім OK) if (openFileDialog1.ShowDialog ()! = DialogResult.OK) return; // Усе. ім'я файлу тепер зберігається в openFileDialog1.FileName textBox1.Text = openFileDialog1.FileName; } Private void button2_Click (object sender, EventArgs e) {File_Arrays f = new File_Arrays (); string s = textBox1.Text; f.ct (s, textBox2); f.ras (textBox3); f.zap (); } СеріалізаціяСеріалізація представляє процес перетворення будь-якого об'єкта в потік байтів. Після перетворення ми можемо цей потік байтів або записати на диск або зберегти його тимчасово в пам'яті. А при необхідності можна виконати зворотний процес - десеріалізацію, тобто отримати з потоку байтів раніше збережений об'єкт.Ранее ми подивилися, як зберігати інформацію в текстові файли, а також торкнулися збереження нескладних структур в бінарні дані. Але нерідко подібних механізмів виявляється недостатньо особливо

для збереження складних об'єктів. З цією проблемою покликаний впорається механізм серіалізації. Серіалізация представляє процес перетворення будь-якого об'єкта в потік байтів. Після перетворення ми можемо цей потік байтів або записати на диск або зберегти його тимчасово в пам'яті. А при необхідності можна виконати зворотний процес - десеріалізацію, тобто отримати з потоку байтів раніше збережений об'ект.Атрібут SerializableЧтоби об'єкт певного класу можна було серіалізовать, треба цей клас помітити атрибутом Serializable: [Serializable] class Person {public string Name {get; set; } Public int Year {get; set; } Public Person (string name, int year) {Name = name; Year = year; }} При отстутствии даного атрибута об'єкт Person не зможе бути серіалізовані, і при спробі серіалізації буде викинуто виключення SerializationException.Еслі ми не хочемо, щоб якийсь член класу серіалізовані, то ми його помічаємо атрибутом NonSerialized: [Serializable] class Person {public string Name {get; set; } Public int Year {get; set; } [NonSerialized] public string AccNumber {get; set; } Public Person (string name, int year) {Name = name; Year = year; }} При спадкуванні подібного класу, слід враховувати, що атрибут Serializable автоматично не успадковується. І якщо ми хочемо, щоб похідний клас також міг би бути серіалізовані, то знову ж таки ми застосовуємо до нього атрибут: [Serializable] class Worker: PersonФормат серіалізаціїХотя сериализация є перетворення об'єкта в певний набір байтів, але в дійсності тільки бінарним форматом вона не обмежується . Отже, в .NET можна використовувати такі формати: • бінарний • SOAP • xml • JSONДля кожного формату передбачений свій клас: для серіалізації в бінарний формат - клас BinaryFormatter, для формату SOAP - клас SoapFormatter, для xml - XmlSerializer, для json - DataContractJsonSerializer. для класів BinaryFormatter і SoapFormatter сам функціонал серіалізації визначено в інтерфейсі IFormatterДля серіалізації використовується метод Serialize, який в якості параметрів приймає потік, куди поміщає серіалізовані дані (наприклад, бінарний файл), і об'єкт, який треба серіалізовать. А для десеріалізациі буде застосовуватися метод Deserialize, який в якості параметра приймає потік з серіалізовані даними.