Práce s textovými soubory

Úvod

V této kapitole si ukážeme, jak číst a zapisovat do textových souborů.

Cesty

Než si ukážeme konkrétní třídy a metody pro práci se soubory, vysvětlíme si některá fakta ohledně zápisu cest k souborům v jazyce C#. K přístupu do podadresářů používáme lomítka. V unixových systémech používáme klasické lomítko (`/`) a na platformě Windows lomítko zpětné (`\`). Jazyk C# umožňuje použití obou možností, ale z historických důvodů bychom důrazně doporučovali držet se lomítka zpětného.

 # Zpětné lomítko

Kromě znázorňování hierarchické struktury adresářů slouží lomítko i k tzv. escapování znaků. Pro lepší vysvětlení tohoto pojmu si vše ukážeme na jednoduchém příkladě. Představme si situaci, kdy chceme do stringové proměnné s uložit jednu uvozovku:

Ukázky

```
string s = """; // Ukázka 1
string s = "\""; // Ukázka 2
```

Kompilátor by uvozovku ve prostřed vnímal jako validní znak a ukončil by tak textový řetězec na pravé straně výrazu. Třetí uvozovka by pak byla vnímána jako další neukončený řetězec. Zpětné lomítko je cesta, jak uvozovku escapnout a říct tím kompilátoru, aby ji vnímal jako obyčejný literál a nikoliv jako klíčový znak (viz ukázka 2). Pokud tedy budeme chtít uložit cestu k souboru do stringové proměnné, musíme nějakým způsobem kompilátoru říct, aby zpětné lomítko vnímal jako literál pro oddělení adresářové struktury a nikoliv jako pokus o escapování následujícího znaku. Máme hned dvě možnosti, jak tohoto chování docílit

Možnosti

```
string cesta = "c:\\pepa\\aplikace\\data.txt"; // Možnost 1
string cesta = @"c:\pepa\aplikace\data.txt"; // Možnost 2
```

První možností je escapnout samotný znak escapnutí a použít tedy dvě zpětná lomítka. Druhou, více elegantnější možností je prefixovat celý řetězec znakem zavináče (`@`), čímž řekneme kompilátoru, aby všechny znaky v řetězci vnímal jako literály.

> Pokud vyvíjíte na unixovém operačním systému (např. MacOS nebo Linux), musíte používat klasické lomítka (`/`)!

 # Relativní cesty

Jazyk C# nám dává možnost odvíjet cesty i relativně od rootovského adresáře. Root neboli kořenový adresář programu je taková složka, kde se nachází spustitelný `.exe` soubor (nebo jiný spouštěcí soubor) naší aplikace. Standardně je to složka `název-projektu\bin\debug\`. Cestu označíme jako relativní, pokud na její začátek uvedeme tečku (`.`):

Cesty

```
string relativniCesta = @".\data"; // Relativní cesta
string absolutniCesta = @"c:\aplikace\mujProgram\data"; // Absolutní cesta
```

Vždy je lepší cesty odvíjet relativně od spouštěcího souboru, neboť absolutní cesty mohou přestat fungovat v momentě, kdy program spustíme na jiném počítači. V ukázce 2 je pak na prohlédnutí i cesta absolutní. Pro vstoupení do vyšší adresářové struktury (rozuměj pro opuštění současné složky) pak slouží tečky dvě:

Notace dvou teček

```
// Začneme v kořenovém adresáři, vystoupíme o úroveň výše
// a poté vstoupíme do složky se jménem "data" a vybereme
// soubor se jménem "soubor.txt":
string cesta = @".\..\data\soubor.txt";

/*

aplikace/
├── root/ (kořenový adresář)
│   └── aplikace.exe
├── data/
│   └── soubor.txt (vybraný soubor)
└── src/
    ├── obrazek.png
    └── hudba.mp3

*/
```

Třída File

Nejjednodušší způsob, jak manipulovat s textovým souborem je třída `File`. Ta obsahuje statické metody, které pro nás obstarají všechny základní funkčnosti, které bychom mohli při práci se soubory potřebovat. Třída `File` je šikovná i v tom, že soubor sama otevře i uzavře, takže tuto činnost nemusíme provádět ručně. Abychom mohli třídu `File` vůbec používat, musíme našemu projektu říct, aby využíval namespace `System.IO`, který třídu obsahuje. Uděláme to pomocí klíčového slova `using`:

Namespace import

```
using System.IO;
```

> Tento namespace (i jakýkoliv jiný) si pochopitelně nemusíme pamatovat zpaměti. Každé dobré vývojové prostředí nám tento namespace sám nabídne k doplnění v momentě, kdy se na třídu `File` odkážeme.

 # Vytvoření souboru

Začneme od nejjednoduššího a ukážeme si, jak vytvoříme nový `.txt` soubor. K tomuto účelu slouží metoda `Create()`:

Vytvoření souboru

```
// Vytvoří textový soubor v kořenovém adresáři
File.Create("soubor.txt");
```

> Pokud vytvářený soubor již existuje, dojde k přepsání toho stávajícího!

Třída dále nabízí metody `Move()` a `Delete()`, jejichž názvy jsou více méně sebepopisné:

Přesunutí souboru

```
// Přesun souboru "soubor.txt" z kořenového adresáře
// do složky "data"
File.Move("soubor.txt", @".\data\soubor.txt");
```

Smazání souboru

```
// Smaže soubor "soubor.txt" z
// kořenového adresáře
File.Delete("soubor.txt");
```

Čtení ze souboru

Nyní se podíváme na metody určené ke čtení ze souborů. Dvě nejčastěji používané z nich jsou `ReadAllText()` a `ReadAllLines()`. Oběma stačí v závorce pouze cesta k souboru:

Čtení ze souboru

```
// Načte kompletně celý obsah souboru do stringové proměnné
string obsah = File.ReadAllText("soubor.txt");

// Načte obsah celého souboru po řádcích do pole
string[] radky = File.ReadAllLines("soubor.txt");
```

> Tyto metody načítají obsah souboru do RAM paměti počítače. Pokud bychom pracovali s opravdu velkým textovým souborem v řádech statisíců znaků, máme pro tyto účely třídu `StreamReader`, která je na takovéto situace lépe uzpůsobená.

Obě metody dělají totéž, ale každá vrátí výstupní data v jiném datovém typu. Metoda `ReadAllText()` vrací celý obsah v jedné stringové proměnné a nové řádky odděluje symbolem pro to určeným, kdežto `ReadAllLines()` vrací stringové pole, kde každý záznam je právě jeden řádek.

> Symbol(y) pro nový řádek je na platformě Windows `\r\n` a na unixových operačních systémech `\n`. Pokud si nejsme jistí, jaký symbol použít, máme pro tyto účely k dispozici i vlastnost `Environment.NewLine`

Zápis do souboru

K jednoduchému zápisu využíváme metody `WriteAllText()` a `WriteAllLines()`. Fungují podobně, jako jejich protějšky na čtení:

Zápis do souboru

```
// Pole s textem
string[] pole = new string[] {"Věta 1", "Věta 2"};

// Zápis do souboru
WriteAllText("soubor.txt", "Text, který chceme zapsat do souboru.");
WriteAllLines("soubor.txt", pole);
```

Opět platí, že pokud cílový soubor neexistuje, vytvoří se nový. Pokud textový soubor již obsahuje nějaký text, dojde k jeho přemazání. Pokud nechceme přepsat celý soubor a ztratit tím všechna data, co v něm předtím byla, můžeme použít metodu `AppendAllText()`. Tato metoda vezme obsah, který v souboru byl a na konec přidá námi zadaný text:

Zápis do souboru

```
File.AppendAllText("soubor.txt", "Text, který se přidá na konec, bez přepsání původního obsahu souboru.");
```

Dalším způsobem jak pracovat s textovými soubory jsou třídy `StreamReader` a `StreamWriter`. Tyto třídy nenačítají soubory do své paměti celé najednou, ale umožňují z nich číst nebo do nich zapisovat řádek po řádku. Daní za tento fakt je zdlouhavější zápis, neboť musíme vytvářet jejich instance a poté i ručně uzavírat datový stream. Výhodou tohoto přístupu je však možnost pracovat s objemnými soubory v poměrně krátkém čase za rozumného vytížení paměti.

 # Třída StreamReader

Když vytvoříme instanci třídy `StreamReader`, dojde k otevření souboru, který jsme mu dodali jako argument. V praxi to znamená, že od této doby k němu nedostane přístup žádný jiný program. Aby se nám nestalo, že necháme nějaký soubor "zamknutý" musíme ho vždy poté, co jsme s ním hotovi, zavřít pomocí metody `Close()`:

Založení StreamReaderu (1)

```
// Vytvoření nové instance třídy StreamReader
StreamReader reader = new StreamReader("soubor.txt");

// ... logika čtení

// Uzavření streamu
reader.Close();
```

Druhou naší možností je samotné instancování třídy obalit do bloku `using`, jež sám zajistí uzavření streamu:

> V prvním ročníku jsme zmínili existenci tzv. `garbage collectoru`, který automatizuje správu paměti. V praxi to znamená, že pokud založíme novou proměnnou, běhové prostředí automaticky rozpozná, kdy proměnná již nebude potřeba a alokovanou (rozuměj zabranou) paměť samo uvolní. Některé zdroje jako naše textové soubory jsou však mimo kompetence tohoto collectoru a o jejich uvolnění se musíme postarat sami. Třídy pracující s takovýmito zdroji implementují rozhraní `IDisposable`, které nutí danou třídu implementovat metodu `Dispose()`, jež se stará o uklizení "nepořádku" po volání takovýchto tříd. Mezi takovéto třídy spadají i naše třídy `StreamReader` a `StreamWriter`. Jejich použití v using bloku zajistí zavolání výše zmíněné `Dispose()` metody a vyhneme se tak tudíž potencionálním nepříjemnostem. Tato informace je spíše doplňující látkou pro pokročilé.

Díky tomu, že nám soubory zůstávají otevřené, nemusíme číst celý soubor najednou, ale můžeme s ním manipulovat jen po řádcích či znacích. Když používáme jednotlivé metody na čtení, náš reader si zapamatuje pozici, na které přestal znaky číst:

Čtení po znaku

```
// Vytvoření nové instance třídy StreamReader
using (StreamReader reader = new StreamReader("soubor.txt"))
{
    // Dokud nám zbývají nepřečtené znaky
    while (reader.Peek() >= 0)
    {
        // Převede výstup metody Read() z intu
        // (ASCCI hodnota znaku) na obyčejný znak (char)
        // a poté vypíše výsledek
        Console.Write((char)sr.Read());
    }
}
```

Takovýto kód by obsah souboru přečetl po jednom znaku a každý zvlášť vypsal do konzole. Podobně si počínáme i v případě, kdy chceme souborem iterovat po řádcích:

Čtení po řádcích

```
using (StreamReader reader = new StreamReader("soubor.txt"))
{
    // Pomocná proměnná pro uložení řádku
    string line;

    // Dokud nám zbývají další řádky
    while ((line = reader.ReadLine()) != null)
    {
        // Vypíšeme získaný řádek do konzole
        Console.WriteLine(line);
    }
}
```

 # Třída StreamWriter

Pro zapisování máme ve třídě `StreamWriter` jen dvě metody - `Write()` a `WriteLine()`. Jejich použití je téměř shodné. Obě varianty do souboru zapíší předaný řetězec, ale metoda `WriteLine()` ještě na konec přidá nový řádek

Zápis do souboru

```
using (StreamWriter writer = new StreamWriter("soubor.txt"))
{
```

```
    // Zapíše řetězec na aktuální řádek
    writer.Write("Požadovaný řetězec");

    // Zapíše řetězec na nový řádek
    writer.WriteLine("Budu na novém řádku!");
}
```

Podobně jako u třídy `StreamReader` musíme opět uzavřít stream buď voláním metody `writer.Close()` nebo obalením celého výrazu `using` blokem tak, jako na ukázce výše.

Shrnutí

V této kapitole jsme se seznámili se zápisem a čtením z textových souborů. Měli bychom být obeznámeni se třídami `File`, `StreamReader` a `StreamWriter`.

Revision #1
Created 2025-05-21 10:42:33 UTC by Magdalena Dobešová
Updated 2025-05-21 12:37:32 UTC by Magdalena Dobešová