Code Academy

Dėstytojas

Vilmantas Neviera

Streams

Data

Šiandien išmoksite

01 Kas yra streams?

02 File klasė

03 StreamReader/StreamWriter

04 FileStream

Stream

Stream yra abstrakti klasė suteikianti įvairias galimybes dirbant su byte'ais(pvz.: skaityti, rašyti).

Yra įvairių stream tipų, bet šiandien pakalbėsime **FileStream, StreamWriter, StreamReader** klasėm.

Stream abstrakcijų lygiai

Peržvelkime skirtingus abstrakcijų lygius skaitant iš failų.

Lygiai surašyti sudėtingumo ir kontrolės didėjimo tvarka:

1. **File** klasė.
2. **File** klasės **ReadLines()** metodas.
3. **StreamReader** ir **StreamWriter** klasės.
4. **FileStream** klasė.

File klasė

Pats lengviausias ir mažiausiai efektyvus būdas yra **File** klasė su metodais:

1. **File.ReadAllText**.
2. **File.ReadAllLines**.
3. **File.WriteAllText**.
4. **File.WriteAllLines**.

```
static void Main(string[] args)
{
    string content = File.ReadAllText("path.txt");
    File.WriteAllText("anotherPath.txt", content);
}
```

Privalumai:

Paprasčiausias būdas skaityti ar rašyti iš failo.

Minusai:

Skaito/Rašo visą tekstą iškarto, limituotos galimybės kontroliuoti prieigas prie failų, skirtingų encodinimų ir buffering’o.

File klasės ReadLines su Lazy Evaluation

Kiek sudėtingesnis, bet daug greitesnis skaitymas didesniems failams.

ReadLines grąžina **IEnumerable** grąžina **yield** kas reiškia jog paleidus metodą jis pradės skaityti failą ir iškarto leis naudoti realiu laiku grąžinamą tekstą(grąžinimai vyksta po eilutę), kitaip sakant jeigu iteruosite didelį failą programai nereikės laukti, kada **File** klasė baigs skaityti failą, ji galės iškarto jį pradėti skaityti.

Privalumai:

Lazy evaluation leidžia skaityti po eilutę neužkraunant viso failo į atmintį iškarto. Su laiku failas failas bus perskaitytas visas į atmintį, bet jūs galėsite nutraukti kodą jeigu nebereikės skaityti.

Minusai:

Vistiek limituotos galimybės kontroliuoti prieigą prie failų ar encodinimų.

```
0 references
static void Main(string[] args)
{
    IEnumerable<string> content = File.ReadLines("path.txt");
}
```

StreamReader ir StreamWriter klasės

Kaip pavadinimai ir sufleruoja **StreamReader** klasė yra skirta skaityti iš failo ir **StreamWriter** klasė skirta rašyti į failą.

Privalumai:

Lengvesnis skaitymas tekstinių failų, automatinis tvarkymas skirtingų encodinimų, **lazy evaluation** skaitymas, galimybė keisti **buffering** nustatymus.

Minusai:

Sudėtingesnis naudojimas nei **File** klasės metodai, pagrinde naudojamas skaityti tekstinius failus.

StreamReader ir StreamWriter klasės

```
static void Main(string[] args)
{
    DirectoryInfo[] cDirs = new DirectoryInfo(@"c:\").GetDirectories();

    using (var writer = new StreamWriter("CDriveDirs.txt"))
    {
        foreach (var dir in cDirs)
        {
            writer.WriteLine(dir.Name);
        }
    }

    string line = "";
    using (var reader = new StreamReader("CDriveDirs.txt"))
    {
        while ((line = reader.ReadLine()) != null)
        {
            Console.WriteLine(line);
        }
    }
}
```

FileStream

Sudėtingiausias būdas skaityti failus, taip pat, turintis didžiausią kontrolę:

1. Galimybė specifikuoti **reader'io** teises į failą(read/write)
2. Skirtingi **buffering** būdai
3. Failo dalinimasis
    a. **Exclusive access**: Niekas kitas negali nieko daryti su failu kol šitas **FileStream** jį naudoja.
    b. **Read sharing**: kiti procesai gali skaityti iš failo, bet negali įrašyti.
    c. **Write sharing**: kiti procesai gali įrašyti į failą, bet negali skaityti.
    d. **Read/Write sharing**: Kiti procesai gali tiek skaityti tiek rašyti į failą.

```
static void Main(string[] args)
{
    var fileStream = new FileStream("path.txt", FileMode.Open, FileAccess.Read, FileShare.Read);
}
```

Kurį pasirinkti?

Jeigu jum reikia perskaityti nedidelį tekstinį failą - rinkitės File.ReadAllLines/File.ReadAllText tipo metodus

Jeigu paprastas tekstinis failas, bet jo dydis yra 100mb+ naudokite File.ReadLines().

Jeigu failas didesnis nei 100 megabaitų, arba jis nėra tekstinio formato, arba ne standartinio encodinimo, naudokite StreamReader/StreamWriter

Jeigu išskirtinis encodinimas, didelis failas, reikia leisti kitiem procesam kažką daryti su failu tuo pačiu metu ir bendrai reikia kompleksyvių skaitymo nustatymų - naudokite FileStream.

Užduotis nr. 1

- Perskaitykite visą tekstinį failą naudodami File.ReadAllText() metodą ir išspausdinkite į konsolę.
- Įrašykite į failą turinį iš List<string> sąrašo, kiekvieną elementą kaip naują eilutę. Naudodami File.WriteAllLines().
- Perkopijuokite failą iš vienos vietos į kitą naudodami File.Copy() metodą.

Užduotis nr. 2

- Perskaitykite didelį tekstinį failą po eilutę ir suskaičiuokite kiekvienos eilutės simbolių skaičių. Naudokite File.ReadLines()
- Įrašykite į failą naudodami StreamWriter klasę, įrašinėkite tiek tekstinius tiek skaitinius duomenis.
- Įrašykite binary duomenis į failą, pabandykite panaudoti BinaryReader ir BinaryWriter

Užduotis nr. 3

Advanced, bus nežinomų dalykų iš nepraeitų temų:

- Perskaitykite didelį failą naudodami custom buffer dydį ir paanalizuokite, kokį skirtumą tai padarė failo skaitymo efektyvume.
- Implementuokite asinchroninį failo skaitymą kuris neblokuoja veikiamos programos pagrindinio thread.
- Implementuokite file-based caching sistemą, kuri saugos key-value poras faile ir leis efektyviai jas išsitraukt ar įrašyt. Naudokite FileStream klasę efektyviam skaitymo kontroliavimui, potencialiai implementuodami indexing ar kitas efektyvumo strategijas.