

23. Vlákna, Paralelní programování, Asynchronní metody

Proces

Proces je instance běžící aplikace. Většinou mají aplikace však jen jeden. Některé složitější aplikace mohou využívat několik procesů, např. Google Chrome vytvoří proces pro každou otevřenou záložku.

Vlákno

V procesu může běžet několik jeho vláken. Každá aplikace má minimálně jeden proces, ve kterém běží její hlavní vlákno nebo další vlákna. Hlavní vlákno je vytvořeno programem. Další vlákna vytváří vývojář sám. Operační systém spustí vlákno na nějakém jádru a potom ho uspává a probouzí jak se mu to zrovna hodí. Vlákna v procesu běží paralelně.

Vlákno na pozadí

Říká se mu démon. Od základu mají stejnou prioritu jako vlákna na popředí. Příklad toho by mohla být okenní aplikace, vlákno na popředí je hlavní a řídí vzhled, ovládací prvky, eventy. Vlákno na pozadí třeba připojení k serveru. Pokud hlavní vlákno skončí a jsou spuštěna nějaká vlákna na pozadí, jsou tato vlákna automaticky ukončena bez ohledu na to, jakou činnost prováděla. Ale pokud by činnost okna nebylo hlavní vlákno a převzalo by tuto úlohu jiné, vývojářem vytvořené, tak se musí ukončit všechna vlákna samostatně.

Synchronizace

Vlákna se v praxi používají, jen pokud jsou nutně potřebné. Je s nimi spojený jeden problém - synchronizace. Nikdy se neví, kdy bude vlákno uspáno, protože to je prováděno systémem.

Třída Thread

Statická metoda `Sleep(milisekundy)`

uspí vlákno na určitou dobu.

Místo milisekund se na delší čas dá použít `TimeSpan.FromHours()`

Pokud se chce vlákno pouze zablokovat za účelem, aby bylo přepnuto na jiné, tak stačí nechat vlákno spát na 0 ms, v metodě `Sleep 0` nebo `nic`. Tohohle se dá dosáhnout i metodou `Thread.Yield()`

Má vlastnost `ThreadState` – obsahuje informaci o stavu vlákna.

Metoda `Join()` – zablokuje aktuální vlákno, dokud druhé neskončí.

vlastnost `priority` – určuje, že vlákno dostává více času.

`Highest`, `AboveNormal`, `Normal`, `BelowNormal`, `Lowest`

vlákno má vlastnost `Name`, může mít jméno.

Locker

Uzamčení vláken zajišťuje, že jedno vlákno nevstupuje do kritické části kódu, zatímco jiné vlákno je v kritické části kódu. Jinak by mohla vzniknout chyba, které by třeba změnila 2krát nějaké proměnné, které chceme změnit pouze jednou.

Činí metodu `thread-safe`.

Pokud se jiné vlákno pokusí zadat “zamčený kód“, bude blokováno a čekat do doby dokud nebude objekt uvolněn.

To je provádí objektem, který je zvaný `locker`

Tento `locker` využívá třídu `Monitor`, které obsahuje statické metody `Enter()` pro zamknutí a `Exit()` pro odemknutí.

Dá se to představit jako `bool` reprezentující zamknutí, který pomocí metody `Enter()` změní hodnotu na `true` nebo když je `false` tak čeká na odemčení.

`Exit()` `bool` změní na `false`, aby byl kód odemčený pro jiné vlákna.

`Enter()` způsobí, že vlákno nelze uspat.

Existuje i metoda `TryEnter()` která vyžaduje časový interval, jakou dobu má čekat na uzamčený kód, po uplynutí času, když je kód pořád zamčený ho přeskočí.

Třída Interlocked

Třída pro jednoduché operace jak matematické jednoduché operace. Pro které by nebylo výhodné psát `locker`.

Třída ThreadPool

Vytvoření nového vlákna je na zdroje poměrně náročná operace, která zkonsumuje několik set ms času a také poměrně vysoké množství paměti. Proto je výhodné nevytvářet stále nová vlákna, ale recyklovat vlákna již existující, čímž snížíme nároky na systém.

K tomu slouží ThreadPool, který efektivně spravuje větší množství vláken. Také zajišťují, aby vláken neběželo příliš mnoho najednou a proces nebyl neefektivní.

Vlákno v ThreadPoolu je vždy v pozadí a nemá jméno.

Každé vlákno má vlastnost `IsThreadPoolThread`, která vrací zda je vlákno spravováno v ThreadPoolu

Funguje jako Object pool pattern.

Asynchronní delegát

Asynchronní delegáti řeší problém předávání parametrů vláknu, získávání jeho návratové hodnoty a také propagaci výjimek z vlákna do metody, která ho spustila.

Přidává se k nim callback, který oznamí až metoda skončí, aby se na konec metody nemuselo čekat.

Třída Task

Jsou to menší části aplikace, které běží paralelně a mohou se skládat do sebe. Vnitřně jsou reprezentovány jako background vlákna v poolu.

Když svou aplikaci tedy rozdělíme na úlohy, získáme vyšší výkon.

Metoda `Run()` vrací novou instanci třídy `Task`, aby bylo možné mít instanci probíhající úlohy. Instance slouží hlavně k monitorování stavu úlohy, pomocí vlastností `Status` a `IsCompleted`.

Metoda `wait()` je stejná jako `Join()` u `Thread`.

`Task` může mít i návratový typ a být generický. Návratový typ se získá pomocí vlastnosti `Result`. Ale pokud bude zavolán, tak povinně čeká, až dostane `return`, proto by se mělo vlákno, které ho chce zeptat, jestli je metoda dokončena.

`Task` také obsahuje vlastnosti `IsCancelled`, `IsFaulted` které obsahují informaci zda byla nějaké chyby s provedením metody.

Čekání na task

`Awaiter` je objekt, který má v sobě událost `OnComplete`, který se získá pomocí metody `GetAwaiter()`.

Používá se k hlídání informace, jestli task skončil, poté si s něj může vzít návratový typ pomocí metody `getResult()`

Asynchronní metody

Příklad klientská aplikace, která se snaží připojit na server, připojení by ale mohlo zabrat několik sekund (tak 2) a po tu dobu nemůže reagovat na požadavky od uživatele. Použití vlákna by bylo na takový problém zbytečné. Proto se použije asynchronní metoda.

Asynchronní metoda je metoda, která nevrátí svou hodnotu ihned, ale až po nějakém časovém intervalu. Mezitím program na návratovou hodnotu volané metody nečeká, ale pokračuje dále.

Je pouze u nějakých tříd: `TcpClient`, `StreamWriter`, `StreamReader`, `XmlReader`. Název těchto metod končí slovem `Async` a vrací objekt typu `Task`.

Nevyvolá vytvoření nového vlákna, vůbec nemusí multithreading používat.

Technologie je o neblování současného vlákna metodami, které čekají na dokončení úlohy. Do jiného vlákna se může přesunout pomocí `Task.Run()`.

Asynchronní programování je ve většině případů výhodnější než vlákna.

Kód je jednodušší, nepoužívá se `locker`.

Concurrent design patterns

Návrhový vzor, který se zabývá multi-thread programem.

Active Object

Tento návrhový vzor odděluje spouštění metod od provádění metod, přičemž spouštění metod může být ve svém vlastním vlákně. Cílem je přidat souběžnost použitím asynchronních volání metod a plánovače, který obsluhuje požadavky.

Event-based asynchronous

Na událostech založený asynchronní návrhový vzor řešící problémy s Asynchronním vzorem, které nastávají ve vícevláknových programech.

Balking

Tento vzor je softwarovým vzorem, který na objektu vykoná nějakou akci, pouze pokud je objekt v určitém stavu.

Double checked Locking

Tento vzor je také znám jako „optimalizace zamykání s dvojnásobnou kontrolou“. Návrh je vytvořen tak, aby zredukoval zbytečné náklady na získávání zamčení tím, že nejdříve otestuje kritérium pro zamčení nezabezpečeným způsobem ('lock hint'). Pouze pokud uspěje, pak se opravdu zamkne.

Guarded

Jde o vzor obstarávající operace, které požadují uzamčení a navíc mají nějakou podmínku, která musí být splněna předtím, než může být operace provedena.

Monitor Object

Monitor je přístup k synchronizaci dvou nebo více počítačových úloh, které používají sdílené zdroje, zpravidla hardwarové zařízení nebo sadu proměnných.

Read write lock

Tento vzor, také známý jako RWL, je vzor, který umožňuje souběžný přístup k objektu pro čtení, ale vyžaduje exkluzivní přístup pro zápis.

Scheduler

Jde o souběžný vzor, který se používá pro explicitní kontrolu, kdy mohou vlákna vyvolávat jednovláknový kód.