Threading

TEORIA DI PROGETTAZIONE DEL SOFTWARE

Caratteristiche principali dei thread

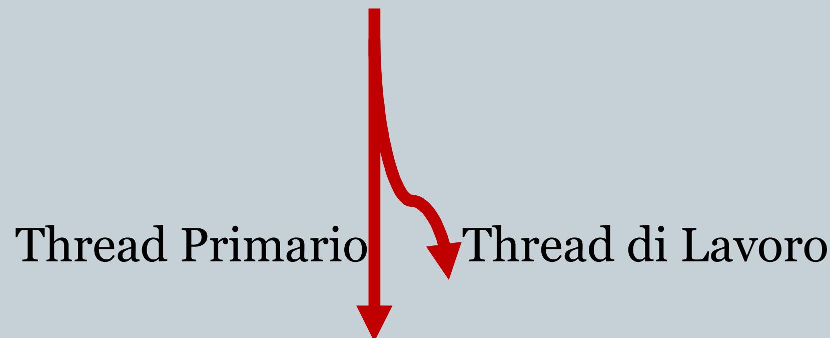
- Consentono di eseguire elaborazioni simultanee in un programma scritto in C#.
- Possono essere utilizzati facilmente grazie allo spazio dei nomi System.Threading di .NET Framework.
- Condividono le risorse dell'applicazione.

Threading

- Il threading consente di eseguire elaborazioni simultanee in un programma in C# in modo da poter eseguire più di un'operazione alla volta
- Ad esempio è possibile utilizzare il threading per monitorare l'input dell'utente, eseguire attività in background e gestire flussi di input simultanei.

Thread di lavoro

- Per impostazione predefinita, i programmi Visual Basic o C# comprendono un unico thread. È tuttavia possibile creare e utilizzare thread ausiliari per eseguire il codice in parallelo con il thread primario.
- Questi thread vengono definiti thread di lavoro.



I thread processo di lavoro

- utilizzati per eseguire attività lunghe o in cui il tempo riveste molta importanza, senza bloccare il thread primario.
- utilizzati nelle applicazioni server per soddisfare le richieste in arrivo senza attendere il completamento della richiesta precedente
- utilizzati per eseguire attività in background nelle applicazioni desktop, in modo tale che il thread principale, che gestisce gli elementi dell'interfaccia utente, garantisca tempi di risposta ottimali per le azioni dell'utente.

Namespaces

- La strategia più diffusa consiste nell'utilizzare i thread di lavoro per eseguire attività lunghe o in cui il tempo riveste importanza, ma che non richiedono molte delle risorse utilizzate da altri thread.
- Ovviamente alcune risorse del programma devono essere accessibili a più thread.
- In questi casi lo spazio dei nomi System.Threading fornisce le classi per la sincronizzazione dei thread. Tali classi comprendono Mutex, Monitor, Interlocked, AutoResetEvent e ManualResetEvent.

System.Threading

- Lo spazio dei nomi System.Threading fornisce classi e interfacce che supportano la programmazione multithreading e consentono di eseguire facilmente attività quali la creazione e l'avvio di nuovi thread, la sincronizzazione di più thread, nonché la sospensione e l'interruzione di thread.

Esempio di utilizzo

- Per incorporare il threading nel codice C#, creare una funzione da eseguire all'esterno del thread principale e puntarvi un nuovo oggetto Thread.
- Nell'esempio di codice riportato di seguito viene creato un nuovo thread in un'applicazione C#:

```
System.Threading.Thread newThread;
newThread = new System.Threading.Thread(anObject.AMethod);
```

Esempio

- Nell'esempio di codice riportato di seguito viene avviato un nuovo thread in un'applicazione C#:

```
newThread.Start();
```

- Il multithreading risolve i problemi legati ai tempi di risposta e al multitasking, ma potrebbe anche generare errori di condivisione e sincronizzazione delle risorse, perché i thread vengono interrotti e ripresi senza avviso in base a un meccanismo di pianificazione centrale.

Sincronizzazione di thread

- Uno dei vantaggi associati all'utilizzo di più thread in un'applicazione è che ogni thread viene eseguito in modo asincrono.
- Per le applicazioni Windows, ciò consente di eseguire in background le attività dispendiose in termini di tempo mantenendo su livelli ottimali i tempi di risposta della finestra e dei controlli dell'applicazione.
- Per le applicazioni server, il multithreading offre la possibilità di gestire ogni richiesta in arrivo con un thread differente. In caso contrario, ogni nuova richiesta verrebbe soddisfatta solo dopo il completamento della richiesta precedente.

Asincronia

- La natura asincrona dei thread implica tuttavia che è necessario coordinare l'accesso a risorse quali manipolazione di file, connessioni di rete e memoria.
- In caso contrario, due o più thread potrebbero accedere contemporaneamente alla stessa risorsa, senza che l'uno rilevi le azioni dell'altro.
- Il risultato è un danneggiamento imprevedibile dei dati.

Lock

- La parola chiave **lock** consente di assicurarsi che un blocco di codice venga eseguito fino al completamento senza subire interruzioni da altri thread. Questo risultato si ottiene specificando un blocco a esclusione reciproca per un determinato oggetto per la durata del blocco di codice.
- Un'istruzione lock inizia con la parola chiave lock, cui viene assegnato un oggetto come argomento, seguita da un blocco di codice che dovrà essere eseguito solo da un thread alla volta.

esempio

```
public class TestThreading
{
    private System.Object lockThis = new System.Object();

    public void Function()
    {

        lock (lockThis)
        {
            // Access thread-sensitive resources.
        }
    }

}
```

Ambito del blocco

- L'argomento fornito alla parola chiave lock deve essere un oggetto basato su un tipo di riferimento e viene utilizzato per definire l'ambito del blocco.
- Nell'esempio precedente l'ambito del blocco è limitato a questa funzione, perché all'esterno della stessa non sono presenti riferimenti all'oggetto lockThis. Se tale riferimento fosse presente, l'ambito del blocco si estenderebbe fino a tale oggetto.
- In teoria, l'oggetto fornito a lock viene utilizzato esclusivamente per identificare in modo univoco la risorsa condivisa tra più thread, pertanto può essere un'istanza di classe arbitraria

- Nella pratica, tuttavia, l'oggetto rappresenta in genere la risorsa per cui è necessaria la sincronizzazione dei thread.
- Se ad esempio un oggetto contenitore deve essere utilizzato da più thread, è possibile passare il contenitore a lock per fare in modo che il blocco di codice sincronizzato che segue il blocco acceda al contenitore. Purché altri thread vengano bloccati sullo stesso contenitore prima di accedervi, l'accesso all'oggetto è sincronizzato in modo sicuro.

Monitor

- Analogamente alla parola chiave lock, i monitor impediscono l'esecuzione simultanea di blocchi di codice da parte di più thread. Il metodo Enter consente a un unico thread di procedere nelle seguenti istruzioni. Tutti gli altri thread risultano bloccati finché il thread in esecuzione non chiama Exit. Questo risultato è analogo a quello che si ottiene con la parola chiave lock. In realtà, la parola chiave lock viene implementata con la classe Monitor.

Equivale

```
lock (x)
{
    DoSomething();
}
```

```
System.Object obj = (System.Object)x;
System.Threading.Monitor.Enter(obj);
try
{
    DoSomething();
}
finally
{
    System.Threading.Monitor.Exit(obj);
}
```

lock o monitor

- È in genere preferibile utilizzare la parola chiave lock anziché direttamente la classe Monitor, sia perché lock è più concisa, sia perché lock assicura il rilascio del monitor sottostante, anche se il codice protetto genera un'eccezione.
- A tal fine viene utilizzata la parola chiave finally, che esegue il blocco di codice associato sia che venga o meno generata un'eccezione.

sincronizzazione

- L'utilizzo di una parola chiave lock o monitor risulta utile per impedire l'esecuzione simultanea di blocchi di codice sensibili ai thread, ma questi costrutti non consentono la comunicazione di un evento tra un thread e l'altro.
- Per questa funzione sono necessari gli eventi di sincronizzazione, ossia oggetti caratterizzati da uno di due diversi stati, segnalato e non segnalato, che possono essere utilizzati per attivare e sospendere i thread.
- Per sospendere i thread, è possibile fare in modo che attendano un evento di sincronizzazione con lo stato non segnalato, mentre per attivarli è possibile cambiare lo stato dell'evento in segnalato.
- L'esecuzione di un thread che tenta di attendere un evento già segnalato continua senza ritardi.