CONTATTO

Sergioroselli.it

JAVASCRIPT MULTITHREADING

UNO SGUARDO AGLI WEB WORKER

Come forse saprai, Java script è a thread singolo, questo significa che un singolo thread gestisce il loop degli eventi. Sebbene i thread dedicati abbiano migliorato la reattività delle pagine Web, hanno però fatto in modo che ogni scheda sia incapace di gestire più script in esecuzione contemporaneamente.

Perché dovremmo aver bisogno di più script in esecuzione contemporanea?

I computer sono estremamente veloci e con ogni scheda che utilizza il proprio thread, le pagine Web non dovrebbero avere problemi. Questo è vero per la maggior parte dei casi fino a quando il browser non esegue un algoritmo complesso o rende una visualizzazione complessa. Questi laboriosi processi finiscono per bloccare il thread principale che rallenta gli eventi dell'interfaccia utente. Fortunatamente nel 2009, **JavaScript** ha introdotto un nuovo "giocattolo", il Web **Workers**, per affrontare questo problema.

Che cosa sono gli **Web Worker**?

Gli Web Worker sono script eseguiti da una pagina HTML che viene eseguita in un thread, quindi in background lontano dal thread di esecuzione principale. I dati vengono inviati tra il thread principale ed i secondari tramite messaggi. Poiché i secondari vengono eseguiti su un thread a parte rispetto al thread principale, è possibile utilizzare i secondari per eseguire attività ad alta intensità di processo nel browser Web senza creare istanze di blocco.

È piuttosto semplice generare un **Web Worker** , quindi iniziamo con il creare due file: un file principale ed un file secondario con il codice da eseguire. I due file separati sono necessari affinché il Web Worker venga eseguito come un thread isolato.

Chiamiamo i due file **main.js** e **worker.js** e creiamo un nuovo oggetto Worker assegnando il percorso del file **worker** al costruttore Worker in questo modo:

```
// main.js
var worker = new Worker ('worker.js');
```

Ora che abbiamo un thread, inviamo un messaggio. Per fare ciò dovremo richiamare la funzione postMessage () dall'oggetto worker.

```
// main.js
worker.postMessage ('Buon Compleanno');
```

I dati inviati dal thread principale e secondario verranno copiati e non condivisi, lo si tenga presente quando si pensa all'allocazione della memoria ed alla velocità di trasferimento dei dati.

Ora, affinché il nostro **Thread** secondario possa ascoltare il messaggio "Buon compleanno", dovremo impostare un **listener** di eventi nel file worker.js così:

```
// worker.js
self.addEventListener ('message', function (e) {
  // codice da eseguire
}
```

Una volta che worker.js sente un evento (un messaggio), esegue il codice all'interno del blocco della funzione che invece di restituire una variabile e relativo valore, restituisce un messaggio al thread principale con la funzione **postMessage ()**.

Possiamo leggere i dati del messaggio inviato con **e.data**.

```
// worker.js
  self.addEventListener ('message', function (e) {
  var message = e.data + 'a me stesso!!';
  self.postMessage (message);
}
```

Nel codice sopra, stiamo inviando il messaggio "'Buon Compleanno'" e poi ritorna al thread di esecuzione principale.

In conseguenza, avremo anche bisogno di un listener di eventi messaggio nel file principale per ricevere i dati e agire di conseguenza.

Ecco il file main.js finale:

```
// main.js
var worker = new Worker ('worker.js');
worker.addEventListener ('message', function (e) {
    console.log (e.data);
}
worker.postMessage ('Buon Compleanno');
```

Infine bisogna ricordarsi di chiudere il thread, aggiungendo la funzione

self. close ()

Ecco il worker.js finale:

```
// worker.js

 self.addEventListener ('message', function (e) {
 var message = e.data + 'a me stesso!';
 self.postMessage (message);
 self.close ();
}
```

Analizziamo quello che è appena successo:

1. La nostra applicazione ha creato un **web worker** in main.js che esegue il codice da worker.js
2. Invia al **thread** un messaggio con la stringa "Buon Compleanno"
3. Il thread con il listener di eventi per "messaggio", riceveva il messaggio ed esegue il codice all'interno.
4. Il thread ha aggiunto la stringa "a me stesso!!" ai dati del messaggio ricevuto "Buon Compleanno a me stesso!!" e li invia come dati al main.js
5. Main.js, che comprende a sua volta un listener di eventi per i messaggi stamperà il mesaggio completo con console.log "Buon Compleanno a me stesso!".

Nota

Per motivi di sicurezza Chrome non consente di distribuire i web worker a livello locale. Usa Firefox.