

### JavaScript: buone pratiche per la performance sul DOM

L'attraversamento del DOM (Document Object Model) è una delle operazioni più comuni e fondamentali nello sviluppo web front-end. Tuttavia, un'inefficiente gestione delle operazioni di manipolazione e attraversamento del DOM può portare a problemi di performance, soprattutto in applicazioni complesse. Questo articolo esplorerà le tecniche più efficaci per effettuare un attraversamento del DOM in modo performante utilizzando JavaScript, evitando rallentamenti e migliorando l'esperienza utente.

Effettuare operazioni sul DOM è relativamente lento rispetto ad altre operazioni in JavaScript. Questo accade perché il browser deve ricalcolare il layout della pagina e aggiornare la visualizzazione ogni volta che avviene una modifica. Pertanto, eseguire operazioni frequenti o mal gestite sul DOM può causare ritardi visibili, influenzando negativamente la fluidità dell'applicazione.

## Approcci performanti

### 1. Ridurre gli accessi ripetuti al DOM

Ogni volta che accediamo a un elemento del DOM (ad esempio, utilizzando `document.getElementById()` o `querySelector()`), il browser effettua un'operazione relativamente costosa. Per ridurre il numero di accessi, è buona norma salvare il riferimento agli elementi che vengono utilizzati più volte in una variabile locale.

### 2. Usare selettori specifici

Quando possibile, usa selettori specifici come `getElementById()` o `getElementsByClassName()` anziché il più generico `querySelector()` o `querySelectorAll()`. I selettori specifici sono più veloci, in quanto operano direttamente sugli indici interni del DOM, mentre `querySelector()` utilizza un motore di selezione CSS completo, che è più flessibile ma anche più lento.

### 3. Minimizzare i reflow e i repaint

Ogni volta che manipoliamo il DOM, il browser può dover ricalcolare il layout (reflow) e ridisegnare la pagina (repaint). Questi processi sono costosi, soprattutto se coinvolgono molti elementi. Ecco alcune tecniche per ridurre i reflow e i repaint:

- **Manipolazione fuori dal DOM:** Se devi apportare molte modifiche a un elemento, è meglio farlo al di fuori del DOM attivo e poi reinserirlo. Puoi farlo creando frammenti di documenti (`document.createDocumentFragment()`).
- **Batching:** Raggruppa le modifiche in un singolo aggiornamento del DOM invece di effettuare cambiamenti uno alla volta.
- **Evita il ciclo "misura-modifica":** Se leggi le proprietà di layout come `offsetWidth` o `offsetHeight` e poi modifichi il DOM all'interno di un ciclo, il browser sarà costretto a ricalcolare il layout per ogni iterazione. Evita di intercalare letture e scritture del layout.

### 4. Usare la delegazione degli eventi

Invece di assegnare eventi a numerosi elementi figli, puoi utilizzare la delegazione degli eventi. Questo significa attaccare un singolo event listener a un genitore comune, ascoltando gli eventi in bubbling (propagazione) e gestendoli in modo efficiente.

### 5. Ridurre il ricalcolo degli stili

Operazioni ripetute che forzano il browser a ricalcolare gli stili possono essere costose. Ad esempio, modificare frequentemente la classe CSS di un elemento in un ciclo o manipolare singole proprietà CSS può rallentare l'applicazione. Utilizza proprietà CSS come `classList` per modificare più classi contemporaneamente.

## 6. Lazy loading e virtualizzazione

Se hai un gran numero di elementi (come in liste o tabelle di grandi dimensioni), considera l'uso di tecniche di "lazy loading" o "virtualizzazione", dove solo una piccola porzione di elementi viene effettivamente renderizzata nel DOM, mentre gli altri vengono caricati man mano che l'utente scorre la pagina.

## 7. Richiedere meno ricalcoli forzati

Il reflow si verifica quando forzi il browser a ricalcolare il layout in seguito a una lettura o modifica delle proprietà legate al layout. Cerca di ridurre al minimo queste richieste, accumulando modifiche e applicandole tutte in un solo colpo.

# Conclusione

Ottimizzare l'attraversamento del DOM in JavaScript è fondamentale per mantenere un'esperienza utente fluida e performante, specialmente nelle applicazioni complesse. Le strategie illustrate — ridurre gli accessi al DOM, minimizzare reflow e repaint, usare selettori specifici e delegazione degli eventi — ti permettono di scrivere codice più efficiente e mantenere alta la performance dell'applicazione.

# Applicazioni Correlate

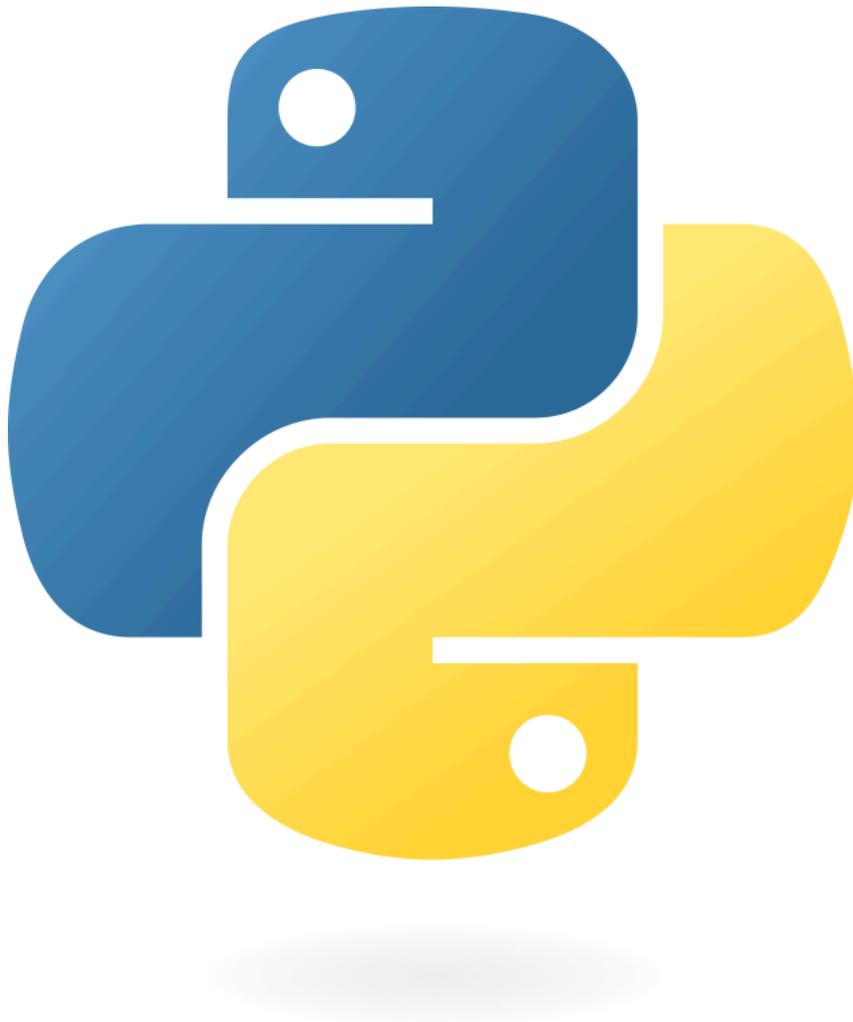


- 

### **JavaScript Password Mask**

Un esempio in JavaScript di mascheramento di una password con l'aggiunta della funzionalità di copia negli appunti.

Docker Docker Compose JavaScript



- 

### **Python Placeholder Image**

Applicazione sviluppata in Python con Flask per la creazione di immagini segnaposto.  
Docker Docker Compose Python Flask JavaScript



- 

### **Go Placeholder Image**

Applicazione in Go per la creazione di immagini segnaposto.  
Docker Docker Compose Go JavaScript