Stato, Logica e Change Detection nell'Era dei Signal

Il concetto di stato

Prima di affrontare la gestione degli eventi e la reattività, è necessario definire con precisione il concetto di stato. Lo stato di un componente è l'insieme dei valori delle sue proprietà in un dato momento, ovvero il suo contenuto informativo in un preciso istante. Lo stato può cambiare nel tempo, e di questo cambiamento si occupa generalmente la logica del componente stesso.

Un componente dotato di logica di calcolo

Il componente HelloWorld visto in precedenza aveva uno stato minimo, rappresentato da una singola variabile sempre uguale a se stessa, e nessuna logica reale. Un esempio più significativo è quello di una pagella, che prevede quattro variabili di stato corrispondenti ai voti delle materie, e tre metodi per calcolare la media delle materie tecniche, quella delle materie umanistiche e la media generale.

Le variabili di stato vengono dichiarate nella classe TypeScript con il proprio tipo e il proprio valore iniziale. I metodi di calcolo operano su questi valori tramite `this`, che in TypeScript, come in JavaScript, è obbligatorio per accedere alle proprietà dell'istanza corrente. Nel template HTML i risultati dei metodi vengono proiettati tramite la sintassi di interpolazione, richiamando i metodi direttamente tra le doppie parentesi graffe.

Un aspetto fondamentale è che non è necessario aggiornare manualmente il DOM dopo aver calcolato i risultati. Angular si occupa automaticamente di aggiornare il template al cambiamento dello stato del componente. Chi proviene dalla programmazione JavaScript tradizionale è abituato alla sequenza evento, calcolo e aggiornamento manuale del DOM elemento per elemento. Nei framework moderni questa preoccupazione è superata: l'interfaccia è concepita come una funzione derivata dello stato, e il framework si occupa di mantenerla sincronizzata.

Aggiunta di interattività e gestione degli eventi

Per rendere la pagella interattiva è necessario permettere all'utente di modificare i voti tramite pulsanti. Questo richiede di intervenire sia sulla logica del componente che sul suo template.

Sul lato TypeScript viene definito un metodo che riceve come parametri il nome della materia e un delta numerico, positivo o negativo, e modifica il voto corrispondente solo se il nuovo valore rimane compreso tra uno e dieci. Sul lato HTML i pulsanti vengono collegati a questo metodo tramite la sintassi dell'event binding, che prevede di racchiudere il nome dell'evento tra parentesi tonde e di associarlo al metodo del componente. La sintassi `(click)="nomeMetodo()"` è l'equivalente angolare del classico `onClick` di JavaScript, con la differenza che non si richiama una funzione della pagina ma un metodo del componente. Angular adotta la convenzione di omettere il prefisso `on` dai nomi degli eventi.

Il problema delle prestazioni con la change detection classica

Sebbene la modifica di un voto aggiorni correttamente le medie nella pagina, emerge un problema di prestazioni rilevante. Quando lo stato del componente cambia, Angular avvia una procedura di aggiornamento globale che ricalcola tutti i metodi del template, inclusi quelli i cui valori non sono effettivamente cambiati. Modificando il voto di italiano, ad esempio, viene rieseguito anche il calcolo della media delle materie tecniche, il cui risultato è rimasto invariato poiché matematica e programmazione non sono state toccate.

Su un progetto didattico di piccole dimensioni questo non ha conseguenze apprezzabili, ma in un'applicazione reale composta da centinaia di componenti questo aggiornamento automatico e indiscriminato può incidere significativamente sulle prestazioni, sul consumo della batteria nei dispositivi mobili e sulla reattività dell'interfaccia. Questa era la best practice fino a circa il 2023, quando la crescita delle applicazioni ha reso necessario un approccio diverso.

I Signal come soluzione

Per rispondere a questa esigenza Angular introduce il concetto di Signal, un sistema di ricalcolo con dipendenze esplicite che sostituisce la change detection globale con una reattività granulare e precisa.

Un Signal è una funzione che avvolge una variabile di qualsiasi tipo e restituisce sempre l'ultimo valore del suo contenuto. Può essere visto come un valore in evoluzione nel tempo, capace di notificare ai soggetti interessati il proprio cambiamento. Quando il valore di un Signal cambia, solo i soggetti che dipendono da quel Signal vengono aggiornati.

Per trasformare le variabili di stato in Signal è sufficiente avvolgerle nella funzione `signal()`, passando il valore iniziale come argomento. Nel template il valore viene letto invocando il Signal come funzione, ad esempio `{{ italian() }}`. Per modificare il valore sottostante si utilizza il metodo `set()`, disponibile sull'oggetto Signal.

I Computed Signal

I risultati derivati dallo stato, come le medie, vengono gestiti tramite i Computed Signal, creati con la funzione `computed()` alla quale si passa una arrow function che definisce il calcolo. Un Computed Signal viene rieseguito esclusivamente quando uno dei Signal da cui dipende cambia valore. Se si modifica il voto di italiano, il Computed Signal della media tecnica non viene ricalcolato, perché non dipende da italiano ma solo da matematica e programmazione.

I Computed Signal hanno due caratteristiche prestazionali importanti. Sono lazy, il che significa che vengono calcolati solo quando vengono esplicitamente letti. Sono inoltre memoized, ovvero il loro valore viene memorizzato e non ricalcolato finché nessuna delle loro dipendenze cambia. Queste due proprietà sono attive per impostazione predefinita e rappresentano un vantaggio significativo nelle applicazioni di medie e grandi dimensioni.

A partire dall'introduzione dei Signal, i metodi di istanza vengono utilizzati quasi esclusivamente per rispondere a eventi specifici, come la modifica di un voto, mentre tutti i calcoli derivati vengono espressi come Computed Signal.

Il metodo update()

Quando il nuovo valore di un Signal dipende dal valore precedente, come nel caso dell'incremento o del decremento di un voto, è preferibile utilizzare il metodo `update()` al posto di `set()`. Il metodo `update()` riceve una arrow function che prende come argomento il valore attuale del Signal e restituisce il nuovo valore.

Questa tecnica previene il rischio di obsolescenza del dato, che si verifica quando il valore viene letto manualmente e poi riscritto modificato: in scenari complessi o asincroni, il valore del Signal potrebbe essere cambiato nell'intervallo di tempo tra la lettura e la scrittura, portando ad aggiornamenti basati su un valore non più attuale. La sintassi con arrow function lavora sempre sull'ultimo valore disponibile, eliminando questo rischio.

I Side Effect con effect()

In alcuni casi è necessario definire risposte automatiche al cambiamento di un Signal che esulino dal calcolo di un valore e dal normale aggiornamento della pagina. Potrebbe trattarsi di una chiamata a una API, di un accesso al localStorage, di un'operazione di logging o di qualsiasi altra attività collaterale. Angular gestisce questi casi tramite la funzione `effect()`, che definisce una reazione al cambiamento di un Signal senza l'obbligo di restituire un valore, a differenza di `computed()`.

Un `effect()` è automaticamente abbonato ai Signal che legge al proprio interno, e viene rieseguito ogni volta che uno di essi cambia. È inoltre legato al ciclo di vita del componente e viene deallocato automaticamente quando il componente viene rimosso. La dichiarazione di un effect avviene nel costruttore del componente, che rappresenta il contesto di iniezione stabile richiesto da Angular per gestirne correttamente il ciclo di vita.

I side effect sono uno strumento potente ma che richiede attenzione: un utilizzo imprudente rischia di introdurre comportamenti difficili da tracciare e di compromettere la stabilità dell'applicazione.