React & Virtual DOM & co.

distillato in poche righe di javascript

Sylvain Jermini <sylvain.jermini@syjer.com> | @sy_jer

Frameworks js: esplosione cambriana

- Ogni mese, una miriade di librerie/frameworks js appaiono
- Chi si ricorda di quando jquery/prototype erano stati rilasciati?
- Di tanto in tanto, ci sono idee ***buone*** che verranno adottate da altri:

Convergenza

Il problema

- Adesso si hanno **applicazioni** web, non soltanto documenti
- Si vuole salvare facilmente lo stato applicativo per resistere al “reload”, gestire ”l’undo”, ...
- Si hanno tanti elementi “attivi” presenti sullo schermo
- Tante operazioni di lettura e scrittura sul DOM

Lentezza del DOM?

Lentezza del DOM?

- globalmente, l’accesso al DOM è veloce
- la lentezza percepita è causata principalmente dal

REFLOW

- (o da scripts scritti male :D)

Reflow is slow

- È l’operazione di ricalcolo delle posizioni e geometrie degli elementi in un documento
- È un operazione **bloccante**
- Cause:
  - aggiunta/rimozione di elementi nel DOM
  - applicazione dinamica di stili
  - lettura di proprietà css “calcolate” (`getComputedStyle()`)

React e il virtual DOM

- Sviluppato da Facebook http://facebook.github.io/react/
- Apparso nel 2013
- “The V in MVC”
- Popolarizzato il concetto di DOM virtuale e tutto quello che ne deriva
- Consiglio fortemente di darci almeno un'occhiata

L’idea

- Aggiungere un “layer” intermedio tra il DOM e il codice dell’applicazione
- Lo stato applicativo è presente in un oggetto al posto di essere disperso nel DOM
- Parte di quel layer offre un interfaccia simile al DOM

Virtual DOM

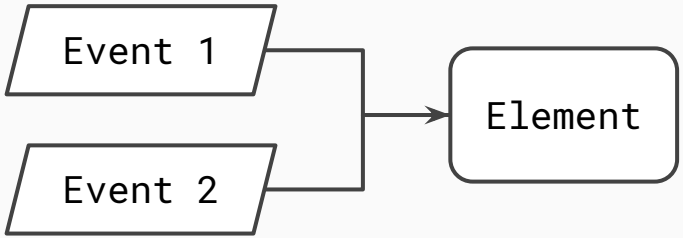
- Le operazioni sul virtual dom non causano reflow

Esempio 1

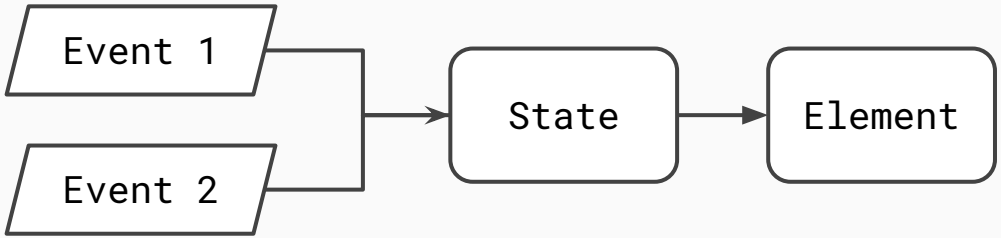
- Esempio base: example-1-basic.html
- Oggetto di stato: example-1-layer.html

Esempio 1: la differenza

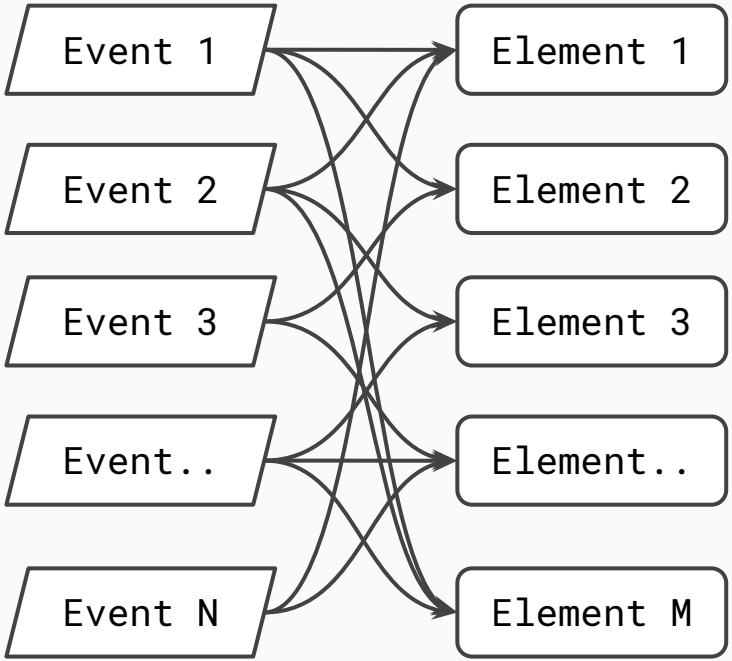
Da:



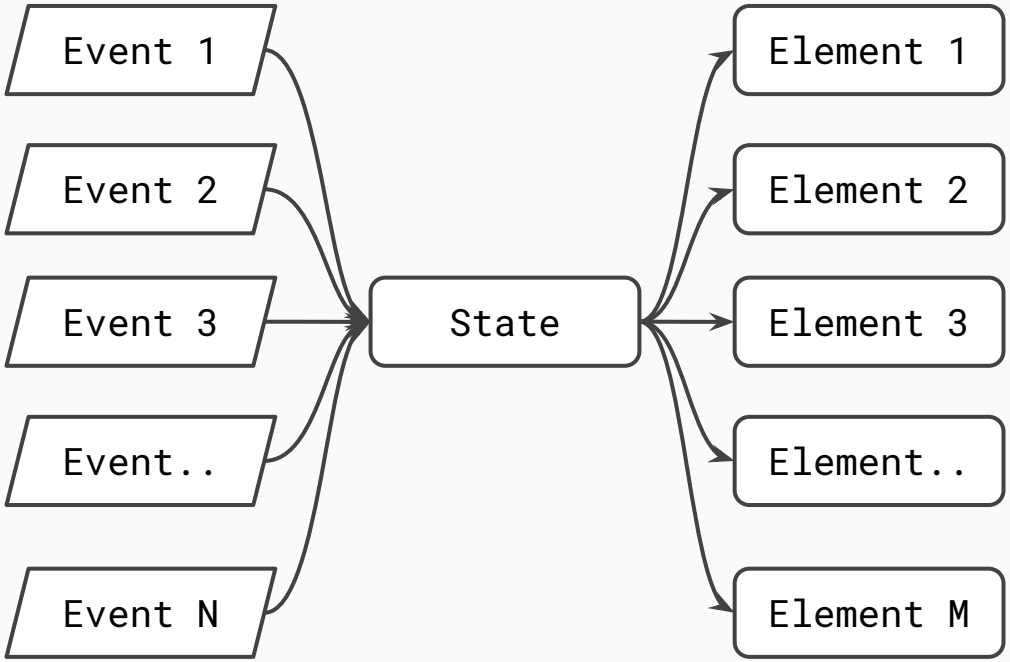
Si ha:



Esempio 1: bisogna pensare grande per vedere i benefici



Esempio 1: bisogna pensare grande per vedere i benefici



Magical features

Con il concetto di stato applicativo descritto da un oggetto, si possono avere abbastanza facilmente le seguenti features:

- save/load/undo “gratis”
- time travel debugging

Save/undo/time traveling debugging

Queste features si basano sul fatto che **tutto** lo stato applicativo è presente in un oggetto che si può salvare o caricare:

```
//save
var savedState = deepCopy(state);

//load
state = loadSavedState();
updateUI();
```

Esempio 2

Resistere al page reload: example-2-save-state.html

Se si può salvare lo stato ad ogni cambiamento, si può salvarlo in una lista:

- undo/redo
- time traveling debugging

Ossia: example-2-list-save-state.html

Esempio 2

Dall'ultimo esempio, si nota che certi "pattern" emergono dal codice:

- l'oggetto di stato "`time`" e la funzione "`updateTimeUI`", molto simile a state e updateUI: due componenti distinti?
- l'UI ad ogni evento viene ricreato attraverso la funzione `render()` e `updateUI()`

Evitare gli sprechi grazie al virtual DOM

Ricreare la UI ad ogni evento è pesante per il browser.

Se fosse possibile, dato lo stato corrente del DOM e il nuovo stato generato dalla funzione `render()`, calcolare unicamente la

`DIFFerenza` da applicare?

Virtual DOM: diffing is the key

```
var root = document.getElementById('ui');
var prevState = state, prevTree = [];

function render(state) {
    // Virtual DOM is really just a tree of
    // JavaScript objects or arrays
    return [
        ['span', {id: 'count'},state.items.length],
        ['ul', {}, state.items.map(function (item) {
            return  ['li', {}, item]
        })]
    ];
}
```

```
function updateUI() {
    var vTree = render(state);
    // Just a diff on data structures
    var diff = vDiff(prevTree, vTree);
    // Apply series of patches to real DOM
    vApply(root, diff)

    prevState = deepCopy(state);
    prevTree = vTree;
}
```

Ultimo passo logico

Negli esempi, la funzione di deepCopy è usata dappertutto. Copiare in modo profondo gli oggetti è abbastanza costoso, soprattutto con modelli grossi.

Per quello l'uso di **strutture di dati immutabili** sono molto alla moda: permettono la condivisione di tutto lo stato che non cambia. Calcolare la differenza è un semplice paragone di puntatori!

Esempio 3

Vediamo se riusciamo a mappare i concetti con un esempio react:

example-3-react.html

Virtual DOM: chi lo usa? Altri vantaggi?

- react
- ember.js con il nuovo motore di rendering “glimmer” http://emberjs.com/
- tutte le librerie che si appoggiano a https://github.com/Matt-Esch/virtual-dom/wiki
- mithril: https://lhorie.github.io/mithril/
- cycle.js: http://cycle.js.org/
- ractivejs: http://www.ractivejs.org/

Virtual DOM = Disacoppiato dal DOM

E per questo ci sono iniziative molto interessanti, per esempio usare react per fare applicazioni native attraverso react native: https://facebook.github.io/react-native/

Una nota a proposito di angularjs

La versione 2 non userà il virtual dom, ma non è certamente l'unica soluzione per avere buone performances! Vedi http://victorsavkin.com/post/110170125256/change-detection-in-angular-2

That’s all

Domande?

Risorse:

- http://facebook.github.io/react/
- http://hackflow.com/blog/2015/03/08/boiling-react-down-to-few-lines-in-jquery/
- https://github.com/patrick-steele-idem/morphdom/
- http://blog.reverberate.org/2014/02/react-demystified.html
- https://developers.google.com/speed/articles/reflow
- https://github.com/facebook/immutable-js