Appunti su Javascript: Temi Trasversali (Parte II)

Modularizzazione in Javascript

Concetti Chiave

- **Modularizzazione:** Suddivisione del codice di un programma complesso in frammenti indipendenti e intercambiabili.
- **Benefici:** Separation of concern, incapsulamento, intercambiabilità del codice, load on demand.
- **Struttura di un Modulo:**
  - File separato.
  - Variabili:
    - Locali alla funzione (ambito usuale).
    - Locali al modulo (globali nel modulo, ma invisibili all'esterno).
    - Esportate dal modulo (visibili a chi include il modulo).
  - Gestione della *namespace pollution* (conflitti tra nomi di variabili/funzioni in moduli diversi).

Evoluzione della Modularizzazione in Javascript

1. **Inizio:** Nessun meccanismo di modularizzazione nativo.
2. **Successivamente:** Introduzione di prototipi, closure, IIFE (Immediately Invoked Function Expression), classi.
3. **Standardizzazione:**
   - **CommonJS (e AMD):** Prima proposta (2009), adottata da NodeJS. Non funziona nativamente sui browser (AMD è una versione parzialmente compatibile). Usa `module.exports` e `require`.
   - **ES Modules (ECMAScript Modules):** Standard WHATWG, supportato dai browser moderni (dal 2016). Usa `export` e `import`.

CommonJS e AMD

- **`module.exports`:** Specifica cosa esportare dal modulo.
- **`require('./path/to/module.js')`:** Importa un modulo.
- **Esempio**

```
// keys.js (modulo)
var keys = { /* ... */ };
var Utils = { /* ... */ };
module.exports = Utils; //keys.js

// main.js (utilizzo del modulo)
var keys = require('./keys.js'); //main.js
// ... usa keys ...
```

ES Modules (1)

- **`export`:** Specifica cosa esportare dal modulo.
- **`import { name1, name2 } from './module.js';`:** Importa specifici nomi da un modulo.
- **`import * as myModule from './module.js';`:** Importa tutto il modulo in un oggetto (namespace).
- **Esempio**

```
// keys.js (modulo)
var keys = { /* ... */ };
export function name(keyCode) { /* ... */ }  //keys.js
export function keyCode(name) { /* ... */ }   //keys.js

// main.js (utilizzo del modulo)
import { name, keyCode } from './keys.js';   //main.js
// ... oppure ...
import * as keys from './keys.js';          //main.js
// ... usa keys.name, keys.keyCode ...
```

ES Modules (2) - Esportazione Predefinita

- **`export default myValue;`:** Esporta un singolo valore (oggetto, funzione, variabile, ecc.) come esportazione predefinita.
- **`import myNewName from './module.js';`:** Importa l'esportazione predefinita, dandole un nome arbitrario.
- **Esempio:**

```
  // keys.js (modulo)
var keys = { /* ... */ };
function name(keyCode) { /* ... */ }
export default name; //keys.js

// main.js
import newName from './keys.js'; //main.js
```

ES Modules (3) - Utilizzo nel Browser

- **`<script type="module" src="module.js"></script>`:** Indica al browser che si sta caricando un modulo ES, e non uno script tradizionale. Senza `type="module"`, il browser genererebbe un errore.

Moduli in Node.js

- Node.js è fortemente modulare.
- Un modulo è un file JavaScript.
- I moduli vengono cercati localmente o globalmente.
- Caricamento con `require()` (CommonJS) o `import` (ES Modules).
- **Struttura del caricamento:**
  1. Modulo core built-in (parte di Node.js).
  2. Dipendenza esterna (installata con `npm`).
  3. Script locale.

Creare un Modulo (Esempi)

- **CommonJS**

```
// greetings.js
  hello = function() { console.log("Hello!\n") }
  ciao = function() { console.log("Ciao!\n") }
```

```
module.exports.hello = hello;
module.exports.ciao = ciao;
```

- **ES Modules**

```
// greetings.js
hello = function() { console.log("Hello!\n") }
ciao = function() {  console.log("Ciao!\n") }

greetings = {hello, ciao}
export default greetings;
```

Interpolazione e Template HTML

Introduzione

- I siti web dinamici generano HTML programmaticamente.
- Template: pagina HTML "vuota" (grafica e struttura) con *placeholder* che verranno sostituiti con valori calcolati.
- Placeholder: sintassi speciale per distinguersi dal contenuto statico.

Problemi con la Generazione di HTML

- Stringhe lunghe e multilinea.
- Conflitti di caratteri (es. virgolette in HTML e come delimitatori di stringhe nel linguaggio di programmazione).
- Necessità di interpolazione di variabili (inserimento di valori dinamici nel template).

Soluzioni per l'Interpolazione

1. **Virgolette Annidate:** Problematico per la leggibilità e la gestione degli errori.

```
var a ="<p class='"+obj.class+"'>"+obj.testo+"</p>"
```

2. **Funzioni di Formattazione:** `sprintf` (C/C++), `.format()` (Python), interpolazione automatica in PHP (con doppi apici).

```
//Esempio Python
v = "La sommadi {0} e {1} vale {2}".format(a, b, a+b)
```

3. **Heredoc:** Sintassi per stringhe multilinea. In PHP, interpolazione automatica. In Python, usare `.format()`.

```
//Esempio Python
V = """\
<html>
<body>
  <h1>Hello {name}!</h1>
</body>
</html>""".format(name="John")
```

4. **Template Server-Side (es. Pug):** Motori di template che elaborano un template e dati, generando l'HTML finale.

```
//Esempio con Express.js e Pug
app.set('view engine', 'pug')
app.get('/', function (req, res){
  res.render('index.pug', {
    title: 'Ciao a tutti',
    message: 'Prima prova!'
  })
})
```

5. **Nascondere i Template (Soluzione "Brutta"):** Inserire il template nell'HTML, ma nascosto (es. con CSS `display: none`). Problemi di accessibilità, SEO, caricamento di risorse inutili.

6. **Tag `<template>` (HTML Living Standard):**

   - Contenuto inerte (non renderizzato, script non eseguiti, immagini non caricate).
   - Attivazione: importazione del contenuto nel DOM con `document.importNode()`.

```
<template id="tpl">
  <p>Testo</p>
  <img src=""/>
</template>

<script>
let tpl = document.getElementById('tpl').content ;
let img = tpl.getElementsByTagName('IMG')[0] ;
img.src = 'fig1.gif' ;
let newContent = document.importNode(tpl, true) ; // Importa in profondità
document.getElementById('output').appendChild(newContent) ;
</script>
```

7. **Interpolazione "dei poveri" (Funzione Personalizzata):** Creare una funzione `tpl()` che usa `replace()` per sostituire placeholder.

```
String.prototype.tpl= function(o) {
  var r = this ;
  for (var i in o) {
    r = r.replace(new RegExp("\\$"+i, 'g'),o[i])
  }
  return r
}

// Uso con un oggetto:
var t = "<p>Io sono il $titolo $nome $cognome.</p>";
var o = {titolo: "prof.", nome: "Fabio", cognome: "Vitali"} ;
var r = t.tpl(o);
```

8. **Template Literals (Backticks):**

- Delimitatori: backtick (`` ` ``).
- Stringhe multilinea.
- Interpolazione: `${variable}`.
- **Importante:** L'interpolazione avviene *solo* al momento dell'assegnazione del template literal.

```
var firstName = 'Jane';
var x = `Hello ${firstName}!
How are you
today?`;
```

9. **Combinazione di `<template>` e `String.prototype.tpl()`:** Approccio comodo per template con placeholder.
10. **Framework per Template Client-Side (Mustache.js, Handlebars.js, Angular, React, Vue):** Librerie che forniscono meccanismi avanzati di templating e data binding.
    - **Mustache.js:** Semplice, logic-less (nessuna logica nel template).
    - **Handlebars.js:** Estensione di Mustache, permette accesso a oggetti annidati e blocchi contestuali.
    - **Angular, React, Vue:** Framework più complessi che usano il concetto di *componenti* (vedi sezione successiva).

Componenti

Concetto di Componente

- Insieme di codice (HTML, CSS, JavaScript) autonomo, autosufficiente e incapsulato.
- Le applicazioni web moderne sono costruite come composizioni di componenti.
- Un componente gestisce:
  - Visualizzazione (HTML + CSS).
  - Interazione con l'utente (eventi, callback).
  - Interazione con altri componenti e con l'applicazione (scambio dati, manipolazione del DOM).

Termini Comuni nei Framework a Componenti

- **Custom Element:** Estensioni di HTML (elementi o attributi) che inseriscono un componente nel DOM.
- **Template:** Frammento HTML sostituito al custom element in fase di rendering.
- **Virtual DOM / Shadow DOM:** Copia nascosta del DOM usata per il rendering, migliorando le performance.
- **Slot e Interpolazione:** Punti di collegamento tra componenti (slot) e visualizzazione di dati (interpolazione).
- **Binding Monodirezionale (One-Way) e Bidirezionale (Two-Way):** Collegamento automatico tra dati e visualizzazione.
  - **Monodirezionale:** Aggiornamento automatico della vista quando il modello cambia.
  - **Bidirezionale:** Aggiornamento automatico della vista *e* del modello quando uno dei due cambia.

Routing

Definizione

- Assegnazione di un URI diverso a ogni stato dell'applicazione web.
- **Semplice con LAMP:** Ogni pagina HTML/script ha un URI (file system routing).
- **Complesso con Ajax e Node.js:** Necessità di gestire il routing esplicitamente.

Tipi di Routing

1. **Routing Server-Side:**
    - Il browser si aspetta una pagina HTML completa.
    - L'application logic server-side decide il contenuto da inviare.
    - Fattori decisionali:
        - Metodo HTTP (GET, POST, PUT, DELETE).
        - URI (protocollo, dominio, path, query, fragment).
        - Header della richiesta (non di competenza del router).
    - **Esempio con Express.js:**

```
// file routes.js
var express = require('express');
var router = express.Router();
router.get('/', function(req, res){
  res.send('<p>This is the home page</p>');
});
router.get('/about', function(req, res){
  res.send('<p>This is the about page</p>');
});

module.exports = router;
```

2. **Routing Client-Side:**
    - Non si usa la navigazione server-side per cambiare contenuto.
    - **Single Page Application (SPA):** Unico documento HTML, il contenuto è generato da JavaScript.
    - **Problemi con SPA:** L'URI non cambia, impatti su SEO, bookmark, back/forward, ecc.
    - **Approcci:**
        - **Client-Side Rendering (CSR):** HTML iniziale vuoto/generico, contenuto caricato via XHR.
        - **Server-Side Rendering (SSR):** Stato mantenuto sul server, HTML generato dal server a ogni richiesta.
        - **Static Site Generators (SSG):** Stato sul server, pagine HTML statiche generate in fase di compilazione.
    - **Gestione di `location` e `history`:**

        - `window.location`: Informazioni sull'URI corrente.
        - `window.history`: Stack degli URI visitati.
        - Necessario manipolarli esplicitamente per avere routing funzionante in una SPA.
    - **Routing "Fatto a Mano" (Esempio):**

```
//HTML
<nav><ul>
  <li><a href="#" onclick="goto('home')">Home</a></li>
</ul></nav>
<main id="content"> </main>

//Javascript
let routes = {
  home:     `<h1>Welcome to my home page</h1>`,
  about:    `<h1>About this web site</h1>`,
  contacts: `<h1>Contacts</h1>`
}
const content = document,getElementById('content');

function goto(destination) {
  content.innerHTML = routes[destination];
  let newUri = "/" + destination ;
  window.history.pushState( {path=destination}, "", newUri );
}
```

    - **Routing con Framework (Angular, React, Vue):** Librerie dedicate (es. `RouterModule` in Angular, `react-router-dom` in React, `vue-router` in Vue).

Routing Dinamico (Parametrico)

- Gestione di URI con parametri (es. `/users/:id`).
- Esempi con Express, Angular, React, Vue.

Binding

Definizione

- Collegamento tra dati usati in parti diverse di un programma.
- Modifiche a un dato si propagano automaticamente agli altri dati collegati.
- **Tipi:**
    - **Monodirezionale (One-Way):** Una locazione primaria del dato, le altre sono secondarie (solo lettura).
    - **Bidirezionale (Two-Way):** Tutte le locazioni sono modificabili, le modifiche si propagano ovunque.
- **Nota:** Il metodo `.bind()` delle funzioni JavaScript *non* è correlato al data binding.

Model-View-Controller (MVC) e Model-View-ViewModel (MVVM)

- Pattern di programmazione per la progettazione delle interfacce.
- **MVC:**
    - **Modello:** Natura concettuale del dato.

    - **Vista:** Widget che mostra il dato.
    - **Controller:** Allinea modello e vista, gestisce l'interazione.
- **MVVM:**
    - **Modello:** Come in MVC.
    - **Vista:** Come in MVC.
    - **ViewModel:** Collega (binding) elementi della vista a strutture del modello.
- **Esempio:** Applicazione per visualizzare fatture (tabella e grafico).

Implementazioni del Data Binding

- **Fatto a Mano:** Uso di eventi (es. `onchange`, `input`) e manipolazione del DOM.
- **Angular.js:** Binding bidirezionale con direttive (es. `ng-model`).
- **Angular:** Uso di property binding (`[ ]`) e event binding (`( )`).
- **React:** Stato gestito con `useState`, propagazione manuale delle modifiche (tipicamente monodirezionale, ma si può implementare il bidirezionale).
- **Mutation Observer:** API del browser per osservare modifiche al DOM. Utile per implementare binding personalizzati.
- Reattività: Meccanismi per gestire in modo efficiente il binding quando ci sono molte dipendenze.
- **Signal:** Un tipo di dato reattivo, gestisce automaticamente la propagazione delle modifiche.

Esercizi Proposti (link forniti nel PDF)

1. Calcolatrice
2. Annotazione testuale di frasi (UmmaGramma)
