Programmazione Web - Classi, AJAX e JSON

5.1 - Classi in JavaScript

Evoluzione delle Classi in JavaScript

JavaScript ha avuto un'evoluzione interessante nel modo in cui gestisce il concetto di classe. Originariamente, JavaScript non aveva vere e proprie classi come i linguaggi di programmazione orientati agli oggetti tradizionali. Le cosiddette "classi" in JavaScript erano in realtà funzioni, e questo rifletteva la natura particolare di JavaScript dove funzioni e oggetti sono essenzialmente la stessa cosa.

Prima dell'ES6 (ECMAScript 2015), le "classi" venivano create utilizzando funzioni constructor. Vediamo un esempio:

```
function pippo() {
  this.a = "a";
}
pippo.b = 2;
var pluto = new pippo();
```

In questo esempio, possiamo considerare `a` come un campo di istanza dell'oggetto e `b` come un campo statico della "classe" pippo. Anche se non stiamo usando una vera e propria classe, il comportamento è simile a quello che ci aspetteremmo da un linguaggio con classi tradizionali.

Classi Moderne in JavaScript (ES6)

Con l'introduzione di ECMAScript 6 nel 2015, JavaScript ha finalmente introdotto una sintassi per le classi più familiare agli sviluppatori provenienti da altri linguaggi. Ecco come appare la stessa classe usando la nuova sintassi:

```
class pippo {
  constructor() {
    this.a = "a";
  }
  a = "";
  static b = 0;
  static {
    this.b = 2
  }
}

var pluto = new pippo();
```

La struttura generale di una classe moderna in JavaScript è la seguente:

```
class nomeClasse {
  constructor(parametri) { /* ... */ }
  // campi/metodi di istanza
  static campo/metodo // di classe
  static { /* inizializzatore della classe */ }
}
```

Limitazioni e Caratteristiche delle Classi JavaScript

Le classi JavaScript hanno alcune caratteristiche e limitazioni importanti:

1. **Un solo costruttore**: A differenza di linguaggi come Java, JavaScript permette un solo costruttore per classe. Per simulare costruttori multipli, si utilizzano oggetti di configurazione passati come parametri al costruttore.
2. **Comportamento dinamico**: Le classi JavaScript mantengono il comportamento dinamico tipico del linguaggio. I metodi (incluso il costruttore) possono aggiungere campi e metodi "al volo", e questo è possibile anche dall'esterno della classe.
3. **Campi e metodi privati**: ES6 ha introdotto i campi e metodi privati, che devono essere preceduti dal carattere `#`. Non esistono parole chiave come "private".

```
constructor() {
  this.#a = "a";
}
#a = "";
get() {
  return this.#a;
}
```

Ereditarietà in JavaScript

ES6 ha introdotto anche un meccanismo di ereditarietà più simile a quello dei linguaggi tradizionali:

```
class pippone extends pippo {
  constructor(v) {
    super();
    this.#n = v;
  }

  #n;
  get() {
    return super.get() + " " + this.#n;
  }
}

var pluto = new pippone("Bob");
```

Alcune caratteristiche dell'ereditarietà in JavaScript:

- I metodi e campi pubblici vengono ereditati dalla sottoclasse
- I metodi possono essere ridefiniti (overriding)
- Per accedere ai metodi ereditati e nascosti dall'overriding si usa `super.metodo(parametri)`
- I campi privati non sono visibili alle sottoclassi
- I campi pubblici vengono ridefiniti, quindi non è possibile usare `super.campo`

Confronto con il Vecchio Modello

Rispetto al vecchio modello basato su funzioni, le classi moderne di JavaScript offrono:

- Una gestione degli oggetti che rimane dinamica
- Un certo grado di information hiding grazie ai campi privati

- Una separazione tra oggetti e funzioni: un oggetto definito su una classe è solo un oggetto e non anche una funzione
- La perdita del dualismo oggetto/funzione che caratterizzava il vecchio modello (che rimane comunque valido)

5.2 - AJAX

Evoluzione delle Applicazioni Web Dinamiche

Le prime applicazioni web dinamiche erano piuttosto scomode da utilizzare. Tutto veniva elaborato dal server, quindi ogni azione dell'utente richiedeva di attendere l'arrivo della pagina rigenerata. Con l'introduzione di AJAX, il codice JavaScript associato alla pagina può comunicare direttamente con il server e aggiornare le informazioni nella pagina in modo trasparente per l'utente.

L'Oggetto XMLHttpRequest

Per consentire al codice JavaScript di comunicare con il server, è stato aggiunto all'interprete JavaScript un oggetto specifico: XMLHttpRequest.

AJAX: Asynchronous JavaScript And XML

- **Asynchronous**: Il client invia una richiesta al server, ma la risposta arriva in modo asincrono. In attesa della risposta, il client può continuare a lavorare.
- **JavaScript**: Estende le capacità di JavaScript per creare vere applicazioni client-server. L'oggetto Window è stato arricchito con funzionalità specifiche per gestire la comunicazione asincrona.
- **XML**: I messaggi inviati dal server sono in formato XML, non pagine web complete ma contenuti che il client JavaScript deve integrare nella pagina HTML esistente.

Differenza tra HTML Tradizionale e AJAX

Nel modello tradizionale, l'utente deve cliccare, attendere e vedere l'intera pagina aggiornata. Con AJAX, l'esperienza utente è più fluida: mentre l'interfaccia rimane reattiva, i dati vengono aggiornati in background.

Funzionamento dell'Oggetto XMLHttpRequest

L'oggetto XMLHttpRequest gestisce la comunicazione con il server, l'analisi del documento XML ricevuto e la costruzione della struttura dati DOM. Ecco come viene utilizzato:

1. Si crea una nuova istanza con `new XMLHttpRequest()`
2. Si impostano proprietà e metodi
3. Si chiama il metodo `open()` per aprire la connessione
4. Si chiama il metodo `send()` per inviare la richiesta

Lo schema della comunicazione è il seguente:

1. Lo script invia una richiesta al server tramite l'oggetto XMLHttpRequest
2. L'oggetto XMLHttpRequest invia la richiesta HTTP e attende la risposta
3. Quando la risposta arriva, l'oggetto chiama una funzione dello script detta "callback", fornendo il messaggio ricevuto

Tipo di Comunicazione

La comunicazione può essere:

- **Sincrona**: l'oggetto XMLHttpRequest invia la richiesta HTTP e attende la risposta (lo script è bloccato) - DEPRECATA
- **Asincrona**: l'oggetto XMLHttpRequest opera su un thread parallelo; l'interprete JavaScript non si blocca e lo script continua a lavorare

Gestione della Risposta

L'oggetto XMLHttpRequest riceve la risposta HTTP e analizza il contenuto. Se questo è un documento XML, costruisce l'albero DOM (campo `responseXML`). In ogni caso, lo tratta anche come testo (campo `responseText`).

Stato della Comunicazione

Il campo `readyState` indica lo stato della comunicazione. I valori possibili sono:

- 0 (Uninitialized): Oggetto non inizializzato (metodo `open` non invocato)
- 1 (Open): Metodo `open` invocato, ma metodo `send` non invocato
- 2 (Sent): Metodo `send` invocato, ma i campi `responseText` e `responseXML` non hanno valore
- 3 (Receiving): Alcuni dati ricevuti, ma i campi `responseText` e `responseXML` non hanno valore
- 4 (Loaded): Dati ricevuti, i campi `responseText` e `responseXML` hanno valore

Quando il valore di `readyState` cambia, l'oggetto XMLHttpRequest chiama il metodo `onreadystatechange`, cioè la funzione di callback interna.

Principali Campi e Metodi

- **onreadystatechange**: la funzione da chiamare quando `readyState` cambia
- **status**: il codice HTTP (200 = OK)
- **statusText**: il testo associato al codice HTTP (per 404, "Not Found")

Metodi principali:

- **open(requestMethod, url, asynchronousFlag, username, password)**: Apre la connessione
- **send(bodyContent)**: Invia la richiesta
- **abort()**: Interrompe la comunicazione
- **getAllResponseHeaders()**: Restituisce tutte le intestazioni della risposta HTTP
- **getResponseHeader(headerField)**: Restituisce un'intestazione specifica
- **setRequestHeader(headerField, headerValue)**: Imposta un'intestazione della richiesta
- **overrideMimeType(mimeType)**: Sostituisce il MIME type del contenuto

La Funzione makeAjaxRequest

Tipicamente, non si lavora direttamente al livello dell'oggetto XMLHttpRequest, ma si utilizzano funzioni di più alto livello. Una di queste è `makeAjaxRequest` , che semplifica l'impostazione delle chiamate AJAX. Tuttavia, questa funzione ha limitazioni e non è raccomandata per uso professionale perché:

- Non è robusta
- Non gestisce le chiamate cross-domain
- Non verifica automaticamente la correttezza dei contenuti ricevuti (rischio di JavaScript Injection)

5.3 - JSON

AJAX Senza XML

Gestire un documento XML, anche quando è già disponibile nella rappresentazione DOM, può essere complicato sia lato client che lato server. Per questo motivo, molti programmatori hanno iniziato a utilizzare un formato alternativo: JSON (JavaScript Object Notation).

Caratteristiche di JSON

JSON deriva dalla terza forma per creare oggetti in JavaScript, ma con alcune limitazioni:

- Non consente di specificare metodi

- Un documento JSON contiene solo dati

Il vantaggio principale di JSON è la facilità con cui si può convertire un documento JSON in un oggetto JavaScript.

Formato JSON

Un documento JSON segue queste regole:

- I nomi dei campi sono racchiusi tra virgolette (non apici singoli)
- I campi semplici possono essere numeri o stringhe (con doppi apici o apici singoli)
- I campi complessi possono essere oggetti annidati
- Gli array possono contenere valori semplici e/o oggetti

Esempio di oggetto JSON:

```
{
  "bindings": [
    {
      "ircEvent": "PRIVMSG",
      "method": "newURI",
      "regex": "http://.*"
    },
    {
      "ircEvent": "PRIVMSG",
      "method": "deleteURI",
      "regex": "delete.*"
    }
  ]
}
```

Serializzazione e Deserializzazione

Una stringa JSON è una versione "serializzata" di un oggetto JavaScript. Per convertire una stringa JSON in un oggetto JavaScript (deserializzazione), si può usare la funzione `eval` di JavaScript:

```
var myObject = eval('(' + myJSONtext + ')');
```

Tuttavia, l'uso di `eval` comporta rischi per la sicurezza. È preferibile utilizzare la libreria JSONparser, che fornisce due funzioni più sicure:

- `JSON.parse(JSONtext)` : converte una stringa JSON in un oggetto JavaScript
- `JSON.stringify(object)` : converte un oggetto JavaScript in una stringa JSON

```
var myObject = JSON.parse(myJSONtext);
var myText = JSON.stringify(myJSONObject);
```

Queste funzioni garantiscono una maggiore sicurezza e affidabilità nella manipolazione dei dati JSON.