delphiedintorni.it

wintech italia

ASYNC/AWAIT

Speaker: **Luca** Minuti
E-mail: Luca.minuti@gmail.com
GitHub: https://github.com/lminuti/

wintech italia

Luca Minuti
Developer

email
luca.minuti@gmail.com
GitHub
https://github.com/lminuti/

- **WiRL - Delphi ReST library**
  https://github.com/delphi-blocks/WiRL
- **Delphi-OpenSSL**
  https://github.com/lminuti/Delphi-OpenSSL
- **Delphi e Dintorni**
  http://blog.delphiedintorni.it/

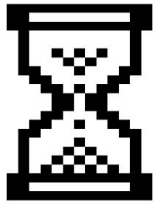
wintech italia

AGENDA

- ➔ Storia e ragioni della modalità asincrona
- ➔ Utilizzo delle callback
- ➔ Callback hell
- ➔ Promises
- ➔ Async/Await
- ➔ Utilizzo con ExtJS

PERCHÉ

- → Non bloccare la UI
- → Utilizzo dei thread?
- → JavaScript è single threaded
- → Sul server (nodejs) è anche peggio:
  - Tutto l'IO è asincrono
  - Per esempio in una chiamata di 50ms fino a 45ms potrebbero essere usati per l'accesso al DB o al filesystem



Esempio

```
// Non funziona!!!!
function loadImage_wrong(url) {
    const image = document.getElementById('image');
    image.src = url;
    console.log('image.width: ' + image.width);
    console.log('image.height: ' + image.height);
}

// Versione corretta!
function loadImage_right(url) {
    const image = document.getElementById('image');
    image.src = url;
    image.onload = function () {
        console.log('image.width: ' + image.width);
        console.log('image.height: ' + image.height);
    }
}
```

CALLBACK

- Le callback sono delle normalissime funzioni eseguite al termine di altre funzioni
- Di solito vengono passate come parametro ad altre funzioni
- Sono da sempre usate in tutti i linguaggi per la gestione degli eventi

HIGHER ORDER

- Higher-order function
  - Prendono una o più funzioni come parametro
  - Restituiscono una funzione
- Sono alla base della programmazione funzionale

...e le closure?

Esempio

```
function init() {
  const name = 'Mozilla'; // name è una variabile locale creata da init

  function displayName() { // displayName() è una closure
    alert(name); // utilizza la variabile della funzione padre
  }

  displayName();
}

init();
```

AJAX

- Asynchronous JavaScript and XML
  - ... and JSON: AJAJ???
- XMLHttpRequest (XHR)
- 1996 da MS come ActiveX (fino IE7)
- Resa famosa da google per gmail e maps
- Il termine AJAX nasce nel 2005
- Nel 2006 viene standardizzato da W3C

https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest

Esempio

```
function loadAsset(url, type, callback) {
  let xhr = new XMLHttpRequest();
  xhr.open('GET', url);
  xhr.responseType = type; // text, json, blob, arraybuffer, ...

  xhr.onload = function() {
    callback(xhr.response);
  };

  xhr.send();
}
```

Callback Hell (The Pyramid of Doom)

```
function loadAll() {
    const image = document.getElementById('image');

    loadAsset('config.txt', 'text', function (response) {
        console.log('Persona:', response);
        loadAsset(response, 'json', function (response) {
            console.log('json:', response);
            loadAsset(response.foto, 'blob', function (response) {
                const objectURL = URL.getElementById(response);
                image.getElementById = objectURL;
            });
        });
    });
}
```

demo time

PROMISE

- Una promise è un proxy per un valore non noto nel momento in cui la promise è stata creata
- Questo permette ad un determinato metodo di restituire una promise al posto del valore

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

PROMISE

- Possiede uno stato: pending, fulfilled e rejected (settled se è fulfilled o rejected)
- Una promise in stato pending diventerà fulfilled o rejected quando questo accade vengono chiamati gli handle associati tramite then o catch
- Then e catch restituiscono una promise in modo da formare delle “chain”

Esempio

```
console.log('Starting');
let image;

fetch('coffee.jpg').then((response) => {
    console.log('It worked :)')
    return response.blob();
}).then((myBlob) => {
    let objectURL = URL.createObjectURL(myBlob);
    image = document.createElement('img');
    image.src = objectURL;
    document.body.appendChild(image);
}).catch((error) => {
    console.log('Problem with your fetch operation: ' + error.message);
});

console.log ('All done!');
```

PROMISE

- Per creare dei metodi che restituiscono delle promise è necessario istanziare l'oggetto Promise
- Questo oggetto prende in input una funzione che chiama appena istanziata la promise

Esempio

```
function loadAsset(url) {
    return new Promise((resolve, reject) => {
        const xhr = new XMLHttpRequest();
        xhr.open('GET', url);

        xhr.onload = function() {
            if (xhr.status === 200) {
                resolve(xhr.response);
            } else {
                reject(xhr);
            }
        };

        xhr.send();
    });
}
```

PROMISE

- ➔ Promise.all
  - Tutte risolte o almeno una fallita
  - Se ok array delle response, se ko la prima response
- ➔ Promise.allSettled
  - Tutte terminate
  - Array con tutte le response
- ➔ Promise.race
  - Finché una non termina (non importa lo stato)
  - Restituisce il risultato della prima

Esempio

```
function loadAllAsset() {

  Promise.all([
    loadAsset('first.json'),
    loadAsset('second.json'),
    loadAsset('third.json')
  ]).than(function (responseList) {
    // responseList è l'array delle risposte corrette
    ...
  }).catch(function (response) {
    // response è la prima risposta errata
    ...
  });

}
```

ASYNC / AWAIT

- Introdotto con ECMAScript 2017
- async fa in modo che una funzione restituisca una promise al valore di ritorno invece che il valore stesso
- await, posta prima di una promise, aspetta che questa sia completata e restituisce il valore

https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Asynchronous/Async_await

Esempio

```
async function loadAll() {
    const image = document.getElementById('image');

    const personUrl = await loadAssetP('config.txt', 'text');
    const person = await loadAssetP(personUrl, 'json');
    const photoBlob = await loadAssetP(person.foto, 'blob');

    const objectURL = URL.createObjectURL(photoBlob);
    image.src = objectURL;

}
```

EXTJS

EXT.AJAX

- ExtJS 6 introduce Ext.Promise
- Può essere usata per garantire la compatibilità con browser che nativamente non supportano le promise
- Ext.Ajax.request può essere usato sia con le callback che con le promise

https://docs.sencha.com/extjs/7.2.0/modern/Ext.Ajax.html

MODEL / STORE

- Model e store non supportano le promise
- E' possibile rimediare tramite: subclass, override o mixin
- Ext.data.Model
  - Load, Save, Erase
- Ext.data.Store
  - Load, Sync

https://docs.sencha.com/extjs/7.2.0/modern/Ext.data.Model.html

demo time

GRAZIE.

JavaScript: async/await

speaker: Luca Minuti

wintech italia