Promise e Fetch

Promises

Le promises tentano di risolvere il problema di mettere in comunazione un codice *produttore* con un codice *consumatore*.

1. Un codice *produttore* è un' operazione che fa qualcosa che richiede tempo (Ad esempio il download di un file da un server)
2. Un codice *consumatore* è un'operazione che necessita del risultato del codice *produttore*
3. Una *promise* è un particolare oggetto che connette il codice produttore con il codice consumatore, rendendo il risultato disponibile al consumatore quando il codice produttore è terminato.

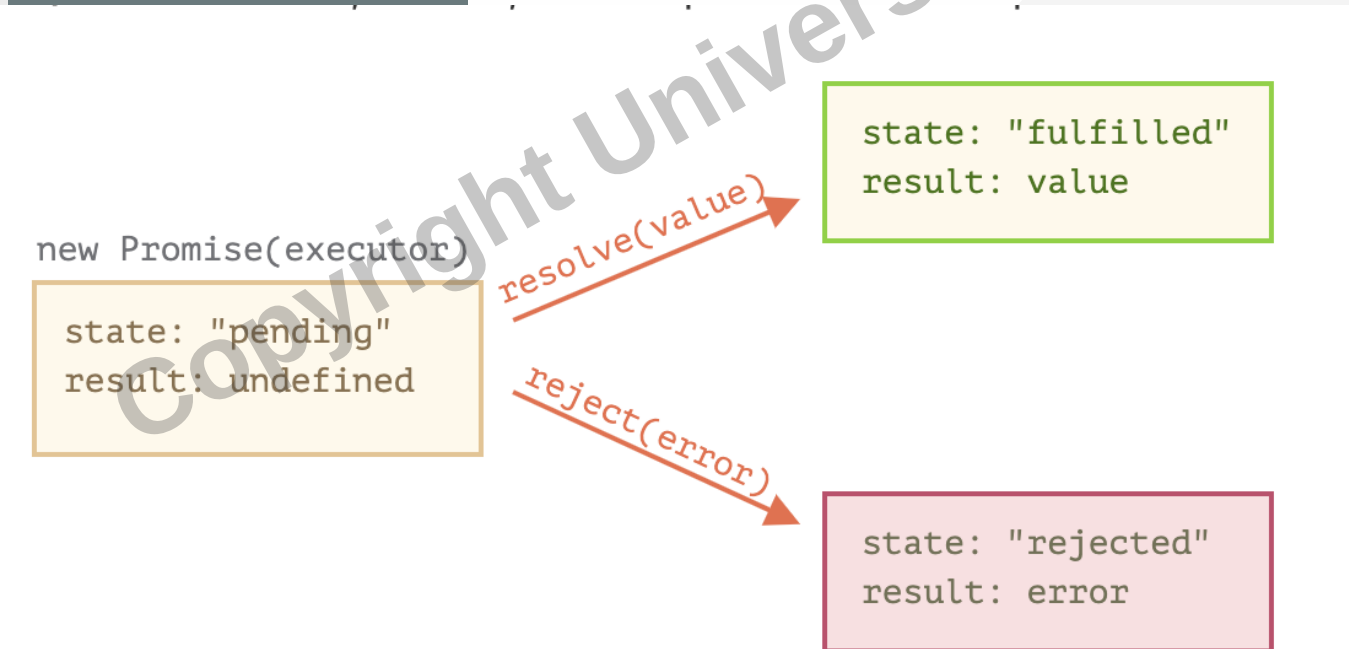
```
let promise = new Promise(function(resolve, reject) {
  // codice produttore
});
```

I due argomenti della funzione sono delle callback che vengono eseguite quando la promise è risolta o rifiutata:

1. resolve(value) - se il processo termina correttaemnte con il valore passato come parametro
2. reject(reason) - se il processo termina con un errore

L'oggetto Pomise ha anche delle proprietà interne:

1. state- inizialmente è `pending` che poi cambia in `fulfilled` se viene invocato resolve o `rejected` se viene invocato reject.
2. result — inizialmente `undefined`, poi assume il valore di `value` se viene invocato `resolve(value)` o in `error` se viene invocato `reject(error)`.



```
var promise = new Promise(function(resolve, reject) {
  // dopo 1 secondo segnala che il lavoro è finito con successo
  setTimeout(() => resolve("fatto"), 1000);
});
```

Che si usa

```
promise.then(valore => alert(valore)); // mostra "fatto!" dopo 1 secondo
```

oppure

```
promise.then(function(valore){alert(valore)}); // mostra "fatto!" dopo 1 secondo
```

Fetch

JavaScript può inviare richieste di rete al server e caricare nuove informazioni ogni volta che è necessario.

Per esempio, possiamo usare le richieste di rete per:

- Valutare un film,
- Caricare informazioni di un utente,
- Ricevere gli ultimi aggiornamenti del server,
- etc...

Il metodo `fetch()` è tra tutti il più moderno e versatile.

Questo metodo non è supportato dai browser troppo datati, ma lo è ampiamente tra quelli recenti.

La sintassi di base è:

```
var promise = fetch(url, [options])
```

- `url` – l'URL da raggiungere.
- `options` – parametri **opzionali**: metodi, headers etc.

Senza specificare nessuna `options`, la richiesta è una semplice GET che scarica il contenuto di `url`.

La funzione restituisce una *promise*
Il cui valore può essere ottenuto con *then* oppure utilizzando la parola chiave *await*

Ottenere una risposta è comunemente un processo che si svolge in due fasi:

1. Valutazione dello stato
2. Risposta

1. Valutazione dello stato

Possiamo valutare gli status HTTP dalle proprietà:

- status – HTTP status code, ad esempio 200.
- ok – boolean, true se l'HTTP status code è 200-299.

Esempio:

```
let response = await fetch(url);

if (response.ok) { // se l'HTTP-status è 200-299
  var json = await response.json();
} else {
  alert("HTTP-Error: " + response.status);
}
```

2. Risposta

Response fornisce molteplici metodi **promise-based** per accedere al body in differenti formati:

- response.text() – legge la risposta e ritorna un testo,
- response.json() – interpreta e ritorna la risposta come un JSON,
- response.formData() – ritorna la risposta come un oggetto (object) FormData,
- response.blob() – ritorna la risposta come Blob (binary data con type)

Esempio:

```
fetch('https://swapi.dev/api/films/')
  .then(response => response.json())
  .then(results => alert(results.results[0].title));
```

Richieste POST (o POST requests)

Per eseguire una richiesta `POST` o una richiesta con un altro metodo, possiamo usare le opzioni di `fetch`:

- `method` -- metodo HTTP, es. `POST`,
- `body` -- il body della richiesta ad esempio una stringa (string) (es. JSON-encoded),

Per esempio, il codice seguente invia un oggetto `user` sotto forma di JSON:

```
var user = {
  nome: 'Valerio',
  cognome: 'Bellandi'
};

var response = await fetch('esempio.com/utenti/', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json;charset=utf-8'
  },
  body: JSON.stringify(user)
});

var result = await response.json();
alert(result.message);
```