JAVASCRIPT

Riccardo Cattaneo

Lezione 9

Introduzione

Una promise è un oggetto Javascript che rappresenta un valore futuro sconosciuto. Concettualmente, una Promise è solo **Javascript che promette di restituire un valore**.

Potrebbe essere il risultato di una chiamata API o potrebbe essere un oggetto di errore da una richiesta di rete non riuscita. Hai la garanzia di ottenere qualcosa.

Promise

Le promise ci permettono di gestire processi asincroni in un modo più sincrono. Rappresentano un valore che possiamo gestire in un futuro. Si istanzia Promise e si passa una funzione chiamata resolve che riceve due parametri, uno per risolvere la promise, e l'altro per rifiutarla. Questa è la sintassi :

```
var test = new Promise(function(resolve, reject)){
        if(condizione){
                resolve(valore);
        }else{
                reject(valore);
        }
}
```

Stati di una Promise

Le Promise hanno 3 stati :

- **Pending** (pendente) quando non è stata risolta, quando siamo in attesa della risposta.
- **Fullfilled** (risolta) quando ci risponde correttamente.
- **Reject** (rifiutata) quando viene rifiutata.

Esempio

Immaginiamo di voler chiamare un web service per recuperare i dati da un database. Una volta fatta la richiesta, il server **promette** di inviarmeli **quando avrà finito**. E anche se qualcosa **dovesse andare storto**, ad esempio un errore nel database, invierà comunque una risposta.

Un “codice produttore” che fa qualcosa e che richiede tempo. Per esempio, il codice che carica uno script remoto. Un “codice consumatore” che vuole il risultato del “codice produttore” una volta che è pronto. Molte funzioni possono aver bisogno di questo risultato.

Una ***promise*** è uno speciale oggetto JavaScript che collega il "codice produttore" con il "codice consumatore". Il "codice produttore" si prende tutto il tempo necessario a produrre il risultato promesso, e la "promise" rende il risultato disponibile per tutto il codice iscritto quando è pronto. La sintassi del costruttore per un oggetto promise è:

app.js > ...

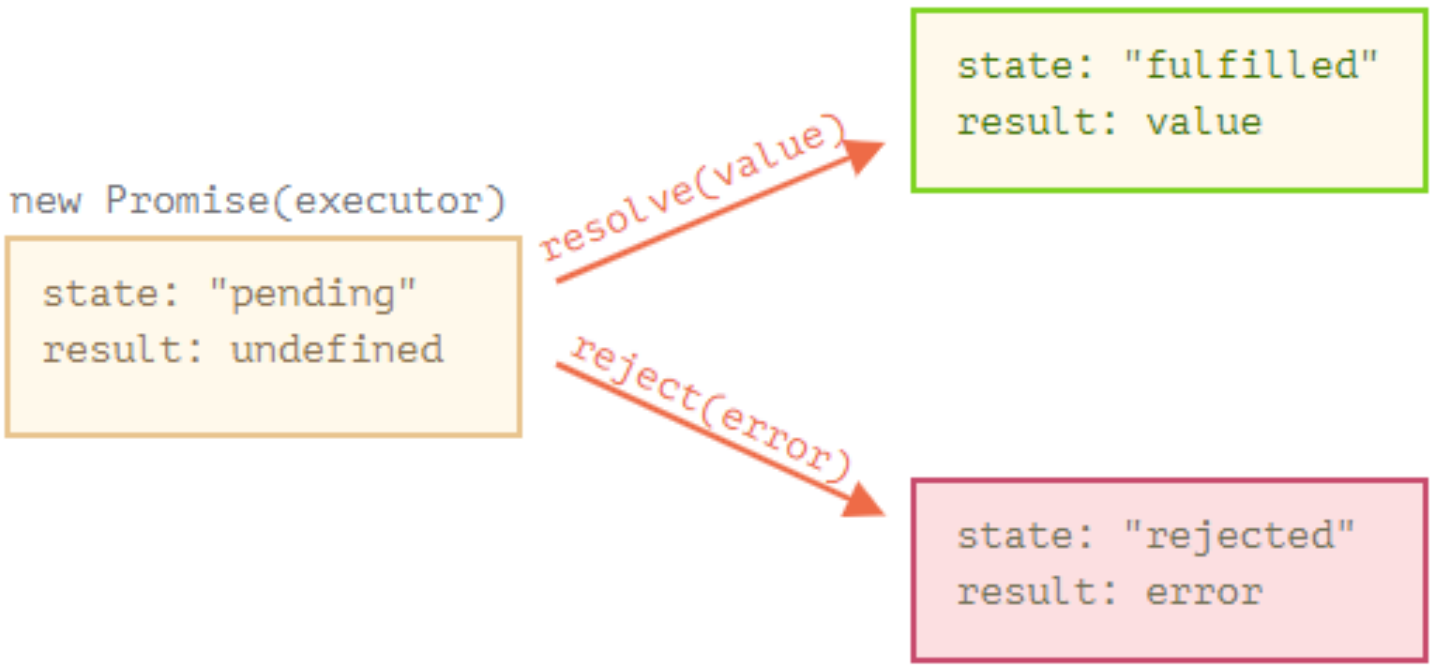
```
1
2   let promessa = new Promise(function(resolve,reject){
3       // codice produttore
4   });
5
```

La **funzione** passata a **new Promise** è chiamata esecutore. Quando la promise è creata, questa funzione esecutore viene eseguita **automaticamente**. Contiene il codice produttore, che eventualmente produrrà un risultato.

I suoi argomenti **resolve** e **reject** sono delle **callback** fornite da JavaScript stesso. Il nostro codice sta solamente dentro l'esecutore.

resolve(value) — se il processo termina correttamente, col risultato value.

reject(error) — se si verifica un errore, error è l'oggetto errore.

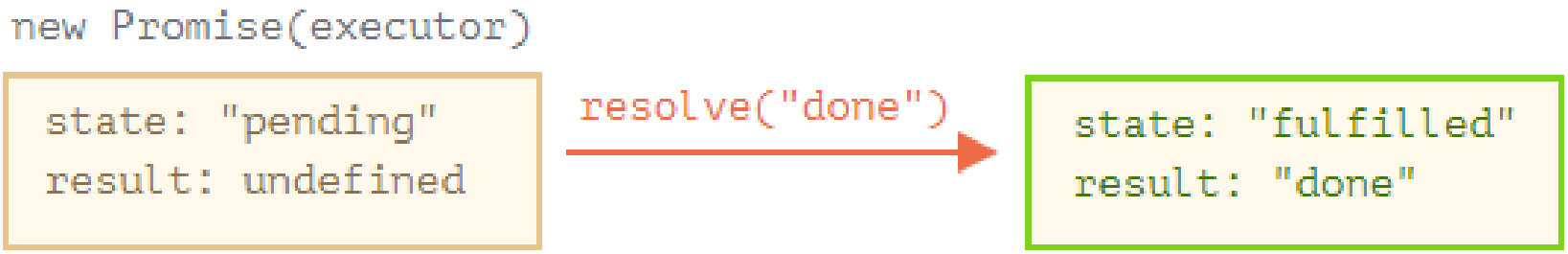

new Promise(executor)
state: "pending"
result: undefined
resolve(value)
reject(error)
state: "fulfilled"
result: value
state: "rejected"
result: error

```
1
2   let promessa = new Promise(function(resolve,reject){
3       // codice produttore
4       // dopo 2 sec segnala che è tutto ok
5       setTimeout( ()=>resolve('done'), 2000);
6   });
7
```

Possiamo vedere due cose eseguendo il codice sopra:

L'esecutore è chiamato automaticamente ed immediatamente (da new Promise). L'esecutore riceve due argomenti: resolve e reject, queste funzioni sono predefinite dal motore JavaScript. Quindi non abbiamo bisogno di crearle. Dovremo invece scrivere l'esecutore per chiamarle quando è il momento.

Dopo un secondo di “elaborazione” l’esecutore chiama resolve(“done”) per produrre il risultato. Questo cambia lo stato dell’oggetto promise :



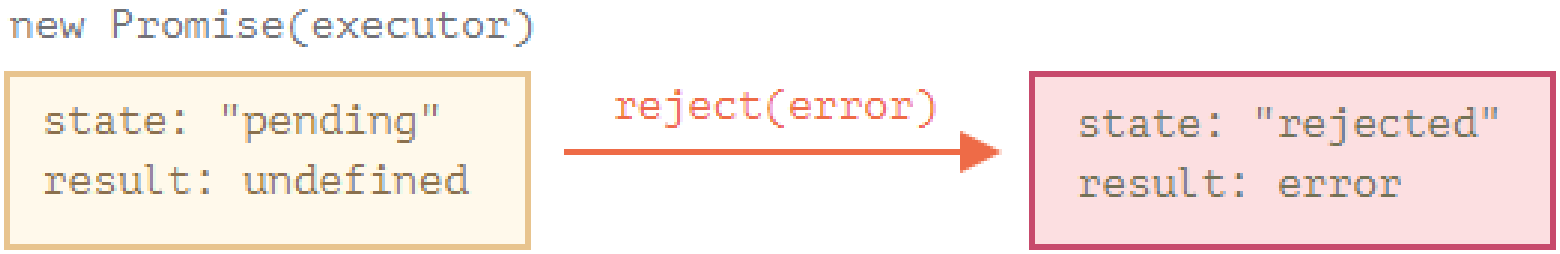
Ed ora un esempio dell'esecutore che respinge (rejecting) la promise con un errore:

```
let promise = new Promise(function(resolve, reject) {

  // dopo 1 secondo segnala che il lavoro è finito con un errore
  setTimeout(() => reject(new Error("Qualcosa non va")), 1000);

});
```

La chiamata a reject(...) sposta lo stato della Promise a "rejected":



L’esecutore può chiamare solo un resolve o un reject. Il cambiamento di stato della promise è definitivo. Tutte le chiamate successive a ‘resolve’ o ‘reject’ sono ignorate:

```
let promise = new Promise(function(resolve, reject) {

    resolve("done");

    reject(new Error("…")); // ignorato

    setTimeout(() => resolve("…")); // ignorato
});
```

Nel caso in cui qualcosa vada male, possiamo chiamare reject con qualunque tipo di argomento. Ma è raccomandato utilizzare gli oggetti Error (o oggetti che estendono Error).

Un oggetto **Promise** fa da collegamento tra l'esecutore e le funzioni consumatore, che riceveranno il risultato o un errore. Le funzioni consumatori possono essere registrate usando i metodi **.then**, **.catch** e **.finally**.

.then

Il più importante è **.then**. La sintassi è:

```
promessa.then( (ris)=>{} ).catch( (err)=>{} );
```

Il primo argomento di .then è una funzione che esegue quando una promise viene risolta, e ne riceve il risultato.

Il secondo argomento di .then è una funzione che esegue quando una promise viene rifiutata e riceve l'errore.

```javascript
let promessa = new Promise( (resolve,reject) => {

    setTimeout( () => {
        resolve("Ci sono riuscito");
    } , 3000);
});

promessa.then( (ris)=>{
    console.log(ris);
} ).catch( (err)=>{
    console.log(err);
} );
```