Node.js: Promise e async/await

JavaScript è asincrono by design e le Promise sono un modo per sfruttare al meglio questa caratteristica.

Una Promise ci permette di gestire l'esito di un'operazione che avrà un risultato nel futuro, non subito. Immaginiamo di effettuare una richiesta HTTP ad un server remoto per scaricare un file JSON.

I linguaggi sincroni come PHP aspettano il risultato della richiesta ma non possono fare altro che bloccare l'esecuzione del resto del codice durante l'attesa.

In JavaScript invece l'esecuzione del resto del codice non viene bloccata, perché la richiesta viene messa nella coda dei task da completare mentre il resto del codice procede in parallelo.

Una Promise può avere 2 esiti:

1. resolved: l'operazione ha avuto esito positivo
2. rejected: si è verificato un errore che ha impedito il completamento dell'operazione.

Definiamo una Promise in questo modo:

```
const asyncOperation = () => {
    return new Promise((resolve, reject) => {
        const response =
getJSON('https://api.site.tld/file.json');

        if(response.status === 200) {
            resolve(JSON.parse(response));
        } else {
            reject({
```

```
                status: response.status,
                message: 'Request failed'
            });
        }
    });
};
```

Invochiamo il callback `resolve` se l'operazione ha avuto esito positivo. Viceversa usiamo il callback `reject` se si è verificato un errore. In entrambi i casi passiamo un oggetto ai callback con il risultato dell'operazione.

Possiamo quindi usare la nostra funzione che restituisce una Promise con i normali metodi `then()` (successo) e `catch()` (errore):

```
app.get('/request', (req, res) => {
        asyncOperation().then(data => {
                res.json(data);
        }).catch(err => {

res.status(err.status).send(err.message);
        });
});
```

Il problema con `then()` e `catch()` è che creano dei callback, quindi se ci sono più Promise alla lunga si presenterebbe il problema del callback hell, ossia dell'annidamento di più callback.

Per ovviare a questo problema, JavaScript fornisce il modello async/await. Una funzione dichiarata come `async` può avere al suo interno la parola chiave `await` posta davanti alla funzione che restituisce la Promise. Come suggerisce il suo nome, `await` aspetta che la Promise sia completata e salva i risultati in una variabile. Con questo modello, venendo a mancare

`then()` e `catch()`, occorre usare un tradizionale blocco `try/catch` di JavaScript.

```
app.get('/request', async (req, res) => {
        try {
            const data = await asyncOperation();
            res.json(data);
        } catch(err) {
            res.status(err.status).send(err.message);
        }
```

Con async/await si ovvia al problema del callback hell mantenendo il codice conciso e leggibile.