Node.js: download multipli con le Promise

In questo tutorial vedremo come effettuare download multipli in Node.js con le Promise.

Supponiamo di avere una pagina web contenente una serie di link HTML che fanno riferimento a file video MP4. Come primo passo dobbiamo installare il modulo `cheerio` per poter estrarre gli URL da questi link.

```
npm install cheerio --save
```

Quindi definiamo una funzione che effettua il download della pagina web remota.

```
'use strict';

const https = require('https');
const http = require('http');
const fs = require('fs');
const cheerio = require('cheerio');

const getPage = pageURL => {
    return new Promise((resolve, reject) => {
        https.get(pageURL, res => {
            let data = [];
            res.on('data', chunk => {
                data.push(chunk);
            });
            res.on('end', () => {
```

```
resolve(Buffer.concat(data).toString());
            });
        });
    });
};
```

Il download della pagina avviene ricevendo un buffer di dati che verrà convertito in un'unica stringa solo alla fine del trasferimento dei dati. A questo punto dobbiamo definire la procedura di estrazione dei link dalla pagina.

```
const getFileURLS = async url => {
    const urls = [];
    try {
        const html = await getPage(url);
        const $ = cheerio.load(html);
        $('a[download]').each(function() {
            let href = $(this).attr('href');
            urls.push(href);
        });
        return urls;
    } catch(err) {
        return urls;
    }
};
```

L'array `urls` conterrà tutti gli URL dei video MP4 e viene popolato effettuando un loop con cheerio su tutti gli elementi a aventi l'attributo `download` e inserendo il valore del loro attributo `href`.

Ora dobbiamo definire la funzione che effettuerà il download di un singolo video.

```
const download = (url, destPath) => {
    return new Promise((resolve, reject) => {
        http.get(url, res => {
            const filePath =
fs.createWriteStream(destPath);
            res.pipe(filePath);
            resolve(true);
        });
    });
};
```

I dati del video contenuti come buffer di byte verrano reindirizzati allo stream creato con il metodo `createWriteStream()` tramite il metodo `pipe()` dell'oggetto `response` restituito dalla richiesta HTTP. In questo modo la performance viene ottimizzata rispetto ai normali metodi di scrittura dei file del modulo core `fs`.

Ora non ci resta che creare un array di Promise contenenti tutte le richieste HTTP dei video.

```
const createDownloadRequests = urls => {
    const requests = [];
    for(const url of urls) {
        let urlObj = new URL(url);
        let parts = urlObj.pathname.split('/');
        let filename = parts[parts.length - 1];
        requests.push(download(url,
`./video/${filename}`));
    }
    return requests;
};
```

Infine, possiamo usare il metodo `Promise.all()` (volendo anche `Promise.allSettled()` se accettiamo il fatto che alcuni download potrebbero fallire) per effettuare tutti i download.

```
(async () => {
    const baseURL =
'https://see.stanford.edu/course/cs107';
    try {
        const urls = await getFileURLS(baseURL);
        const requests =
createDownloadRequests(urls);
        await Promise.all(requests);

    } catch(err) {
        console.log(err);
    }
})();
```

In conclusione, la combinazione di Promise e stream apporta notevoli benefici in termini di performance ad un'operazione IO-bound come il download del file.