Node.js: parsing e conversione in JSON di un feed RSS

In questo tutorial vedremo come effettuare il parsing e la conversione in JSON di un feed RSS di Medium con Node.js.

Node.js non dispone di una soluzione nativa per effettuare la trasformazione di una stringa XML in un documento DOM, quindi utilizzeremo il package cheerio per questo scopo.

All'interno di un feed RSS possono essere presenti degli elementi con sezioni `CDATA` che a loro volta contengono degli elementi HTML. In tal caso è necessario utilizzare cheerio per effettuare di nuovo la conversione di tale stringa HTML in un frammento DOM.

Alcuni elementi possono essere contenuti in un namespace XML. In questo caso dovremo ricorrere alla notazione XPath compatibile di cheerio poiché il token `:` verrebbe interpretato come delimitatore di uno pseudo-elemento se utilizzassimo la notazione CSS predefinita.

Definiamo una classe con la seguente struttura iniziale:

```
'use strict';

const https = require('https');
const cheerio = require('cheerio');
const noop = arg => arg;

class MediumFeedParser {
    constructor({ url, linkFormatter = noop,
contentFormatter = noop, dateFormatter = noop }) {
        this.url = url;
```

```
        this.linkFormatter = linkFormatter;
        this.contentFormatter = contentFormatter;
        this.dateFormatter = dateFormatter;
    }
}
```

Le tre funzioni di callback definite dall'utente nel costruttore servono rispettivamente a formattare il link, il contenuto e la data di ciascun post del feed.

Il primo metodo da definire è quello che reperisce il feed remoto.

```
getFeed() {
        return new Promise((resolve, reject) => {
            try {
                const uri = new URL(this.url);
                const { hostname, pathname } = uri;
                const options = {
                    hostname: hostname,
                    port: 443,
                    path: pathname,
                    method: 'GET'
                };
                const req = https.request(options,
res => {
                    let body = '';
                    res.on('data', d => {
                        body += d;
                    });
                    res.on('end', () => {
                       resolve(body);
                    });
                });
```

```
                req.on('error', error => {
                    reject(error);
                });

                req.end();

            } catch(err) {
                reject(err);
            }
        });
    }
```

Quindi dobbiamo definire tre metodi helper per gestire le sezioni `CDATA` e reperire la prima immagine ed il testo del primo paragrafo nel post.

```
removeCDATA(str) {
        return str.replace('<![CDATA[',
'').replace(']]>', '').trim();
    }

    getImage(content) {
        const $ = cheerio.load(content);
       return $('img').eq(0).attr('src');
    }

    getContent(html) {
        const $ = cheerio.load(html);
        return $('p').eq(0).text();
    }
```

Ora definiamo il metodo che andrà ad effettuare il parsing della stringa XML del feed.

```
parseFeed(xml) {
        const $ = cheerio.load(xml, null, false);
        const output = [];
        const self = this;

        $('item').each(function() {
            let $item = $(this);
            let title =
self.removeCDATA($item.find('title').text());
            let link = $item.find('guid').text();
            let published = self.dateFormatter(new
Date($item.find('pubDate').text()));
            let permalink = self.linkFormatter(link);
            let content =
self.contentFormatter(self.getContent(self.removeCDAT
A($item.find('content\\:encoded').html())));
            let image =
self.getImage(self.removeCDATA($item.find('content\\:
encoded').html()));

            output.push({
                title,
                permalink,
                published,
                content,
                image
            });
        });
        return output;
    }
```

Il contenuto dei post è racchiuso nell'elemento `<content:encoded>` in cui a sinistra dei due punti abbiamo il riferimento al namespace, mentre a

destra troviamo il nome locale dell'elemento. Di conseguenza dobbiamo utilizzare la notazione XPath se vogliamo accedere a questo elemento.

Ora possiamo definire il metodo principale della classe.

```
async render() {
        try {
            const xml = await this.getFeed();
            const data = this.parseFeed(xml);
            return data;
        } catch(err) {
            return new Error('Error while parsing RSS
feed.');
        }
    }
```

Un esempio d'uso potrebbe essere il seguente:

```
(async () => {
    const mediumFeedParser = new MediumFeedParser({
        url: 'https://medium.com/feed/@gabriele-
romanato',
        linkFormatter(link) {
            return
link.replace('https://medium.com/p/',
'https://gabrieleromanato.it/');
        },
        dateFormatter(pubDate) {
            const year =
pubDate.getFullYear().toString();
            const month = (pubDate.getMonth() + 1) >=
10 ? (pubDate.getMonth() + 1).toString() : '0' +
(pubDate.getMonth() + 1);
            const day = pubDate.getDate() >= 10 ?
```

```
pubDate.getDate().toString() : '0' +
pubDate.getDate();

            return `${day}/${month}/${year}`;
        }
    });

    const data = await mediumFeedParser.render();

    console.log(JSON.stringify(data));
})();
```

Per le funzioni di formattazione predefinite passate al costruttore dobbiamo tenere presente che una funzione senza valore di ritorno comporterebbe la restituzione del valore `undefined` se non restituiamo invece l'argomento passato nel caso in cui un parametro del costruttore venga omesso.