Node.js: gestire le richieste HTTP

Node.js dispone del modulo core `http` per la gestione delle richieste HTTP.

Creare un server

Per poter gestire le richieste HTTP dobbiamo creare un server attraverso il metodo `createServer()` che accetta una funzione di callback come parametro.

Tale funzione a sua volta accetta due parametri, `req` (request) e `res` (response) che non sono altro che due oggetti contenenti metodi e proprietà per la gestione delle richieste e delle risposte.

Il metodo `listen` di fatto pone l'istanza del nostro server in ascolto su una porta specifica (in questo caso la 3000).

```
'use strict';

const http = require('http');

const server = http.createServer((req, res) => {
    res.writeHead(200, { 'Content-Type':
'application/json' });
    res.end(JSON.stringify({
        method: req.method,
        url: req.url,
        headers: req.headers
    }, null, 2));
});

server.listen(3000);
```

Il metodo `writeHead()` serve a inviare gli header HTTP al client mentre il metodo `end()` invia i dati al client e chiude la richiesta.

Nell'esempio possiamo vedere alcune proprietà importanti dell'oggetto request, ossia il metodo HTTP usato (GET, POST, PUT ecc.), l'URL richiesto e gli header inviati dal client al server.

Il nostro server di base risponde su qualsiasi URL passato all'indirizzo di loopback sulla porta 3000, come ad esempio `http://localhost:3000/` o `http://localhost:3000/test`.

Query string

Per impostazione predefinita il modulo di Node non gestisce automaticamente le query string come proprietà dell'oggetto `request`.

Infatti le query string fanno parte dell'URL passato al server e vanno estratte utilizzando il metodo `parse()` del modulo core `querystring` che restituisce un oggetto letterale in cui le coppie parametro/valore diventano proprietà e valori di tale oggetto.

```
'use strict';

const http = require('http');
const qs = require('querystring');

const normalizeUrl = (url) => {
    if(url.indexOf('?') !== -1) {
        return url.replace( /\/\?/g, '' );
    }
    return url;
};

const server = http.createServer((req, res) => {
    res.writeHead(200, { 'Content-Type':
```

```
'application/json' });
    res.end(JSON.stringify(
        { url: req.url, query:
qs.parse(normalizeUrl(req.url), '&', '=') }
        , null, 2));
});

server.listen(3000);
```

In questo caso abbiamo rimosso dall'URL la stringa `/?`se presente prima di passare l'URL al metodo `parse()` che come potete vedere accetta come parametri anche il separatore dei parametri della query string e il separatore tra nome del parametro e valore corrispondente.

Richieste POST

Le richieste POST vengono trattate come flusso (stream) di dati, quindi dobbiamo utilizzare gli eventi degli stream per assemblare il corpo della richiesta come stringa e quindi ricavare la query string come abbiamo visto in precedenza.

```
'use strict';

const http = require('http');
const qs = require('querystring');

const post = (request) => {
    return new Promise((resolve, reject) => {
        if(request.headers['content-type'] ===
'application/x-www-form-urlencoded') {
            let body = '';
            request.on('data', chunk => {
                body += chunk.toString();
            });
```

```
            request.on('end', () => {
                resolve(qs.parse(body));
            });
        } else {
            reject(null);
        }
    });
};

const server = http.createServer((req, res) => {
    post(req).then(body => {
        res.writeHead(200, { 'Content-Type':
'application/json' });
        res.end(JSON.stringify(body, null, 2));
    }).catch(err => {
        res.writeHead(403, { 'Content-Type':
'application/json' });
        res.end(JSON.stringify({msg: 'Invalid
request'}, null, 2));
    });

});

server.listen(3000);
```

In questo caso verifichiamo che la richiesta venga effettivamente inviata da un form HTML e quindi assembliamo in modo incrementale il corpo della richiesta da cui estrarremo la query string utilizzando il metodo visto in precedenza con l'unica differenza che in questo caso non abbiamo a che fare con un URL.

Servire pagine HTML

Per servire pagine HTML dobbiamo interagire con il file system e leggere il contenuto dei file come stringa e quindi inviare al browser tale contenuto con il tipo MIME appropriato.

```
use strict';

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

const get = (request, response) => {
  if(request.method === 'GET' ) {
    if(request.url === '/') {

        let home =
fs.readFileSync('./templates/home.html').toString();
        response.writeHead(200, { 'Content-Type':
'text/html' });
        response.end(home);

    } else {
        let notFound =
fs.readFileSync('./templates/404.html').toString();
        response.writeHead(404, { 'Content-Type':
'text/html' });
        response.end(notFound);
    }
 } else {
    response.writeHead(405, { 'Content-Type':
'text/plain' });
    response.end('Method not allowed');
 }
};

const server = http.createServer((req, res) => {
```

```
get(req, res); });

server.listen(3000);
```

La home page servirà il file richiesto, mentre gli altri URL serviranno una pagina di errore HTTP 404 (Not Found). Se il tipo di richiesta non è GET, invieremo uno status HTTP 405 che indica che il metodo usato non è ammesso.