GABRIELE ROMANATO

Node.js: introduzione ad ExpressJS

ExpressJS è il framework di riferimento per sviluppare siti ed applicativi web con Node.js.

Sostanzialmente Express è un contenitore che gestisce principalmente il modulo `http` di Node.js. Creando un'istanza dell'oggetto `express` si possono gestire percorsi, URL, e più in generale tutte quelle caratteristiche che nelle lezioni precedenti abbiamo dovuto implementare manualmente ed in un modo tutt'altro che efficiente.

Poniamo il caso in cui dobbiamo gestire un URL come `/lezioni`. In Express è sufficiente scrivere:

```
'use strict';
const app = require('express')();
app.get('/lezioni', (req, res) => {
});
app.listen(3000);
```

Come suggerisce il suo nome, il metodo `get()`gestisce le richieste GET. Se volessimo gestire un URL come `/lezioni/nodejs`, Express accetta anche i parametri come variabili:

```
app.get('/lezioni/:lezione', (req, res) => {
   console.log(req.params.lezione); // 'nodejs'
});
```

Per evitare di aggiungere due route invece di una, possiamo sfruttare una caratteristica fondamentale di Express: le espressioni regolari nei percorsi:

```
app.get('/lezioni/:lezione?', (req, res) => {
   if(req.params.lezione) {
```

```
      console.log(req.params.lezione);
   } else {
       console.log('Lezioni');
   }
});
```

Il parametro `lezione` è stato reso opzionale dal modificatore `?` delle espressioni regolari, quindi ora la route gestisce sia `/lezioni` che `/lezioni/nodejs` o altre tassonomie.

Se volessimo invece controllare anche il formato dei parametri, possiamo utilizzare espressioni regolari complete. Supponiamo di voler aggiungere il parametro della lingua ad una route, ad esempio `/percorso/en`. Vogliamo che il parametro abbia una lunghezza compresa tra 2 e 5 caratteri e sia composto solo da lettere e da un trattino.

```
app.get('/percorso/:lang([a-z-]{2,5})?', (req, res) => {

});
```

Il parametro è sempre opzionale ma se usiamo un numero non funzionerà. In pratica l'espressione regolare completa fa in modo che il parametro `lang` accetti solo un determinato tipo di formato.

Express ovviamente supporta tutti i principali verbi HTTP tramite metodi che ne rispecchiano il nome.

Per leggere i valori di una query string, Express fornisce l'oggetto `query` per le richieste GET:

```
// http://localhost:3000/?a=1&b=2
app.get('/', (req, res) => {
     console.log(req.query.a); // '1'
     console.log(req.query.b); // '2'
});
```

Le richieste POST ci introducono al concetto di middleware, ossia di quei moduli o funzioni che Express integra nel suo flusso di esecuzione e che aggiungono metodi e proprietà agli oggetti `request` e `response`.

Per gestire le richieste POST dei form si usa il middleware `body-parser`:

```
'use strict';
const app = require('express')();
const bodyParser = require('body-parser');
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended:true}));
```

Questo middleware, inserito tramite il metodo `use()`aggiunge il metodo `json()` all'oggetto risposta e l'oggetto `body` con i parametri delle richieste POST all'oggetto richiesta. Quindi avremo:

```
// POST http://localhost:3000/?a=1&b=2
app.post('/', (req, res) => {
     console.log(req.body.a); // '1'
     console.log(req.body.b); // '2'
});
```

Express gestisce i file statici in modo quasi automatico. È sufficiente infatti utilizzare il metodo `static()` specificando la directory di partenza:

```
'use strict';
const express = require('express');
const path = require('path');
const app = express();

app.use('/public', express.static(path.join(__dirname,
'/public'), {
    maxAge: 0,
    dotfiles: 'ignore',
    etag: false
}));
```

Come si può notare, Express semplifica di molto lo sviluppo con Node.js.