EBook Gratuito

APPRENDIMENTO express

Free unaffiliated eBook created from
Stack Overflow contributors.

#express

Sommario

Di

Capitolo 1: Iniziare con express

Osservazioni

Express.js è, nelle parole degli sviluppatori, un "framework web veloce, non professionale, minimalista per Node.js."

Progettato per essere minimo e flessibile, Express offre una serie di funzionalità per la creazione di applicazioni Web e mobili. Dai metodi HTTP al middleware integrato, Express è progettato per fornire le funzionalità necessarie per creare un'applicazione web o mobile su Node.js.

Se vuoi creare un'app su Node.js Express è un'ottima scelta, sia che tu utilizzi vanilla Express o uno dei tanti framework basati su Express o costruiti su Express. Alcuni di questi framework possono essere trovati qui .

Versioni

Versioni da **qui** .

Versione	Gli appunti	Data di rilascio
4.15.3		2017/05/16
4.15.2		2017/03/06
4.15.1		2017/03/05
4.15.0		2017/03/01
4.14.1		2017/01/28
4.14.0		2016/06/16
4.13.4		2016/01/21
4.13.3		2015/08/02
4.13.2		2015/07/31
4.13.1		2015/07/05
4.13.0		2015/06/20
4.12.4		2015/05/17
4.12.3		2015/03/17

Versione	Gli appunti	Data di rilascio
4.12.2		2015/03/02
4.12.1		2015/03/01
4.12.0		2015/02/23
4.11.2		2015/01/20
4.11.1		2015/01/20
4.11.0		2015/01/13
4.10.8		2015/01/13
4.10.7		2015/01/04
4.10.6		2014/12/12
4.10.5		2014/12/10
4.10.4		2014/11/24
4.10.3		2014/11/23
4.10.2		2014/11/09
4.10.1		2014/10/28
4.10.0		2014/10/23
4.9.8		2014/10/17
4.9.7		2014/10/10
4.9.6		2014/10/08
4.9.5		2014/09/24
4.9.4		2014/09/19
4.9.3		2014/09/18
4.9.2		2014/09/17
4.9.1		2014/09/16
4.9.0		2014/09/08
4.8.8		2014/09/04

Versione	Gli appunti	Data di rilascio
4.8.7		2014/08/29
4.8.6		2014/08/27
4.8.5		2014/08/18
4.8.4		2014/08/14
4.8.3		2014/08/10
4.8.2		2014/08/07
4.8.1		2014/08/06
4.8.0		2014/08/05
4.7.4		2014/08/04
4.7.3		2014/08/04
4.7.2		2014/07/27
4.7.1		2014/07/26
4.7.0		2014/07/25
4.6.1		2014/07/12
4.6.0		2014/07/11
4.5.1		2014/07/06
4.5.0		2014/07/04
4.4.5		2014/06/26
4.4.4		2014/06/20
4.4.3		2014/06/11
4.4.2		2014/06/09
4.4.1		2014/06/02
4.4.0		2014/05/30
4.3.2		2014/05/28
4.3.1		2014/05/23

Versione	Gli appunti	Data di rilascio
4.3.0		2014/05/21
4.2.0		2014/05/11
4.1.2		2014/05/08
4.1.1		2014/04/27
4.1.0		2014/04/24
4.0.0		2014/04/09
3.21.2	Da qui a	2015/07/31
3.21.1	3.18.6 date	2015/07/05
3.21.0	sembra sbagliato	2015/06/18
3.20.3		2015/05/17
3.20.2		2015/03/16
3.20.1		2015/02/28
3.20.0		2015/02/18
3.19.2		2015/02/01
3.19.1		2015/01/20
3.19.0		2015/01/09
3.18.6		2014/12/12
3.18.5		2014/12/11
3.18.4		2014/11/23
3.18.3		2014/11/09
3.18.2		2014/10/28
3.18.1		2014/10/22
3.18.0		2014/10/17
3.17.8		2014/10/15
3.17.7		2014/10/08

Versione	Gli appunti	Data di rilascio
3.17.6		2014/10/02
3.17.5		2014/09/24
3.17.4		2014/09/19
3.17.3		2014/09/18
3.17.2		2014/09/15
3.17.1		2014/09/08
3.17.0		2014/09/08
3.16.10		2014/09/04
3.16.9		2014/08/29
3.16.8		2014/08/27
3.16.7		2014/08/18
3.16.6		2014/08/14
3.16.5		2014/08/11
3.16.4		2014/08/10
3.16.3		2014/08/07
3.16.2		2014/08/07
3.16.1		2014/08/06
3.16.0		2014/08/05
3.15.3		2014/08/04
3.15.2		2014/07/27
3.15.1		2014/07/26
3.15.0		2014/07/22
3.14.0		2014/07/11
3.13.0		2014/07/03
3.12.1		2014/06/26

Versione	Gli appunti	Data di rilascio
3.12.0		2014/06/21
3.11.0		2014-06-19
3.10.5		2014/06/11
3.10.4		2014/06/09
3.10.3		2014/06/05
3.10.2		2014/06/03
3.10.1		2014/06/03
3.10.0		2014/06/02
3.9.0		2014/05/30
3.8.1		2014/05/27
3.8.0		2014/05/21
3.7.0		2014/05/18
3.6.0		2014/05/09
3.5.3		2014/05/08
3.5.2		2014/04/24
3.5.1		2014/03/25
3.5.0		2014/03/06
3.4.8		2014/01/13
3.4.7		2013/12/10
3.4.6		2013/12/01
3.4.5		2013/11/27
3.4.4		2013/10/29
3.4.3		2013/10/23
3.4.2		2013/10/18
3.4.1		2013/10/15

Versione	Gli appunti	Data di rilascio
3.4.0		2013/09/07
3.3.8		2013/09/02
3.3.7		2013/08/28
3.3.6		2013/08/27
3.3.4		2013/07/08
3.3.3		2013/07/04
3.3.2		2013/07/03
3.3.1		2013/06/27
3.3.0		2013/06/27
3.2.6		2013/06/02
3.2.5		2013/05/21
3.2.4		2013/05/09
3.2.3		2013/05/07
3.2.2		2013/05/03
3.2.1		2013/04/29
3.2.0		2013/04/15
3.1.2		2013/04/12
3.1.1		2013/04/01
3.1.0		2013/01/25
3.0.6		2013/01/04
3.0.5		2012/12/19
3.0.4		2012/12/05
3.0.3		2012/11/13
3.0.2		2012/11/08
3.0.1		2012-11-01

Versione	Gli appunti	Data di rilascio
3.0.0		2012/10/23
3.0.0rc5		2012/09/18
3.0.0rc4		2012/08/30
3.0.0rc3		2012-08-13
3.0.0rc2		2012-08-03
3.0.0rc1		2012-07-24
3.0.0beta7		2012-07-16
3.0.0beta6		2012-07-13
3.0.0beta5		2012-07-03
3.0.0beta4		2012-06-25
3.0.0beta3		2012-06-15
3.0.0beta2		2012-06-06
3.0.0beta1		2012-06-01
3.0.0alpha5		2012-05-30
3.0.0alpha4		2012-05-09
3.0.0alpha3		2012-05-04
3.0.0alpha2		2012-04-26
3.0.0alpha1		2012-04-15
2.5.9		2012-04-02
2.5.8		2012-02-08
2.5.7		2012-02-06
2.5.6		2012-01-13
2.5.5		2012-01-08
2.5.4		2012-01-02
2.5.3		2011-12-30

Versione	Gli appunti	Data di rilascio
2.5.2		2011-12-10
2.5.1		2011-11-17
2.5.0		2011-10-24
2.4.7		2011-10-05
2.4.6		2011-08-22
2.4.5		2011-08-19
2.4.4		2011-08-05
2.4.3		2011-07-14
2.4.2		2011-07-06
2.4.1		2011-07-06
2.4.0		2011-06-28
2.3.12		2011-06-22
2.3.11		2011-06-04
2.3.10		2011-05-27
2.3.9		2011-05-25
2.3.8		2011-05-24
2.3.7		2011-05-23
2.3.6		2011-05-20
2.3.5		2011-05-20
2.3.4		2011-05-08
2.3.3		2011-05-03
2.3.2		2011-04-27
2.3.1		2011-04-26
2.3.0		2011-04-25
2.2.2		2011-04-12

Versione	Gli appunti	Data di rilascio
2.2.1		2011-04-04
2.2.0		2011-03-30
2.1.1		2011-03-29
2.1.0		2011-03-24
2.0.0		2011-03-17
2.0.0rc3		2011-03-17
2.0.0rc2		2011-03-17
2.0.0rc		2011-03-14
2.0.0beta3		2011-03-09
2.0.0beta2		2011-03-07
2.0.0beta		2011-03-03
1.0.8		2011-03-01
1.0.7		2011-02-07
1.0.6		2011-02-07
1.0.5		2011-02-05
1.0.4		2011-02-05
1.0.3		2011-01-13
1.0.2		2011-01-10
1.0.1		2010-12-29
1.0.0		2010-11-16
1.0.0rc4		2010-10-14
1.0.0rc3		2010-09-20
1.0.0rc2		2010-08-17
1.0.0rc		2010-07-28
1.0.0beta2		2010-07-23

Versione	Gli appunti	Data di rilascio
1.0.0beta		2010-07-15
0.14.0		2010-06-15
0.13.0		2010-06-01
0.12.0		2010-05-22
0.11.0		2010-05-06
0.10.1		2010-05-03
0.10.0		2010-04-30
0.9.0		2010-04-14
0.8.0		2010-03-19
0.7.6		2010-03-19
0.7.5		2010-03-16
0.7.4		2010-03-16
0.7.3		2010-03-16
0.7.2		2010-03-16
0.7.1		2010-03-16
0.7.0		2010-03-15
0.6.0		2010-03-11
0.5.0		2010-03-10
0.4.0		2010-02-11
0.3.0		2010-02-11
0.2.1		2010-02-05
0.2.0		2010-02-03
0.1.0		2010-02-03
0.0.2		2010-01-10
0.0.1	Versione iniziale	2010-01-03

Examples

Installazione

Express JS è il framework goto per lo sviluppo di `Web Applications` , `APIs` e quasi tutti i tipi di `Backend` utilizzano il nodo.

Per installare `express` , tutto ciò che devi fare è eseguire il comando **npm**

```
npm install express --save
```

E hai finito.

Per creare ed eseguire un nuovo server Express

crea un file `app.js` e aggiungi questo codice

```
// require express
var express = require('express');
var app = express();

// when "/" is opened in url, this function will be called.
app.get('/', function (req, res) {
  res.json({ code: 200, message: 'success' });
})

app.listen( 3000, function () {
  console.log('Express server running at http://localhost:3000');
});
```

- Nel tuo terminale, esegui `node app.js` e
- Aprire l'URL `http://localhost:3000` nel browser Web per visualizzare il server Express appena creato.

È anche consigliabile installare `body-parser` e `express-session` insieme a `express` poiché la maggior parte delle volte vorrai leggere i dati inviati in `POST` request e gestire le sessioni utente.

- body-parser su github
- express-session su github

Hello World App, utilizzando ExpressJS 4 e nodo> = 4

Prefazione

Avrai bisogno di `node >= 4` ed `express 4` per questo progetto. È possibile ottenere l'ultima distribuzione del `node` dalla loro pagina di download .

Prima di questo tutorial, è necessario inizializzare il progetto del nodo eseguendo

```
$ npm init
```

dalla riga di comando e compilando le informazioni desiderate. Si noti che è possibile modificare le informazioni in qualsiasi momento modificando il file `package.json` .

Installazione

Installa `express` con `npm` :

```
$ npm install --save express
```

Dopo aver installato Express come modulo nodo, possiamo creare il nostro punto di ingresso. Questo dovrebbe essere nella stessa directory del nostro `package.json`

```
$ touch app.js
```

Contenuto della directory

La cartella dovrebbe avere la seguente struttura di directory:

```
<project_root>
 |-> app.js
 |-> node_modules/
 '-> package.json
```

Codice

Apri `app.js` nell'editor preferito e segui questi quattro passaggi per creare la tua prima app Express:

```
// 1. Import the express library.
import express from 'express';

// 2. Create an Express instance.
const app = express();

// 3. Map a route. Let's map it to "/", so we can visit "[server]/".
app.get('/', function(req, res) {
   res.send('Hello World');
});

// 4. Listen on port 8080
app.listen(8080, function() {
   console.log('Server is running on port 8080...');
});
```

Esecuzione

Dalla directory del progetto, possiamo eseguire il nostro server usando il comando

```
$ node app.js
```

Dovresti vedere il testo

```
$ Our Express App Server is listening on 8080...
```

Ora, visita `http://localhost:8080/` e vedrai il testo "Hello World!"

Congratulazioni, hai creato la tua prima app Express!

Avvio di un'applicazione con il generatore Express

Per iniziare rapidamente con Express, puoi utilizzare il generatore Express che creerà uno scheletro dell'applicazione per te.

Innanzitutto, installalo globalmente con npm:

```
npm install express-generator -g
```

Potrebbe essere necessario mettere `sudo` prima di questo comando se si ottiene un errore "permesso negato".

Una volta installato il generatore, puoi iniziare un nuovo progetto come questo:

```
express my_app
```

Il comando precedente creerà una cartella chiamata `my_app` con un file `package.json` , un file `app.js` e alcune sottocartelle come `bin` , `public` , `routes` , `views` .

Ora vai alla cartella e installa le dipendenze:

```
cd first_app
npm install
```

Se sei su Linux o macOS, puoi avviare l'app in questo modo:

```
DEBUG=myapp:* npm start
```

Oppure, se sei su Windows:

```
set DEBUG=myapp:* & npm start
```

Ora, carica http: // localhost: 3000 / nel tuo browser web e dovresti vedere le parole "Welcome to Express".

Creazione di un'app EJS

```
a@coolbox:~/workspace$ express --ejs my-app
a@coolbox:~/workspace$ cd my-app
a@coolbox:~/workspace/my-app$ npm install
a@coolbox:~/workspace/my-app$ npm start
```

Leggi Iniziare con express online: https://riptutorial.com/it/express/topic/1616/iniziare-con-express

Capitolo 2: Collegare

Examples

Connetti ed esprimi

Express si basa su Connect, che è ciò che fornisce la funzionalità middleware di Express. Per capire cos'è la connessione, puoi vedere che fornisce la struttura di base dell'app che usi quando usi express

```
const connect = require('connect')

const app = connect()
app.listen(3000)
```

Verrà aperto un server http "vuoto" che risponderà 404 a tutte le richieste.

middleware

Il middleware è collegato all'oggetto app, di solito prima che venga chiamato l'ascolto. Esempio di un semplice middleware di registrazione:

```
app.use(function (req, res, next) {
    console.log(`${req.method}: ${req.url}`)
    next()
})
```

Tutto ciò che faremo è loggare `GET: /example` se si è dove GET `localhost:3000/example` . Tutte le richieste restituiranno ancora 404 poiché non si risponde con alcun dato.

Il prossimo middleware nella catena verrà eseguito non appena il precedente chiama `next()` , quindi possiamo andare avanti e rispondere alle richieste aggiungendo un altro middleware come questo:

```
app.use(function (req, res, next) {
    res.end(`You requested ${req.url}`)
})
```

Ora quando richiedi'localhost: 3000 / example `you will be greeted with "You requested /example". There is no need to call` dopo questa volta poiché questo middleware è l'ultimo della catena (ma non succederà niente di male se lo facessi),

Programma completo fino a qui:

```
const connect = require('connect')

const app = connect()
```

```
app.use(function (req, res, next) {
    console.log(`${req.method}: ${req.url}`)
    next()
})

app.use(function (req, res, next) {
    res.end(`You requested ${req.url}`)
    next()
})

app.listen(3000)
```

Errori e middleware di errore

Se vorremmo limitare l'accesso alla nostra app, potremmo scrivere anche un middleware! Questo esempio ti concede l'accesso solo nei giorni festivi, ma un esempio reale potrebbe, ad esempio, essere *l'autenticazione dell'utente* . Un buon posto per mettere questo sarebbe dopo il middleware di registrazione ma prima che venga inviato qualsiasi contenuto.

```
app.use(function (req, res, next) {
    if (new Date().getDay() !== 4) {
        next('Access is only granted on thursdays')
    } else {
        next()
    }
})
```

Come puoi vedere in questo esempio, inviare un errore è facile come fornire un prameter alla funzione `next()` .

Ora, se visitiamo il sito in qualsiasi giorno diverso da giovedì, verremmo accolti con un errore di 500 e la stringa `'Access is only granted on thursdays'` .

Ora, questo non è abbastanza buono per il nostro sito. Preferiamo inviare all'utente un messaggio HTML in un altro middleware:

```
app.use(function (err, req, res, next) {
    res.end(`<h1>Error</h1><p>${err}</p>`)
})
```

Funziona come un blocco catch: qualsiasi errore nel middleware prima del middleware degli errori verrà inviato al primo. Un middleware di errore è identificato dai suoi 4 parametri.

Puoi anche utilizzare il middleware degli errori per recuperare dall'errore chiamando di nuovo il metodo successivo:

```
app.use(function (err, req, res, next) {
    // Just joking, everybody is allowed access to the website!
    next()
})
```

Leggi Collegare online: https://riptutorial.com/it/express/topic/4031/collegare

Capitolo 3: Come funziona ExpressJs

Examples

Gestione richiesta / risposta

Lo zucchero sintattico

La maggior parte degli esempi introduttivi di ExpressJ includono questo pezzo di codice

```
var express = require('express');
var app = express();
...
app.listen(1337);
```

Bene, `app.listen` è solo una scorciatoia per:

```
var express = require('express');
var app = express();
var http = require('http');
http.createServer(app).listen(1337);
```

L'app Express

Il famoso `http.createServer` accetta una funzione che è nota come gestore. Il gestore accetta 2 parametri di **richiesta** e **risposta** come input, quindi li manipola all'interno del suo ambito per fare varie cose.

Quindi fondamentalmente `app = express()` è una funzione, che si svolge come gestore e che si occupa di richiesta, risposta attraverso un insieme di componenti speciali chiamati middleware.

Stack di middlewares

Un middleware di base è una funzione che richiede 3 argomenti di **richiesta** , **risposta** e **successiva** .

Quindi, da `app.use` , un middleware viene montato `app.use` Express Middlewares Stack. La richiesta e la risposta vengono manipolate in ogni middleware, quindi trasmesse a quella successiva tramite la chiamata di `next()` .

Ad esempio, il codice seguente:

```
var express = require('express');
var app = express();
```

```
app.use((request, response, next) => {
    request.propA = "blah blah";
    next();
});

app.use('/special-path', (request, response, next) => {
    request.propB = request.propA + " blah";
    if (request.propB === "blah blah blah")
        next();
    else
        response.end('invalid');
});

app.use((request, response, next) => {
    response.end(request.propB);
});

app.listen(1337);
```

Può essere approssimativamente tradotto in:

```
var http = require('http');
http.createServer((request, response) => {

    //Middleware 1
    if (isMatch(request.url, '*')) {
        request.propA = "blah blah";
    }

    //Middleware 2
    if (isMatch(request.url, "/special-path")) {
        request.propB = request.propA + " blah";
        if (request.propB !== "blah blah blah")
            return response.end('invalid');
    }

    //Middleware 3
    if (isMatch(request.url, "*")) {
        return response.end(request.propB);
    }
});

server.listen(1337);
```

Leggi Come funziona ExpressJs online: https://riptutorial.com/it/express/topic/7815/come-funziona-expressjs

Capitolo 4: Express Database Integration

Examples

Connettiti a MongoDB con Node & Express

Innanzitutto, assicurati di aver installato *mongodb* ed *esprimi* tramite npm. Quindi, in un file convenzionalmente denominato *db.js* , utilizzare il seguente codice:

```
var MongoClient = require('mongodb').MongoClient

var state = {
    db: null,
}

exports.connect = function(url, done) {
    if (state.db) return done()

    MongoClient.connect(url, function(err, db) {
        if(err) return done(err)
        state.db = db
        done()
    })
}

exports.get = function() {
    return state.db
}

exports.close = function(done) {
    if (state.db) {
        state.db.close(function(err, result) {
            state.db = null;
            state.mode = null;
            done(err);
        })
    }
}
```

Questo file si connetterà al database e quindi puoi semplicemente usare l'oggetto **db** restituito dal metodo **get** .

Ora devi includere il file db richiedendolo nel tuo file app.js. Supponendo che il tuo file *db.js* si trovi nella stessa directory di *app.js* puoi inserire la linea:

```
var db = require('./db');
```

Questo, tuttavia, non ti collega effettivamente all'istanza MongoDB. Per fare ciò, inserisci il codice seguente prima di chiamare il metodo app.listen. Nel nostro esempio integriamo la gestione degli errori e il metodo app.listen nella connessione al database. Si noti che questo codice funziona solo se si esegue l'istanza di mongo sullo stesso computer in cui si trova l'app Express.

```
db.connect('mongodb://localhost:27017/databasename', function(err) {
    if (err) {
        console.log('Unable to connect to Mongo.');
        process.exit(1);
    } else {
        app.listen(3000, function() {
            console.log('Listening on port 3000...');
        });
    }
});
```

Ecco fatto, la tua app Express dovrebbe ora essere collegata al tuo Mongo DB. Congratulazioni!

Leggi Express Database Integration online: https://riptutorial.com/it/express/topic/7002/express-database-integration

Capitolo 5: Express-generatore

Parametri

Parametro	Definizione
-h, --help	informazioni sull'utilizzo dell'output
-V, --version	mostra il numero di versione
-e, --ejs	aggiungi il supporto per il motore di template per pjs (JavaScript incorporato) (predefinito per jade, che è stato rinominato in Pug)
--hbs	aggiungi manubri che supportano il supporto del motore
-H, --hogan	aggiungi il supporto del motore hogan.js
--idiota	aggiungi .gitignore
-f, --force	forza sulla directory non vuota
-c <motore>, --css <motore>	aggiungi foglio di stile <motore> supporto (meno, stilo, bussola, sass) (il valore predefinito è css)

Osservazioni

Il generatore espresso è un ottimo strumento per ottenere rapidamente un progetto in piena efficienza. Una volta compresa l'organizzazione che implementa, è un risparmio in tempo reale.

Examples

Installazione di Express Generator

```
npm --install express-generator -g
```

Creare un'app

```
express my-app
```

Avvia l'app

Usando l'opzione di avvio

```
npm start
```

Utilizzando Nodemon

```
nodemon
```

Usando per sempre

```
forever start 'js file name'
```

Fermarsi per sempre

```
forever stop ''js file name'
```

Per ricominciare per sempre

```
forever restart 'js filename'
```

Elenca il server che si sta spegnendo usando per sempre

```
forever list
```

Leggi Express-generatore online: https://riptutorial.com/it/express/topic/4512/express-generatore

Capitolo 6: Gestione degli errori

Sintassi

- `app.use(function(err, req, res, next) {})` // Middleware di base

Parametri

Nome	Descrizione
`err`	Oggetto con informazioni di errore
`req`	Oggetto di richiesta HTTP
`res`	Oggetto di risposta HTTP
`next`	funzione utilizzata per avviare l'esecuzione successiva del middleware

Examples

Campione di base

A differenza di altre funzioni middleware, le funzioni middleware di gestione degli errori hanno quattro argomenti anziché tre: `(err, req, res, next)` .

Campione:

```
app.use(function(err, req, res, next) {
  console.error(err.stack);
  res.status(500).send('Error found!');
});
```

Leggi Gestione degli errori online: https://riptutorial.com/it/express/topic/2739/gestione-degli-errori

Capitolo 7: Gestione di file statici

Sintassi

1. Per servire file statici (immagini, CSS, file JS, ecc.) Utilizzare la funzione middleware **express.static** .
2. Passa il nome della directory che contiene le risorse per **express.static** per servire direttamente i file. (Guarda l' *esempio di base*)
3. È possibile utilizzare più directory, è sufficiente chiamare più volte **express.static** . Ricorda, Express cerca i file nell'ordine in cui hai impostato le directory con **express.static** . (Guarda l' *esempio di directory multiple*)
4. È possibile creare un prefisso del percorso virtuale (ad esempio quello in cui il percorso non esiste effettivamente nel file system) con **express.static** , basta specificare un percorso di montaggio. (Guarda l' *esempio del prefisso del percorso virtuale*)
5. Tutti i percorsi precedenti sono stati relativi alla directory da cui si avvia il processo del *nodo* . Pertanto, è generalmente più sicuro utilizzare il percorso assoluto della directory che si desidera servire. (Guarda l' *esempio di directory Percorso assoluto a file statici*)
6. È possibile combinare e abbinare le opzioni di questo metodo, come mostrato *nell'Esempio prefisso percorso assoluto a directory e percorso virtuale*

Osservazioni

Tutti gli esempi possono essere eseguiti nel nodo. Basta copiare e incollare in un progetto nodo con Express installato ed eseguirli con **nome file nodo** . Per un esempio di come installare express fai clic qui e assicurati di avere installato npm, quindi segui le istruzioni sull'installazione dei pacchetti per installare "express".

Examples

Esempio di base

```
// Basic code for Express Instance
var express = require('express');
var app = express();

// Serve static files from directory 'public'
app.use(express.static('public'));

// Start Express server
app.listen(3030);
```

Esempio di directory multiple

```
// Set up Express
var express = require('express');
```

```
var app = express();

// Serve static assets from both 'public' and 'files' directory
app.use(express.static('public');
app.use(express.static('files');

// Start Express server
app.listen(3030);
```

Esempio di prefisso del percorso virtuale

```
// Set up Express
var express = require('express');
var app = express();

// Specify mount path, '/static', for the static directory
app.use('/static', express.static('public'));

// Start Express server
app.listen(3030);
```

Esempio di directory dei percorsi di file statici assoluti

```
// Set up Express
var express = require('express');
var app = express();

// Serve files from the absolute path of the directory
app.use(express.static(__dirname + '/public'));

// Start Express server
app.listen(3030);
```

Esempio di prefisso percorso per directory e percorso virtuale assoluto

```
// Set up Express
var express = require('express');
var app = express();

/*  Serve from the absolute path of the directory that you want to serve with a
 */ virtual path prefix
app.use('/static', express.static(__dirname + '/public'));

// Start Express server
app.listen(3030);
```

File statici di base e favicon servono come esempio

```
var express = require('express');
var path = require('path');
var favicon = require('serve-favicon');

var app = express();
```

```
app.use(favicon(__dirname + '/public/img/favicon.ico'));
app.use(express.static(path.join(__dirname, 'public')));

app.listen(3000, function() {
  console.log("Express App listening on port 3000");
})
```

Leggi Gestione di file statici online: https://riptutorial.com/it/express/topic/6954/gestione-di-file-statici

Capitolo 8: Registrazione

Osservazioni

`morgan` è un middleware del logger delle richieste HTTP per node.js

Examples

Installazione

Innanzitutto, installa il middleware `morgan` nel tuo progetto

```
npm install --save morgan
```

Semplice registrazione rapida di tutte le richieste su STDOUT

Aggiungi il seguente codice al tuo file `app.js` :

```
var express = require('express')
var morgan = require('morgan')

var app = express()

app.use(morgan('combined'))

app.get('/', function (req, res) {
  res.send('hello, world!')
})
```

Ora quando accedi al tuo sito web vedrai nella console che hai usato per avviare il server che le richieste sono state registrate

Scrivi i log Express in un singolo file

Innanzitutto, installa `fs` e `path` nel tuo progetto

```
npm install --save fs path
```

Aggiungi il seguente codice al tuo file `app.js` :

```
var express = require('express')
var fs = require('fs')
var morgan = require('morgan')
var path = require('path')

var app = express()

// create a write stream (in append mode)
```

```
var accessLogStream = fs.createWriteStream(path.join(__dirname, 'access.log'), {flags: 'a'})

// setup the logger
app.use(morgan('combined', {stream: accessLogStream}))

app.get('/', function (req, res) {
  res.send('hello, world!')
})
```

Ora quando accedi al tuo sito web vedrai un file `access.log` creato nella directory del tuo progetto

Scrivi i log Express in un file di registro rotante

Innanzitutto, installa `fs` , `file-stream-rotator` e `path` nel progetto

```
npm install --save fs file-stream-rotator path
```

Aggiungi il seguente codice al tuo file `app.js` :

```
var FileStreamRotator = require('file-stream-rotator')
var express = require('express')
var fs = require('fs')
var morgan = require('morgan')
var path = require('path')

var app = express()
var logDirectory = path.join(__dirname, 'log')

// ensure log directory exists
fs.existsSync(logDirectory) || fs.mkdirSync(logDirectory)

// create a rotating write stream
var accessLogStream = FileStreamRotator.getStream({
  date_format: 'YYYYMMDD',
  filename: path.join(logDirectory, 'access-%DATE%.log'),
  frequency: 'daily',
  verbose: false
})

// setup the logger
app.use(morgan('combined', {stream: accessLogStream}))

app.get('/', function (req, res) {
  res.send('hello, world!')
})
```

Ora quando accedi al tuo sito web vedrai una directory di `log` creata e un file di log con un formato di nome di `access-%DATE%.log` stato creato nella tua directory di log

Leggi Registrazione online: https://riptutorial.com/it/express/topic/7191/registrazione

Capitolo 9: Routing

Examples

Routing Hello World

Il file principale dell'applicazione carica il file di rotte in cui sono definiti i percorsi.

app.js

```
var express = require('express');
var app = express();

app.use('/', require('./routes'));

app.listen('3000');
```

routes.js

```
var router = require('express').Router();

router.get('/', function(req, res) {
    res.send('Hello World!');
});

module.exports = router;
```

Routing middleware

Il middleware viene eseguito prima dell'esecuzione del percorso e può decidere se eseguire il router in base all'URL.

```
var router = require('express').Router();

router.use(function (req, res, next) {
    var weekDay = new Date().getDay();
    if (weekDay === 0) {
        res.send('Web is closed on Sundays!');
    } else {
        next();
    }
})

router.get('/', function(req, res) {
    res.send('Sunday is closed!');
});

module.exports = router;
```

Il middleware specifico può anche essere inviato a ciascun gestore di router.

```
var closedOnSundays = function (req, res, next) {
    var weekDay = new Date().getDay();
    if (weekDay === 0) {
        res.send('Web is closed on Sundays!');
    } else {
        next();
    }
}

router.get('/', closedOnSundays, function(req, res) {
    res.send('Web is open');
});

router.get('/open', function(req, res) {
    res.send('Open all days of the week!');
});
```

Percorsi multipli

Il file principale dell'applicazione carica tutti i file di rotte in cui si desidera definire percorsi. Per fare ciò abbiamo bisogno della seguente struttura di directory: app.js routes / index.js routes / users.js

app.js

```
var express = require('express');
var app = express();

app.use('/', require('./routes/index'));
app.use('/users', require('./routes/users'))

app.listen('3000');
```

Percorsi / index.js

```
var router = require('express').Router();

router.get('/', function(req, res) {
    res.send('Index Page');
});

router.get('/about', function(req, res) {
    res.send('About Page');
});

module.exports = router;
```

Percorsi / users.js

```
var router = require('express').Router();

router.get('/', function(req, res) {
    res.send('Users Index Page');
});
```

```
router.get('/list', function(req, res) {
    res.send('Users List Page');
});

module.exports = router;
```

Eseguendo `$ node app.js` ora dovrebbero esserci pagine ai seguenti URL:

- localhost: 3000 / - Visualizza "Pagina indice"
- localhost: 3000 / about - Visualizza "About Page"
- localhost: 3000 / users - Visualizza "Users Index Page"
- localhost: 3000 / users / list - Visualizza "Users List Page"

Leggi Routing online: https://riptutorial.com/it/express/topic/2589/routing

Capitolo 10: Scrivere il middleware espresso

Sintassi

1. Specifica l'istanza di express che desideri utilizzare. Questa è comunemente *app* .
2. Definire il metodo HTTP per cui si applica la funzione. Nell'esempio, questo è *get* .
3. Definire il percorso a cui si applica la funzione. Nell'esempio, questo è *'/'* .
4. Definire come una funzione con la parola chiave *function* .
5. Aggiungere i parametri richiesti: req, res, next. (Vedi nota nella sezione commenti)
6. Metti un codice nella funzione per fare quello che vuoi

Parametri

Parametro	Dettagli
req	L'oggetto della richiesta.
res	L'oggetto risposta.
Il prossimo	La prossima () chiamata middleware.

Osservazioni

Una funzione middleware è una funzione con accesso all'oggetto request (*req*), all'oggetto response (*res*) e alla funzione middleware *next ()* nel ciclo richiesta-risposta dell'applicazione. La funzione middleware *next ()* è comunemente indicata da una variabile chiamata next.

Le funzioni di middleware sono progettate per svolgere le seguenti attività:

- Esegui qualsiasi codice.
- Apporta le modifiche agli oggetti richiesta e risposta. (Vedi l'esempio requestTime)
- Termina il ciclo richiesta-risposta.
- Chiama il prossimo middleware nella pila. (Chiamando il middleware *successivo ()*)

Nota: non deve essere nominato successivamente. Ma se usi qualcos'altro, nessuno saprà cosa intendi e sarai licenziato. E il tuo codice non funzionerà. Quindi, basta nominare il prossimo. Questa regola si applica all'oggetto richiesta e risposta. Alcune persone useranno richiesta e risposta invece di req e res, rispettivamente. Va bene. Spreca battiture, ma va bene.

Examples

Logger Middleware

Se sei nuovo del middleware in Express consulta la Panoramica nella sezione Note.

Per prima cosa, configureremo una semplice app Hello World a cui verrà fatto riferimento e aggiunta durante gli esempi.

```
var express = require('express');
var app = express();

app.get('/', function(req, res) {
    res.send('Hello World!');
});

app.listen(3000);
```

Ecco una semplice funzione middleware che registrerà "LOGGED" quando viene chiamato.

```
var myLogger = function (req,res,next) {
    console.log('LOGGED');
    next();
};
```

*Calling **next ()** richiama la funzione middleware successiva nell'app.*

Per caricare la funzione chiama app.use () e specifica la funzione che desideri chiamare. Questo viene fatto nel seguente blocco di codice che è un'estensione del blocco Hello World.

```
var express = require('express');
var app = express();

var myLogger = function (req, res, next) {
    console.log('LOGGED');
    next();
};

app.use(myLogger);

app.get('/', function(req, res) {
    res.send('Hello World!');
});

app.listen(3000);
```

Ora ogni volta che l'app riceve una richiesta, stampa il messaggio "LOGGED" sul terminale. Quindi, come aggiungere condizioni più specifiche a quando viene chiamato il middleware? Guarda il prossimo esempio e vedi.

middleware requestTime

Creiamo il middleware che aggiunge una proprietà chiamata requestTime all'oggetto richiesta.

```
var requestTime = function (req, res, next) {
    req.requestTime = Date.now();
    next();
};
```

Ora modifichiamo la funzione di registrazione dall'esempio precedente per utilizzare il middleware requestTime.

```
myLogger = function (req, res, next, requestTime) {
    console.log('LOGGED at ' + requestTime);
    next();
};
```

Aggiungiamo il middleware alla nostra app:

```
var express = require('express');
var app = express();

myLogger = function (req, res, next) {
    console.log('LOGGED at ' + req.requestTime);
    next();
};

var requestTime = function(req, res, next) {
    req.requestTime = Date.now();
    next();
};

app.use(requestTime);

app.use(myLogger);

app.get('/', function(req, res) {
    res.send('Hello World!');
});

app.listen(3000);
```

Ora l'app registrerà l'ora in cui è stata effettuata la richiesta. Questo copre le basi della scrittura e l'utilizzo del middleware Express. Per ulteriori informazioni, consultare Utilizzo del middleware espresso .

!!! TODO: Creare utilizzando sezione middleware Express !!!

Middleware CORS

Questo esempio dimostra come una richiesta HTTP di origine incrociata può essere gestita utilizzando un middleware.

Sfondo CORS

CORS è un metodo di controllo degli accessi adottato da tutti i principali browser per scongiurare vulnerabilità cross scripting intrinseche. In generale la sicurezza del browser, gli script dovrebbero mantenere che tutte le richieste XHR devono essere fatte solo per l'origine da cui sono serviti gli stessi script. Se viene effettuata una richiesta XHR al di fuori del dominio a cui appartengono gli script, la risposta verrà respinta.

Tuttavia, se il browser supporta CORS, farebbe un'eccezione a questa regola se le intestazioni

appropriate nella risposta indicano che il dominio da cui ha origine la richiesta è consentito. L'intestazione seguente indica che qualsiasi dominio è consentito:

```
Access-Control-Allow-Origin: *
```

Esempio

L'esempio seguente mostra come il middleware Express può includere queste intestazioni nella sua risposta.

```
app.use(function(request, response, next){

    response.header('Access-Control-Allow-Origin', '*');
    response.header('Access-Control-Allow-Methods', 'GET,PUT,POST,DELETE,OPTIONS');
    response.header('Access-Control-Allow-Headers', 'Content-Type, Authorization, Content-
Length, X-Requested-With');

   //Handle Preflight
   if (reqest.method === 'OPTIONS') {
      response.status(200).send();
   }
   else {
      next();
   }

});
```

Gestione preliminare

L'ultima parte dell'esempio precedente gestisce Preflight. La verifica preliminare è un'opzione speciale richiesta dal browser per testare CORS se la richiesta contiene intestazioni personalizzate.

Riferimenti utili

MDN - Esercitazione su CORS Http

Leggi Scrivere il middleware espresso online: https://riptutorial.com/it/express/topic/6993/scrivere-il-middleware-espresso

Capitolo 11: Spiega il routing in Express

Examples

Express Router

Il router Express consente di creare più "mini app" in modo da poter assegnare un nome allo spazio, pubblico, auth e altri percorsi in sistemi di routing separati.

```
var express   = require('express');
var app       = express();
var router    = express.Router();

router.get('/', function(req, res){
   res.send('Get request received');
});

router.post('/', function(req, res){
   res.send('Post requestreceived');
});

app.use('/', router);

app.listen(8080);
```

Gestori di itinerari concatenabili per un percorso di instradamento utilizzando app.route

```
var express   = require('express');
var app       = express();
var router    = express.Router();

app.route('/user')
  .get(function (req, res) {
    res.send('Get a random user')
  })
  .post(function (req, res) {
    res.send('Add a user')
  })
  .put(function (req, res) {
    res.send('Update the user details')
  })
  .delete(function (req, res) {
    res.send('Delete a user')
  });
```

Leggi Spiega il routing in Express online: https://riptutorial.com/it/express/topic/6536/spiega-il-routing-in-express

Capitolo 12: usando https con express

Examples

Usare https con express

Per prima cosa devi generare chiavi pubbliche e private usando OpenSSL (tutorial).

```
var express = require("express");
var http =require ("http");
var https=require ("https");
var fs=require("fs");
var app=express();
var httpsKeys={
key:fs.readFileSync("<key.pem>");
crtifcte:fs.readFileSync("<certificate.pem>");
};
http.createserver(app).listen(3000);
https.createserver(httpsKeys,app).listen(3030);
```

Leggi usando https con express online: https://riptutorial.com/it/express/topic/7844/usando-https-con-express

Capitolo 13: Visualizza la configurazione del motore

introduzione

Spesso il server ha bisogno di servire le pagine in modo dinamico. Per un esempio un utente Mr.X visita la pagina e vede alcune cose come "Benvenuto Mr. X alla mia homepage". In questo caso le visualizzazioni possono essere utili. Anche per popolare una vista tabella può essere a portata di mano. Le variabili possono essere iniettate in HTML in modo dinamico utilizzando il motore di visualizzazione. Il motore di visualizzazione è qualcosa che esegue il rendering delle viste. È possibile mantenere le visualizzazioni in una cartella denominata view e servire su richiesta. Il percorso della cartella può essere mostrato in Express utilizzando il percorso. risolvere il metodo.

Osservazioni

installa ejs usando il seguente (so che è ovvio)

```
sudo npm install ejs --save
```

Examples

1: impostazione delle viste

```
var express=require("express");    //express is included
var path=require("path");    //path is included

var app=express();    //app is an Express type of application

app.set("views",path.resolve(__dirname,"views"));    //tells express about the location of the
views in views folder
app.set("view engine","ejs");    //tells express that ejs template engine is used
```

2.Esempio di file EJS (fare riferimento a 1.impostare ... prima di questo)

il seguente è un file EJS.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Hello, world!</title>
    </head>
    <body>
        <%= message %>
    </body>
</html>
```

3. visualizzazione del reso con express (consultare il file 2.EJS ... prima di questo)

```
app.get("/",function(req,res){
response.render("index",{                  //render the index when root(/) is requested
    message:"rendered view with ejs"
        });
});
```

4.dopo aver creato il codice HTML finale (fare riferimento a 3.rendering ... prima di questo)

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Hello, world!</title>
</head>
<body>
    message:"rendered view with ejs"
</body>
</html>
```

Leggi Visualizza la configurazione del motore online:
https://riptutorial.com/it/express/topic/8104/visualizza-la-configurazione-del-motore

Titoli di coda

S. No	Capitoli	Contributors
1	Iniziare con express	Akshay Khale, Community, David Vogel, Dima Grossman, dkimot, Everettss, Gregory Worrall, Guillaume Lrv, Jared Hooper, jawadhoot, Kilmazing, Random User, Sumner Evans, user6939352
2	Collegare	Henrik Karlsson, Overflowh
3	Come funziona ExpressJs	rocketspacer
4	Express Database Integration	dkimot
5	Express-generatore	rickrizzo, Rupali Pemare
6	Gestione degli errori	gevorg, jawadhoot, Kilmazing
7	Gestione di file statici	dkimot, Sujithrao
8	Registrazione	Mor Paz
9	Routing	jawadhoot, Kelvin, phobos, S.L. Barth, zurfyx
10	Scrivere il middleware espresso	Charlie H, dkimot
11	Spiega il routing in Express	Dima Grossman, Sujithrao
12	usando https con express	nilakantha singh deo
13	Visualizza la configurazione del motore	Daniele Giussani, nilakantha singh deo