Introduzione a Node.js

Angelo Di Iorio

Università di Bologna

ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI

Node.js

- Node.js è un **ambiente di esecuzione Javascript** progettato per costruire applicazioni **server-side efficienti**
- Nato da un progetto individuale (di Ryan Dahl nel 2009) è sviluppato con un modello di governance aperto sotto la guida della Node.js Foundation
- E' alla base di un vasto ecosistema di moduli software e supportato da una vasta comunità open-source
- Questo ecosistema è gestito tramite *npm* (*Node Package Manager*)
- Si basa sul motore JavaScript V8, open-source, lo stesso motore usato da Google Chrome
- *Javascript full stack*

IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI

Hello World

hello.js

```
console.log("Are you kidding me?")
```

```
node hello.js
```

app.js

```
http = require("http")
console.log("Let's play with a web server…")
http.createServer(
  function (request, response) {
      response.write('Hello World!');
      response.end();
      console.log('One more request');
  }
).listen(8099);
```

```
node app.js
```


IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI

Esercizio 0

- Scrivere uno script Node.js da linea di comando che stampa in console il numero di parametri ricevuto in input

  `countParams.js 715` stampa `1`
  `countParams.js a.txt b.txt` stampa `2`
  `countParams.js` stampa `0`

- Nota: l'oggetto `process` rappresenta il processo in esecuzione e l'oggetto `process.argv` il vettore di parametri di input

- **Node.js è a tutti gli effetti JavaScript, si possono usare tutti i costrutti visti finora**

IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI

Aspetti principali

- Vediamo alcuni aspetti di Node.js:
  - Modello di esecuzione: single threaded, con operazioni di I/O non bloccanti
  - Moduli
  - Estensioni e gestione dei pacchetti via npm

IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI

Single Thread e Event Loop

- Node.js utilizza **un unico thread** per tutte le richieste ricevute, con un notevole guadagno in termini di efficienza
  - Ridotto il tempo di context-switch da una richiesta all'altra
  - Si usa meno memoria, si possono ricevere molte più richieste in parallelo

- Per ottenere questo risultato sfrutta **Event Loop** e l'asincronicità di Javascript:
  - Tutte le richieste sono gestite da un solo processo
  - Questo processo passa i task da eseguire (*work items*) ad altri worker che lavorano in background attraverso la registrazione di **funzioni di callback:** registrata una callback quindi l'esecuzione continua e viene processata la richiesta successiva
  - quando un worker termina la sua esecuzione il thread principale viene notificato (tramite eventi) e la callback invocata

IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI

Event Loop (1)

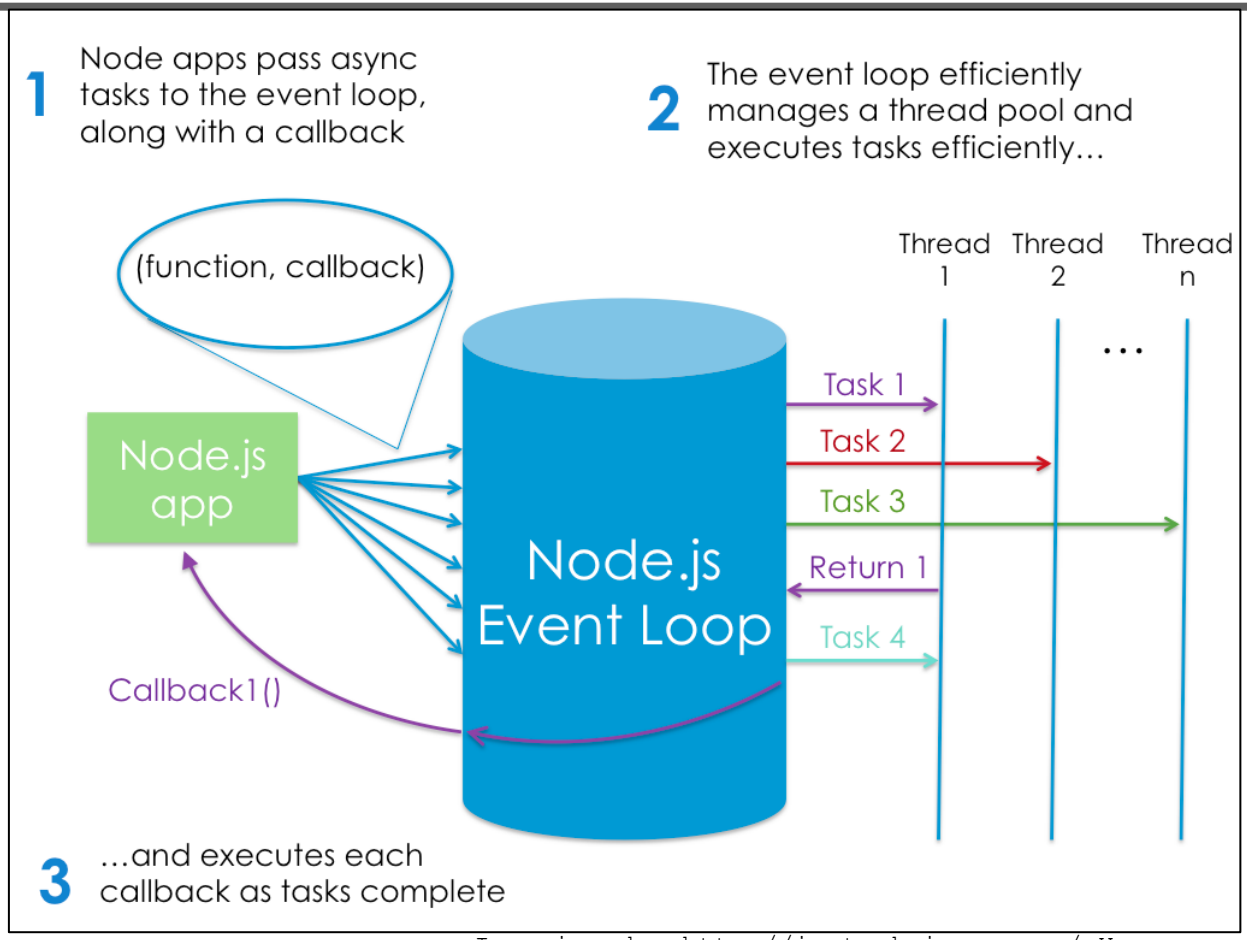


Immagine da: http://i.stack.imgur.com/zYgcr.png

IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI

Event Loop (2)

- Event Loop non è una libreria esterna ma un elemento *core* del motore Node.js
- Node.js entra nell'Event Loop quando viene eseguito lo script ed esce quando non ci sono più funzioni di callback pendenti da invocare
- Internamente usa la libreria *libuv* for operazioni di I/O asincrone, che a sua volta mantiene un pool di thread
- La comunicazione avviene tramite **emissione di eventi**
- Gli oggetti che possono generare eventi (Event Emitter) espongono una funzione che permette di associare altre funzioni ad eventi che quell'oggetto genererà

```
eventEmitter.on(eventName, listener)
```

- Al verificarsi dell'evento queste funzioni saranno invocate in modo sincrono

IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI

Funzioni sincrone vs. asincrone

```
[…]
var queryResult = db.query("select * form data");
console.log("Done.");
```

```
[…]
db.query("select * form data", function(rows) {
var result = rows;
…
});
console.log("Done? Maybe yes, maybe no…");
```


IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI

Funzioni asincrone e callback

- Node.js è quasi esclusivamente basato su **funzioni asincrone** e **callback**.
- La **convenzione** suggerisce di creare funzioni che accettano una funzione **callback asincrona come ultimo parametro**
- La funzione callback per **convenzione** prende in input come **primo parametro un oggetto che contiene l'errore** (o meglio un oggetto in cui la funzione che chiamerà la callback avrà memorizzato l'errore)
- Resta il problema di ***callback hell***
- Siamo in JS e possiamo usare le tecniche per gestire operazioni asincrone: promesse, async/await, ecc.

IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI

Callback e gestione errori

```
var fs = require("fs");

fs.readFile(process.argv[2], function (err, data) {
   if (err){
      console.log("Wrong path?");
      return;
   }
   console.log(data.toString());
});
```


IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI

Moduli Node.JS

- I moduli in Node.js permettono di includere altri file JS nelle applicazioni e di riusare librerie esistenti
- Favoriscono l'organizzazione del codice in parti indipendenti e riutilizzabili
- L'enorme quantità di moduli disponibili gratuitamente ha contribuito al successo di Node.js
- Node.js ha un sistema di caricamento dei moduli semplice ma potente:
  - un modulo è un file Javascript
  - quando si include un modulo, questo viene cercato **localmente** o **globalmente**
- Per includere un modulo si usa la keyword `require(<modulo>)`

IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI

Caricare moduli (1)

- Esistono tre tipi di moduli:
  - **core:** built-in nel sistema, non è necessario installarli separatamente
  - **dipendenze locali:** installati per l'applicazione corrente nella directory `./node_modules/`
  - **dipedenze globali:** disponibili per tutte le applicazioni e installati nelle directory globali specificate nella variabile d'ambiente `NODE_PATH`
- Un modulo può essere caricato specificando il percorso o il nome:
  - `foo = require('./lib/bar.js');`
  - `foo = require('bar')`
- L'interprete cerca il modulo tra quelli *core*, poi tra le dipendenze locali e poi globali
- Le dipendenze locali e globali si installano via *npm* (vedi prossime slide)

IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI

Caricare moduli (2)

```
http = require("http")

fs = require(”redis")

require("./greetings.js")


console.log("You just loaded a lot of modules! ")
```

built-in

dipendenza (installata con npm)

file locale

IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI

Riusare moduli

- I moduli sono eseguiti in uno ***scope*** indipendente
- Questo permette di evitare conflitti e di creare librerie facilmente riutilizzabili
- I moduli inoltre possono essere assegnati a variabili sulle quali invocare i metodi che il modulo espone

```
greetings = require("./greetings.js")
jsonDiff = require("json-diff")


greetings.hello()
jsonDiff.diff({ greet: 'Hello' }, { greet: 'Mataste' })
```

Attenzione: deve essere installato

IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI

Creare moduli

- Abbiamo bisogno di un meccanismo per rendere gli oggetti visibili dall'esterno del file (lo scope è interno al file/modulo)
- Si usa un oggetto speciale **`module`** che rappresenta il modulo corrente
- In particolare l'oggetto **`module.exports`** contiene tutto ciò che il modulo espone pubblicamente
- Aggiungere una funzione (o altri oggetti) a **`module.exports`** vuol dire quindi renderlo pubblico e accedibile dall'esterno
- Questo stesso meccanismo è usato, insieme a IIFE, dai moduli che installiamo via *npm*

IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI

Creare un modulo

```
greetings = require("./greetings.js")
greetings.hello()
greetings.hindi()
```

greetings.js

```
hello = function() {
        console.log("\n Hello! \n")
}
hindi = function() {
        console.log("\n Mataste! \n")
}
module.exports.hello = hello;
module.exports.hindi = hindi;
```


IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI

npm

- I moduli di node.js vengono distribuiti ed installati con npm (*node package manager*)
- npm viene eseguito via command-line e interagisce con il *registro npm*
  - meccanismo robusto per gestire dipendenze e versioni dei pacchetti (moduli)
  - semplice processo di pubblicazione di pacchetti e condividerli con altri utenti.
- Una piattaforma per repository pubblici e privati, e servizi enterprise e di integrazione con altri sistemi

IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI

Creare un pacchetto per *npm*

- Un pacchetto *npm* è un insieme di file Node.js, tra cui il file manifest **`package.json`** che specifica alcuni metadati del pachetto, tra cui *nome*, *autore*, *dipendenze*, ecc.
- Il seguente comando eseguito nella directory che contiene gli script permette di creare il manifest attraverso un'interfaccia testuale interattiva:
  - **`npm init`**

```
{
  "name": "My app",
  "version": "1.0.0",
  "description": "Nodeplayground",
  "main": "myapp",
  "author": "Angelo Di Iorio",
  "license": "MIT"
}
```

IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI

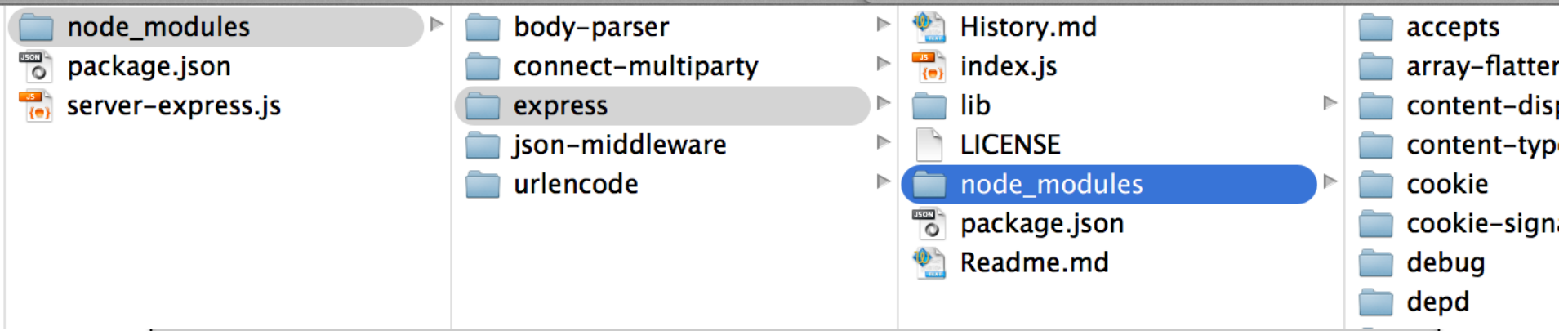
Installare un pacchetto

- Una volta creato un package si possono installare le dipendenze locali:
  - **`npm install express`**
- Il comando modifica il file *package.json*
- Di default i pacchetti sono installati nella directory locale:
  - **`./node_modules/`**
- Note:
  - nelle versioni precedenti alla 5.0 era necessario specificare il parametro **`-save`** per installare i pacchetti in questa directory, ora non è più necessario
  - l'opzione **`-g`** permette invece di installarli globalmente (ovviamente ha bisogno dei permessi)

IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI

package.json

```
{
  "name": "server-express",
  "version": "1.0.0",
```




```
  "dependencies": {
    "body-parser": "^1.15.2",
    "connect-multiparty": "^2.0.0",
    "express": "^4.14.0",
    "json-middleware": "^1.0.2",
    "urlencode": "^1.1.0"
  }
}
```


IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI

Alcuni comandi per gestire i pacchetti

- `npm list`
- `npm config list`
- `npm search express`
- `npm install express`
- `npm install express -global`
- `npm install express@1.0.2`
- `npm uninstall express@1.0.3`
- `npm update express`
- `npm init`
- `..`

ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA
IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI

Conclusioni

- Oggi abbiamo visto le caratteristiche salienti di Node.js
- Esiste una vastissima offerta di pacchetti (moduli) disponibili via npm
- Tra questi:
  - **`express`**: framework server-side per applicazioni Node.js
  - **`mondogodb`**: driver per usare un database
- L'installazione e l'estensione di questi pacchetti richiederà a sua volta di installare altri pacchetti
- Siete invitati a sperimentarne altri e installarli nel vostro progetto se lo ritenete utile

IL PRESENTE MATERIALE È RISERVATO AL PERSONALE DELL'UNIVERSITÀ DI BOLOGNA E NON PUÒ ESSERE UTILIZZATO AI TERMINI DI LEGGE DA ALTRE PERSONE O PER FINI NON ISTITUZIONALI