Tecnologie e applicazioni web

Node.js

Filippo Bergamasco (filippo.bergamasco@unive.it)
http://www.dais.unive.it/~bergamasco/
DAIS - Università Ca'Foscari di Venezia
Anno accademico: 2017/2018

Server-side Javascript

Storicamente, realizzare la parte server di un'applicazione web risulta più complesso e tedioso perché richiede una conoscenza più o meno approfondita di:

- Programmazione multi-threaded
- Scalabilità
- Gestione della sicurezza
- Server deployment
- .. etc ...

Server-side Javascript

Javascript non era stato pensato per offrire caratteristiche e performance simili ad altri linguaggi comunemente usati in ambito server:

- Sistema di gestione della memoria basilare
- Mancanza di integrazione con il sistema operativo
- Velocità di esecuzione molto ridotta (il linguaggio veniva interpretato direttamente dal browser)

Prima dell'avvento di V8, è rimasto per anni confinato all'interno del browser

Node.js

Node.js è un **runtime Javascript** composto dal motore V8 di Google unito ad un **layer di integrazione** con il sistema operativo che fornisce un ambiente Javascript full-featured al di fuori del browser.

- Open-source
- Orientato allo sviluppo di codice Server-side
- Cross-platform
- Ideale per lo sviluppo di applicazioni di rete

Perchè usare Node.js?

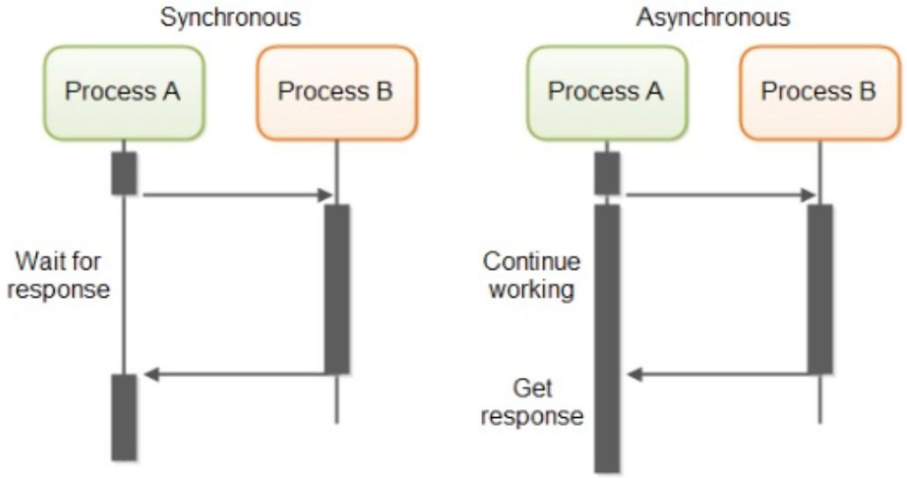
- E' molto leggero e veloce rispetto alle soluzioni "tradizionali" basate su Java, PHP, etc.
- Facile da configurare e da installare
- Ci sono moltissimi moduli (librerie) disponibili gratuitamente e semplicemente installabili attraverso il gestore pacchetti **npm**
- Contiene moduli per la connessione a database relazionali e non (NoSQL)
- Permette di utilizzare Javascript come unico linguaggio dell'intera web application (sia server che client)

Tre punti chiave di Node.js

1. Operazioni asincrone (non-bloccanti)
2. Single threaded
3. Shared-state concurrency

Operazioni asincrone

In Nodejs la maggior parte delle funzionalità offerte dalle API sono di tipo non-bloccante ed eseguono i rispettivi task in modo asincrono

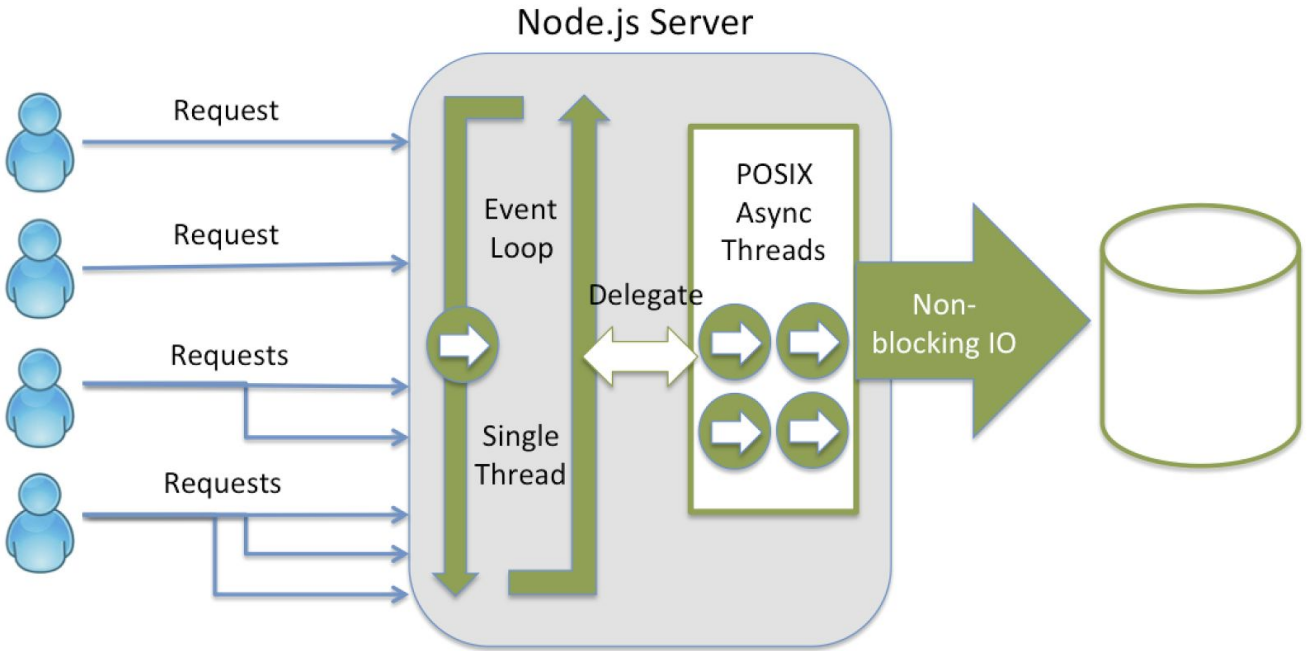


Single threaded

In Nodejs le operazioni sono asincrone ma **non parallele**. Vi è un solo event-loop (come nel browser) che processa a turno tutte le callbacks.

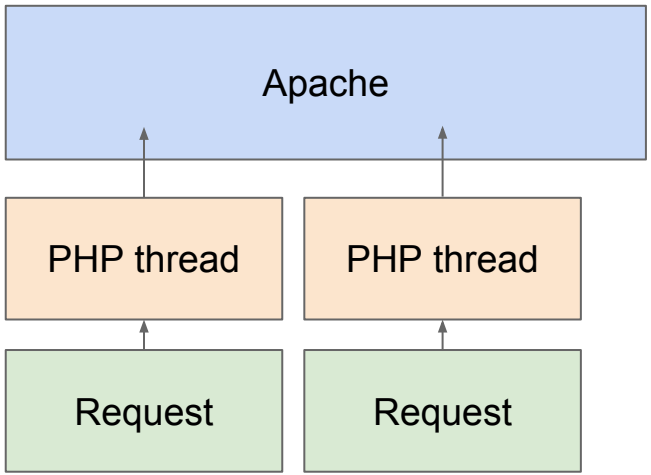
Internamente Nodejs gestisce un thread-pool e IO non bloccante per le operazioni

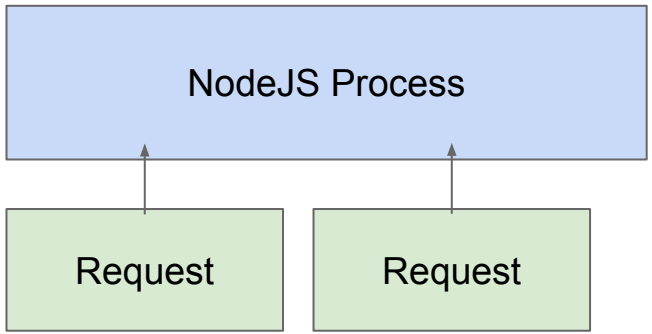
Single threaded



Shared-state concurrency

Tutti gli accessi concorrenti vedono lo stesso stato condiviso della memoria (Non vi è l'overhead dovuto alla creazione/distruzione di processi o thread)





Scenari di utilizzo

Quando conviene usare Node.js?

Quando il programma è IO-bound, cioè la maggior parte del tempo viene speso per la gestione dell'IO (dal disco, dalla rete, dal database) anziché per l'esecuzione di calcoli complessi

Caso tipico dei server Web, database, etc!

Scenari di utilizzo

Quando NON conviene usare Node.js?

Per la sua natura single-threaded, non è la scelta più adatta se il software è CPU-bound:

- Le richieste che vengono effettuate al server richiedono l'esecuzione di task molto complessi e cpu-intensive
- Le operazioni da effettuare sono naturalmente parallelizzabili in un ottica multi-thread

Installazione

Node è disponibile per Windows, OSX e Linux.
http://nodejs.org

Su Linux è preferibile l'installazione attraverso il gestore pacchetti della distribuzione, su OSX attraverso homebrew

Tools forniti

Node.js viene distribuito con:

- Un front-end REPL (Read, Evaluate, Print, Loop)
- Un eseguibile da riga di comando per l'esecuzione di script da files
- Un gestore pacchetti chiamato npm (Node Package Manager)
  - Gestisce l'installazione di moduli aggiuntivi
  - Gestisce e risolve le dipendenze fra i moduli
  - Installa e gestisce il path delle utilities da riga di comando

Node REPL

```
$ node --version
v8.9.4
$ node
> var a="filippo"
undefined
> console.log(a)
filippo
undefined
```

Codice e moduli

E' possibile semplicemente eseguire codice Javascript contenuto in un file lanciando il processo

```
$ node nomefile.js
```

Un progetto mediamente complesso richiede solitamente di strutturare le varie componenti in files distinti. L'entry point è definito nel file **package.json**, congiuntamente a metadati sulla versione, descrizione, etc

package.json

https://docs.npmjs.com/files/package.json

```
{
   “name”: “nome-mio-progetto”,
   “main”: “nome-script-principale”,
   “version”: “0.0.1”,
   “dependencies”: {
      “colors”: “0.5.0”
   },
   “private”: “true”
}
```

Moduli

Un package Node, definito in package.json, permette di definire un **modulo** che può essere eseguito (se è presente la proprietà main) oppure utilizzato da altri moduli come libreria.

Il framework Node.js comprende un gestore pacchetti chiamato **npm** che permette di gestire le dipendenze fra i pacchetti, semplificandone l'installazione

Moduli

I moduli seguono la convenzione definita in CommonJS:

- La funzione require() permette di caricare moduli globali (installati via npm) e locali
- Tutte le variabili definite all'interno di un modulo (file) sono locali per quel modulo.
- Per esportare variabili al di fuori del modulo vanno aggiunte all'interno dell'oggetto module.exports o semplicemente exports

npm

Per installare tutte le dipendenze di un dato modulo, definito in package.json, è possibile utilizzare il comando:

```
$ npm install
```

Viene creata una directory `node_modules` con il codice di tutte le dipendenze. La directory è posizionata di default nella stessa dir contenente il file package.json

npm

Il proprio modulo può essere pubblicato nel registro npm (se non è stato marcato come private) con il comando

```
$ npm publish
```

E' possibile cercare fra i moduli disponibili con il comando

```
$ npm search <nomemodulo>
```

L'oggetto global

In Node vi sono due oggetti globali, chiamati rispettivamente **global** e **process**, con funzionalità simili (dato che node è singolo processo). Tra le proprietà più utili ricordiamo:

- __filename, __dirname
- setImmediate(callback[, ...args])
- setInterval(callback, delay[, ...args])
- setTimeout(callback, delay[, ...args])
- require()

API

Fornendo un interprete javascript al di fuori del browser, Node.js espone molte API per interfacciarsi con le funzionalità offerte dal sistema operativo:

- Accesso al filesystem
- Creazione di socket TCP/UDP
- Esecuzione/controllo di altri processi

https://nodejs.org/api/index.html

API

La maggior parte delle API sono di tipo asincrono e orientate alla gestione rapida dell'IO.

Esempio: filesystem API:

```
var stream = fs.createReadStream('my-file.txt');
stream.on('data', function(chunk){
   // do something with part of the file
});
stream.on('end', function(chunk){
   // reached the end
});
```

HTTP

Una delle API più utilizzate di Node sono quelle che forniscono client e server HTTP

Basato sul concetto di requests e responses, che sono gestite da Node rispettivamente dagli oggetti: http.ServerRequest e http.ServerResponse

- Operazioni di ricezione di request e invio di response asincrone per la gestione di più utenti