ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Framework Javascript

Seconda parte

Fabio Vitali

Corsi di laurea in Informatica e
Informatica per il Management
Alma Mater – Università di Bologna

Qui parleremo di…

Alcuni framework:

- *JQuery*

- Node + Express

- *Il problema dei template*
- *Moustache e Handlebar*
- *Angular*
- *React*
- *Vue*

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Node.js, npm ed Express.js

[XXX] across the stack

- Lo stack è l'insieme degli ambienti di cui un progettista web deve tener conto per realizzare un'applicazione, e dei linguaggi di programmazione da conoscere.
- Java across the stack:
  - *Google Web Toolkit*: ambiente integrato client-server, in cui la parte client viene compilata in Javascript da un sorgente Java
- Javascript across the stack:
  - *ASP* e poi *ASP.NET*: soluzione Microsoft per IIS. ASP era solo Javascript, mentre ASP.NET permette di scegliere tra molti linguaggi, incluso C#, VBA e Javascript.
  - *Google Apps Script*: soluzione Google per fornire comportamenti sofisticati ai siti realizzati con Google Sites.
  - *MongoDB, CouchDB*: database NoSQL forniscono meccanismi di interrogazione e gestione dei dati via Javascript
  - NodeJS: ambiente di esecuzione server-side di applicazioni sofisticate (anche indipendentemente da HTTP).

MEAN stack

- Un termine che adesso è MOLTO di moda è MEAN stack:
  - MongoDB: un database NoSQL che fornisce permanenza ai dati delle applicazioni
  - ExpressJs: un framework per applicazioni web
  - AngularJs: un framework MVC per la creazione di template sofisticati di pagine HTML
  - NodeJs: una piattaforma software per applicazioni server-side
- Lo scopo evidente dello stack MEAN è di ripetere il successo e sostituirsi allo stack LAMP che ha costituito il set fondamentale di tecnologie per lo sviluppo di applicazioni web negli anni 2000.

NodeJS e npm

Creatura di una persona sola, Ryan Dahl (2009), per fornire bi-direzionalità alle applicazioni web (tecnologia *push*).

Un'applicazione nativa server-side, che permette di fare chiamate I/O guidate da eventi, non-bloccanti.

Per uniformare la realizzazione di applicazioni miste server/client, decide di usare un motore Javascript anche server-side

- Non è la prima volta, già Microsoft con ASP usa Javascript server-side
- Utilizza un interprete di Javascript molto potente, V8 di Google

Ad esso viene aggiunto un package manager, ***npm***, che permette di installare velocemente librerie e pacchetti associati.

Single-thread non-blocking execution

Invece di dedicare un thread ad ogni connessione ricevuta dalla rete, l'applicazione nodeJS utilizza un solo thread in cui tutte le richieste ricevono attenzione condivisa.

- In un'applicazione Web tradizionale un thread può avere circa 2Mb di memoria associati, quindi una macchina con 8GB ha un massimo teorico di 4000 richieste concorrenti.
- Mantenendo tutto in un unico stack, NodeJS arriva ad un massimo teorico di 1 milione di richieste concorrenti.

Poiché ogni richiesta può richiedere tempi variabili, è prassi inserire richieste onerose in funzioni asincrone, non-bloccanti, con una funzione di callback al completamento.

- Questo permette al thread principale (main loop) di rimanere molto snello e veloce.
- Per contro, è facilissimo inchiodare NodeJS facendo eseguire parti onerose del codice direttamente dentro al main loop.

NodeJS come motore HTTP

NodeJS può elegantemente sostituire PHP, Python, Perl e ogni altro approccio cgi-bin tradizionale. Di seguito un esempio di un'applicazione che restituisce un frammento HTML:

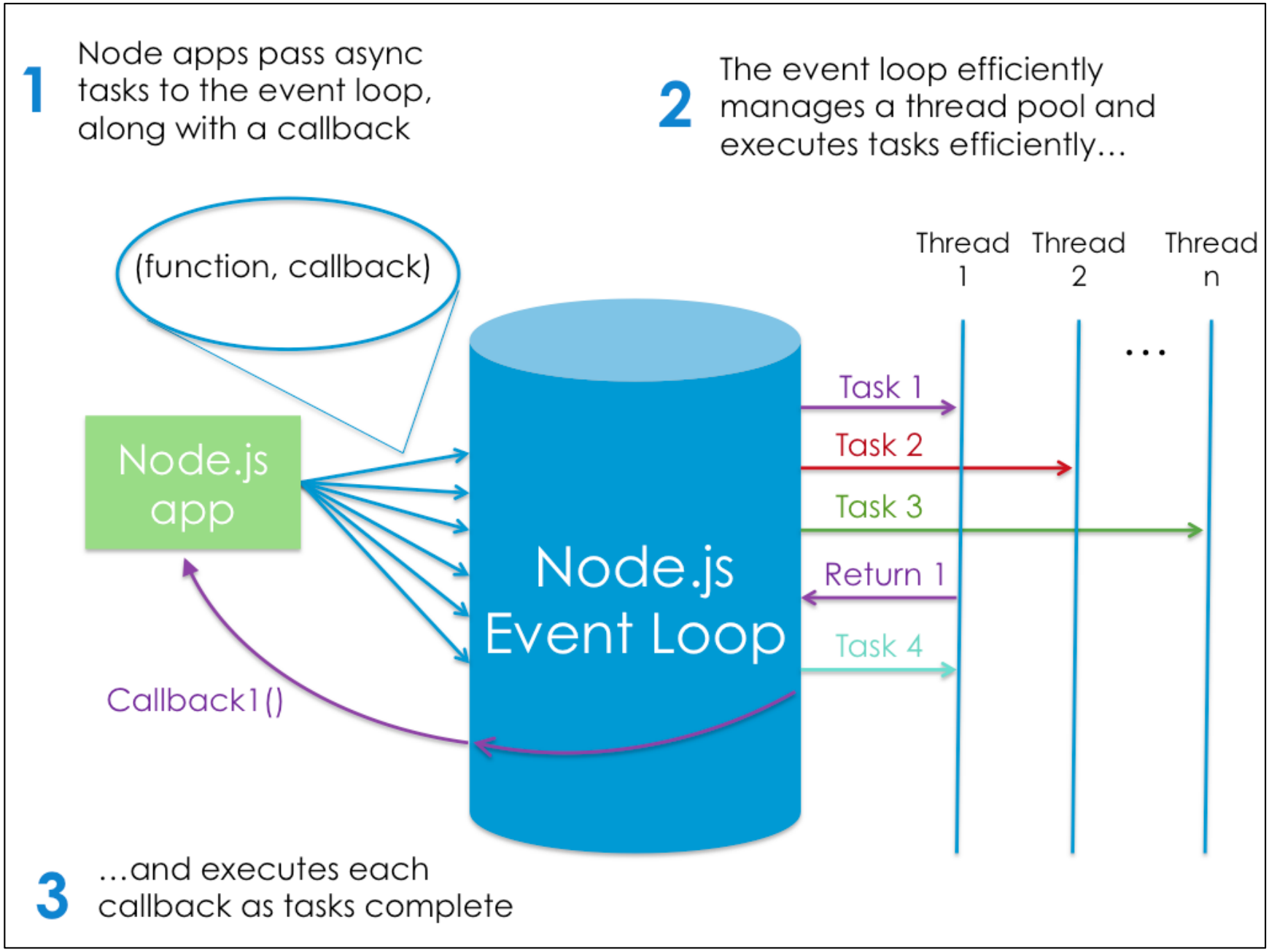
```
var http = require('http');

http.createServer(
   function (request, response) {
      response.writeHead(200, {'Content-Type': 'text/html'});
      response.end('<h1>Hello World</h1>');
   }
).listen(8000);
```

NodeJS con altri protocolli

NodeJS può porsi in ascolto su qualunque porta e implementare anche protocolli diversi da HTTP:

```
var net = require('net');
net.createServer(
  function (stream) {
    stream.write('hello\r\n');
    stream.on('end',
      function () {
        stream.end('goodbye\r\n');
      }
    );
    stream.pipe(stream);
  }
).listen(8000);
```

Event Loop



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

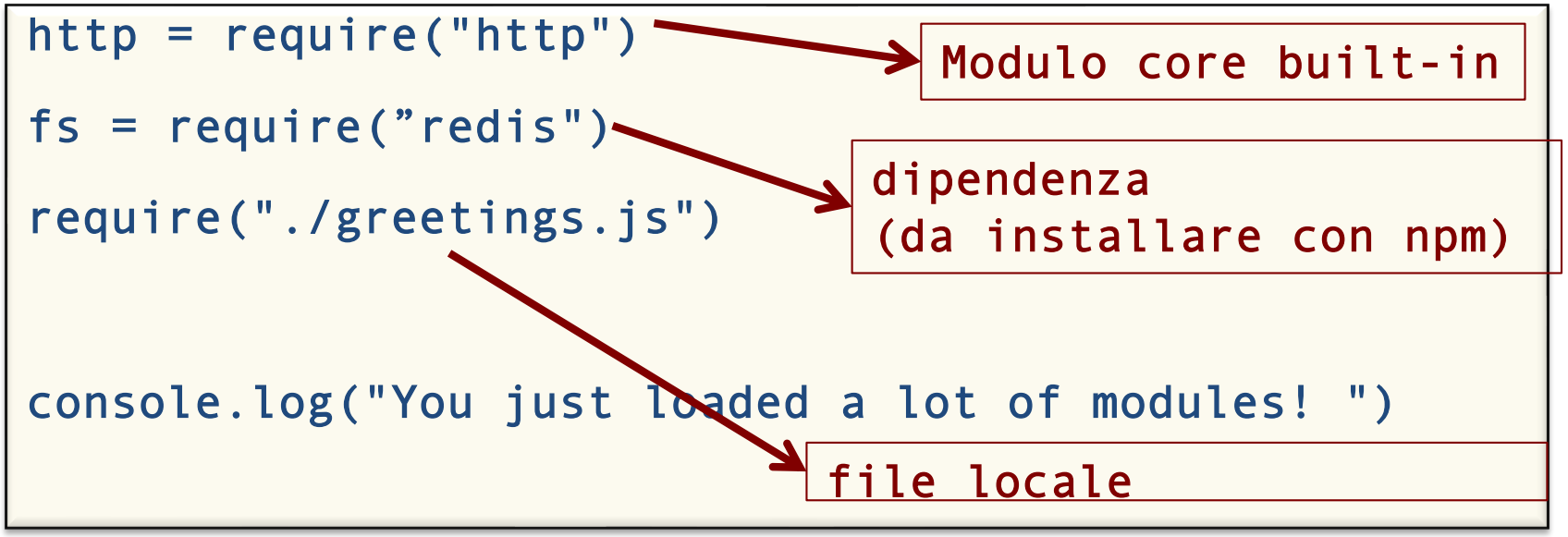
Asincronia e funzioni callback

- Node.js è quasi esclusivamente basato su funzioni asincrone e callback.
- La convenzione di node.js suggerisce di creare funzioni che accettano una funzione callback asincrona come ultimo parametro
- La funzione callback per convenzione usa la sintassi error-first

```
fs.readFile('test.txt', function(err, data) {
  if(err) {
    console.log('Error: '+error.message);
    return;
  }
  doSomething(data);
});
```

I moduli di node.js

- Node.js è estremamente modulare. E' possibile mettere codice indipendente in file javascript ed eseguirlo secondo necessità.
- Il meccanismo di caricamento di moduli di Node.js è semplice:
  - un modulo è un file Javascript.
  - Quando si richiede un modulo, esso viene cercato localmente o globalmente secondo un modello preciso.
- I moduli vengono caricati con `require()`.



Creare un modulo

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

greetings.js

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

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.
- In più: una piattaforma per repository pubbblici e privati, servizi enterprise (integrati con firewall aziendali), integrazione con altre piattaforme, ecc.

Express.js

- Express è un modello di server web per node.js
  - Open-source, licenza MIT
  - molto usato e molto supportato dalla comunità
  - molto semplice e molto espandibile con plugin
- Express si dedica al routing, alla gestione delle sessioni, all'autenticazione degli utenti, e molti altri principi fondanti delle connessioni HTTP.
- Nato nel 2010, acquistato nel 2015 da IBM, poi regalato da IBM alla Node.js Foundation.

Basic server

```
express = require("express")
app = express()

docs_handler = function(request, response){
        var docs = {server : "express”}
        response.json(docs);
}

app.get("/docs", docs_handler);

app.listen(8099, function(){
     console.log("\nExpress is running!!! \n")
})
```

Routing

Express fornisce una semplice interfaccia per fare routing:

```
app.method(path, function(request, response) { ... })
```

dove

- method è uno dei metodi HTTP (get, post, put, delete, ecc.)
- path è il local path dell'URI richiesto al server
- function(req, res) è un handler da eseguire quando viene richiesto il path. I due parametri contengono gli object della richiesta HTTP (uri, intestazioni, parametri, dati, ecc.) e della risposta HTTP in via di restituzione (intestazioni, dati, ecc.)

Ogni route gestisce una o più handler anche sugli stessi path.

```
app.post('/', function (req, res) {
  res.send('<p>Richiesta POST');
});
```

Accedere ai dati di un POST

- La libreria bodyparser (integrata con express) permette di accedere al corpo dei dati spediti dal POST nel body della richiesta.
- Questi middleware si occupano di recuperare e elaborare i dati sottomessi da un form via post e li convertono in oggetti accedibili dall'applicazione.
- I dati del POST si accedono come:

```
<request>.body.<post_variable>
```

POST data

```
var express = require("express")
var bodyParser = require('body-parser');

app = express()

app.use(bodyParser());

app.post('/login',function(request,response){
  console.log("Username:" + request.body.user)
  console.log("Password:" + request.body.pwd)

 // Continue and do authentication….
});
```

‘user’ / ‘pwd’:
nomi degli input usati nel form

Express middleware stack

- Express ha pochissime funzionalità proprie (routing), ma utilizza un grande numero di librerie middleware grandemente personalizzabili in uno stack di servizi progressivamente più complessi.
- Per aggiungere un middleware allo stack:

  ```
  app.use(<middleware>)
  ```

- Un'applicazione Express è allora essenzialmente una sequenza di chiamate funzioni di middleware tra la richiesta e la risposta.
- Il middleware può
  - accedere agli oggetti di richiesta e risposta
  - cambiarli ed eseguire del codice di modifica
  - chiamare la prossima funzione del middleware (`next()`)
  - uscire dal ciclo e mandare la risposta.

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Fabio Vitali

Dipartimento di Informatica – Scienze e Ingegneria
Alma mater – Università di Bologna

Fabio.vitali@unibo.it

www.unibo.it