DOcumentazione nodejs

1. Introduzione

Node.js è un ambiente di esecuzione per JavaScript lato server, costruito sul motore V8 di Google Chrome. Questo permette di eseguire JavaScript al di fuori del browser, utile per creare server web, applicazioni backend, e strumenti di automazione.

* 1. Obiettivi del Corso

Al termine del corso, i partecipanti saranno in grado di:

* Comprendere le basi di Node.js e come funziona.
* Creare applicazioni web semplici utilizzando Express.js.
* Gestire pacchetti con npm (Node Package Manager).
* Interagire con file e directory.
* Lavorare con operazioni asincrone.

1. Installazione di Node.js
   1. 1.1. Requisiti

Assicurati di avere una versione recente di Node.js.

Puoi verificare la versione di Node.js sul tuo sistema aprendo un terminale ed eseguendo il comando: **node -v**

Se non lo hai ancora installato sul tuo pc, vai su [Node.js](https://nodejs.org) per scaricare l'ultima versione stabile (LTS) per il tuo sistema operativo.

* 1. Cos'è Node.js?

Node.js è un ambiente runtime JavaScript basato sul motore V8 di Chrome. È progettato per costruire applicazioni scalabili e veloci, in particolare applicazioni di rete, come server web.

* + 1. Caratteristiche principali
* **Event-driven (Basato su eventi**): Node.js è costruito attorno a un loop di eventi. In un'applicazione tradizionale, il server può diventare bloccato quando elabora richieste. In Node.js, le richieste vengono gestite in modo asincrono. Questo significa che il server può continuare a rispondere a nuove richieste mentre elabora altre, senza dover attendere il completamento di una singola operazione. Questo approccio è particolarmente vantaggioso per applicazioni che richiedono la gestione di molte connessioni simultanee, come server web e chat.
* **Non-blocking I/O (Operazioni di I/O non bloccanti**): Una delle caratteristiche più potenti di Node.js è la gestione asincrona delle operazioni di input/output (come lettura/scrittura di file, query a database, richieste HTTP). Con Node.js, queste operazioni non bloccano il flusso di esecuzione del programma, permettendo al server di rispondere rapidamente a nuove richieste mentre altre operazioni sono in corso. Ciò consente di ottenere alte prestazioni anche sotto carichi di lavoro elevati.
* **Single-threaded**: Sebbene Node.js sia monothread (utilizza un solo thread per eseguire il codice JavaScript), grazie al suo modello di I/O non bloccante, riesce a gestire in modo molto efficiente migliaia di richieste simultanee. Il loop degli eventi gestisce le operazioni in modo asincrono, senza necessità di creare nuovi thread per ogni richiesta, riducendo così il consumo di risorse.
* **Modularità con NPM**: Node.js utilizza un sistema di pacchetti chiamato npm (Node Package Manager), che consente di accedere a una vasta libreria di moduli open-source. Ogni modulo può essere facilmente installato e integrato nelle proprie applicazioni. npm rende facile aggiungere funzionalità al proprio progetto, come l'autenticazione, il supporto per il database, la gestione delle sessioni e tanto altro.
* **V8 JavaScript Engine**: Node.js sfrutta il motore JavaScript V8 di Google, lo stesso utilizzato in Chrome. V8 compila e esegue il codice JavaScript direttamente in linguaggio macchina, rendendo Node.js estremamente veloce nell'elaborazione delle operazioni.
* **Scalabilità**: Node.js è particolarmente adatto per costruire applicazioni scalabili. La gestione asincrona delle richieste e la sua architettura event-driven permette di aggiungere facilmente nuovi processi e nodi al sistema per migliorare le performance in ambienti ad alta domanda
  + 1. Uso di Node.js
* **Applicazioni Web**: Node.js è ideale per costruire server web, in particolare per applicazioni che richiedono prestazioni elevate e connessioni persistenti, come i server di chat in tempo reale (ad esempio, utilizzando WebSockets).
* **Microservizi**: Grazie alla sua modularità e alla facilità di gestire richieste parallele, Node.js è molto usato nella creazione di architetture basate su microservizi.
* **API RESTful**: Node.js è spesso utilizzato per creare API RESTful, che consentono alle applicazioni di comunicare tra di loro tramite richieste HTTP.
* **Applicazioni in tempo reale**: Node.js è perfetto per sviluppare applicazioni in tempo reale come giochi online, sistemi di messaggistica istantanea o collaborazioni in tempo reale grazie alla gestione di WebSocket e altre tecnologie simili.
  1. Gestione dei Pacchetti con npm

npm (Node Package Manager) è il gestore di pacchetti (dipendenze) per Node.js. Con npm, è possibile installare librerie e strumenti da un ampio ecosistema di pacchetti. Per iniziare un progetto Node.js, esegui: **npm init**

Questo comando ti guiderà nella creazione di un file `package.json`, che tiene traccia delle dipendenze del progetto.

Per scaricare e installare un pacchetto nel tuo progetto, usa il comando: **npm install <nome pacchetto>**

* + 1. File package.json

Il file package.json contiene tutte le informazioni riguardanti il progetto, comprese le dipendenze.

{

"name": "nome-del-progetto",

"version": "1.0.0",

"description": "Descrizione del progetto",

"main": "index.js",

"dependencies": {

**"<nome pacchetto>": "<versione (notazione SemVer major.minor.patch)>"**

}

}

* 1. Creare una semplice applicazione con Express.js

Express è un framework minimalista per Node.js che semplifica la gestione delle rotte e delle richieste HTTP.

Installa Express nel tuo progetto: **npm install express**

Ecco un esempio di come creare un server base con Express:

A computer screen with text

AI-generated content may be incorrect.

Salva questo file come `index.js` e avvia il server con: **node index.js**

Puoi aprire un browser e navigare all’indirizzo http://localhost:3000 per vedere il risultato.

* 1. Asincronia

Node.js utilizza un modello di I/O non bloccante, il che significa che può gestire più operazioni simultaneamente senza fermarsi. Questo è un elemento cruciale per la performance di applicazioni di rete.

* + 1. Event Loop

In Node.js, l'**event loop** è il meccanismo che gestisce l'esecuzione di codice asincrono. Poiché Node.js è **monothread** (cioè esegue un solo thread per volta), l'event loop permette di gestire operazioni come la lettura di file, le richieste HTTP, o operazioni di I/O senza bloccare l'esecuzione del programma.

Quando viene chiamata una funzione asincrona (ad esempio, una lettura di file), Node.js la mette in una coda, permettendo al programma di continuare a eseguire altre operazioni. Una volta completata l'operazione, la funzione di callback associata viene eseguita. Questo processo permette di gestire molte operazioni contemporaneamente, senza bloccare il flusso principale dell'applicazione.

* + 1. Uso delle callback (deprecate)

Le callback sono funzioni passate come argomenti ad altre funzioni e vengono chiamate quando l'operazione asincrona è completata. Vengono utilizzate per gestire risultati o errori delle operazioni asincrone. Tuttavia, l'uso eccessivo di callback può portare al fenomeno chiamato callback hell, che rende il codice difficile da leggere e mantenere

Un esempio semplice:

A screen shot of a computer program

AI-generated content may be incorrect.

In questo caso, la funzione di callback viene eseguita quando readFile ha finito di leggere il file, gestendo sia l'errore che il risultato.

* + 1. Uso delle Promesse

Le **promise** rappresentano un valore che potrebbe essere disponibile ora, in futuro o mai. Sono un modo più elegante di gestire il flusso asincrono rispetto alle callback, evitando il "callback hell". Una **promise** ha tre stati: **pending** (in attesa), **resolved** (risolta) e **rejected** (rifiutata). Puoi usare i metodi .then() e .catch() per gestire il risultato (il codice al suo interno verrà eseguito quando la promessa sarà risolta) o l'errore (il codice al suo interno verrà eseguito quando la promessa verrà rifiutata o in errore).

A screen shot of a computer program

AI-generated content may be incorrect.

* + 1. Utilizzo di `async/await`

Un altro approccio per la gestione delle operazioni asincrone è tramite `async` e `await`, è una sintassi più moderna e leggibile per lavorare con le promise. La parola chiave **async** rende una funzione asincrona, e all'interno di questa funzione puoi usare **await** per aspettare che una promise venga risolta. Questo approccio rende il codice asincrono simile a quello sincrono, migliorando la leggibilità e la manutenzione del codice.

A screen shot of a computer code

AI-generated content may be incorrect.

* 1. Riferimenti Utili

- [Documentazione ufficiale di Node.js](https://nodejs.org/en/docs)

- [Guida a Express.js](https://expressjs.com)

- [Documentazione di npm](https://docs.npmjs.com)