

Tecnologie Cloud e Mobile

Lez. 09 **Node.js**

Giuseppe Psaila

Università di Bergamo

giuseppe.psaila@unibg.it

Node.js: Perché

Linguaggi di programmazione nei sistemi web

- Client:
JavaScript
- Server (per rispondere alle richieste):
Java, PHP, C#
- Server (per usare la base dati):
SQL

Node.js: Perché

- Stabilito che SQL dipende dalla scelta fatta sulla base dati
- Due linguaggi di programmazione diversi tra il lato client e il lato server richiedono competenze diverse
- Per i programmatori JavaScript è preclusa la possibilità di lavorare sul lato server
- Ma in un mondo che sta evolvendo verso i Web Service, il lato client potrebbe non essere proprio sviluppato

Node.js: Perché

Obiettivo di Node.js

- Far spostare i programmatori JavaScript sullo sviluppo del lato server
- Usare la tecnologia JavaScript sul lato server, per manipolare i dati JSON con il linguaggio all'interno del quale sono nati

Node.js

- Node.js può essere facilmente scaricato dal sito ufficiale
- `https://nodejs.org/it/`
- Versione scaricata (da me): 12.16.2 (nel 2020)
- Ora è disponibile la 19.8.1 (2023)

Node.js (2020)

[HOME](#)[INFORMAZIONI](#)[DOWNLOAD](#)[DOCUMENTAZIONE](#)[COME PARTECIPARE](#)[SICUREZZA](#)[BLOG](#)

Node.js® è un runtime JavaScript costruito sul motore JavaScript V8 di Chrome.

Download per Windows (x64)

12.16.2 LTS

Consigliata

14.0.0 Corrente

Ultime funzionalità

[Altri download](#) | [Changelog](#) | [Documentazione API](#)

[Altri download](#) | [Changelog](#) | [Documentazione API](#)

Node.js (2023)



[HOME](#) | [INFORMAZIONI](#) | [DOWNLOADS](#) | [DOCUMENTAZIONE](#) | [COME PARTECIPARE](#) | [SICUREZZA](#) | [CERTIFICATION](#) | [BLOG](#)



Node.js® è un runtime JavaScript costruito sul motore JavaScript V8 di Chrome.

Download per Windows (x64)

18.15.0 LTS

Consigliata

19.8.1 Corrente

Ultime funzionalità

[Altri download](#) | [Changelog](#) | [Documentazione API](#)

[Altri download](#) | [Changelog](#) | [Documentazione API](#)

Attenzione

- La procedura di download chiederà se si vogliono scaricare delle utilità supplementari
- Scaricate e installate anche quelle
- Perché una è l'utilità «npm» che consente di installare pacchetti aggiuntivi a Node.js

Node.js

- Nella sezione «Documentation»
- Si trovano alcune guide e il riferimento alle API

Node.js (2020)

[HOME](#)[INFORMAZIONI](#)[DOWNLOAD](#)[DOCUMENTAZIONE](#)[COME PARTECIPARE](#)[SICUREZZA](#)[BLOG](#)

Documentazione

About Docs

[Edit on GitHub](#)[ES6 e oltre](#)[v12.16.2 API LTS](#)[v14.0.0 API](#)[Guide](#)[Dependencies](#)

There are several types of documentation available on this website:

- API reference documentation
- ES6 features
- Guides

API Reference Documentation

Node.js (2023)

[HOME](#)[ABOUT](#)[DOWNLOADS](#)[DOCS](#)[GET INVOLVED](#)[SECURITY](#)[CERTIFICATION](#)[NEWS](#)

Docs

[ES6 and beyond](#)[v18.15.0 API](#) LTS[v19.8.1 API](#)[Guides](#)[Dependencies](#)

About documentation

There are several types of documentation available on this website:

- [API reference documentation](#)
- [ES6 features](#)
- [Guides](#)

API reference documentation

The [API reference documentation](#) provides detailed information about a function or object in Node.js. This documentation indicates what arguments a method accepts, the return value of that method, and what errors may be related to that method. It also indicates which methods are available for different versions of Node.js.

Che cosa Vediamo

- Caratteristiche generali
- Avviare un server
- Esempio completo
- Debugging

Che Cosa Vedremo

Nella prossima lezione

- Come interagire con MongoDB
- Ricevendo dati da inserire come documenti JSON nel DB
- Pubblicando i documenti JSON memorizzati nel DB

Programmazione a Eventi

- I programmi Node.js sono fortemente basati sulla programmazione a eventi
- In JavaScript, sono le funzioni di callback
- Anche elaborazioni complesse andrebbero organizzate con chiamate indirette a funzioni di callback
- Generando, di fatto, eventi interni

Event Loop

- Node.js gestisce diverse code di eventi, a diversa priorità
- Il cosiddetto «Event Loop» ha il compito di
 1. Prendere un evento dalla coda non vuota a più alta priorità
 2. Chiamare la funzione di callback associata
 3. Ripetendo questo processo all'infinito

Richieste da una Porta TCP

- Una richiesta da una porta TCP viene trasformata in evento
- L'evento viene accodato ad una coda di eventi
- Quando è il turno, l'evento viene processato e la funzione di callback corrispondente viene chiamata
- In questo modo, il sistema può gestire richieste multiple su porte diverse

Oggetti Globali

- Un programma per Node.js non lavora nel browser
- Quindi non è associato ad una finestra
- L'oggetto Window non esiste
- Il nuovo oggetto di contesto si chiama **global**

global

- Un programma JavaScript per Node.js ha il compito di attivare un server
- Cioè mettersi in ascolto su una porta TCP e rispondere alle richieste entranti
- Inoltre, può gestire in modo nativo il protocollo HTTP (e HTTPS)

global

- Perciò, global fornisce oggetti utili a creare il server e a gestire il processo di comunicazione
- Vediamo una rapida lista di oggetti e metodi offerti da global

global: oggetti

- **process**

Consente di gestire il processo di elaborazione in corso

- **console**

Consente di scrivere messaggi (di debug) sulla console associata al processo

global: oggetti

- **__dirname**
La cartella da cui il processo è stato lanciato
- **__filename**
Il file con il codice JavaScript del processo

global: Classi (costruttori)

- **Buffer**

Gestisce gli array, in particolare viene usato per ricevere gli argomenti sulla linea di comando

- **URL**

Consente di gestire gli URL

- **URLSearchParams**

Gestisce i parametri di ricerca dell'URL (la query string del metodo GET)

global: Classi (costruttore)

- **TextDecoder**

Effettua il decoding dei caratteri di un testo (es. da UTF-8 a rappresentazione interna)

- **TextEncoder**

Effettua l'encoding di un testo (es. da formato interno a UTF-8)

global: Metodi

- `require()`
Indica i moduli da importare
- `setTimeout()`
Imposta un time out per chiamare una funzione di callback

global: Metodi

- **`setInterval()`**
Schedula la chiamata ripetuta di una funzione di callback ad intervalli regolari
- **`setImmediate()`**
Inserisce la funzione di callback specificata nella coda degli eventi per essere chiamata non appena possibile

global: Metodi

- **clearTimeout()**
Cancella un time out precedentemente impostato
- **clearInterval()**
Cancella un Interval precedentemente schedulato

global: Metodi

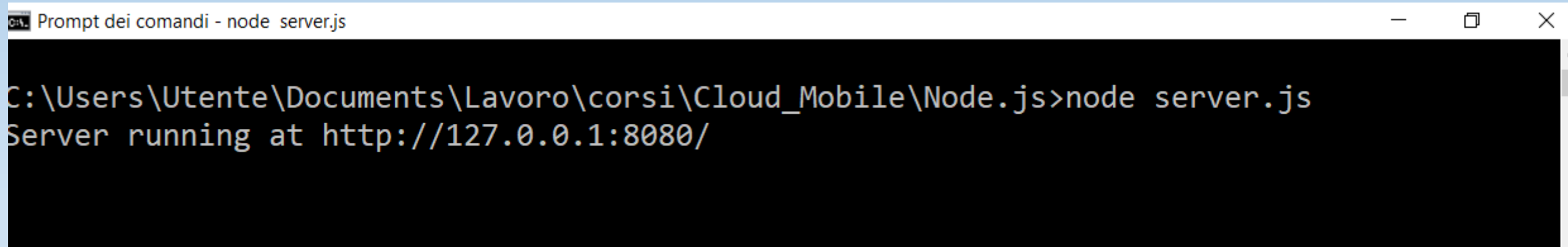
- **`clearImmediate()`**

Rimuove dalla coda degli eventi una richiesta di chiamare una funzione di callback precedentemente schedulata

Esecuzione

- Dato un programma, per esempio «server.js»
- Con il comando «node» si avvia il programma
node server.js

Esecuzione



```
Prompt dei comandi - node server.js

C:\Users\Utente\Documents\Lavoro\corsi\Cloud_Mobile\Node.js>node server.js
Server running at http://127.0.0.1:8080/
```

- Essendo un server, il processo è attivo e aspetta di servire qualche richiesta
- Per interrompere: Ctrl + C

Esempio di Codice: Parte 1

```
const http = require('http')  
const url = require('url')  
  
const hostname = '127.0.0.1'  
const port = '8080'
```

Esempio di Codice: Parte 1

- Nella parte iniziale, occorre specificare i moduli che verranno utilizzati nel seguito
 - http: modulo predefinito per creare server http
 - url: modulo predefinito per manipolare gli URL
- Vengono definite due costanti:
 - Indirizzo IP dell'host
 - Porta TCP

Esempio di Codice: Parte 2

```
const server =  
  http.createServer(function (req, res) {  
...  
  });  
server.listen(port, hostname, function ()  
{  
  console.log('Server running at ' +  
    'http://${hostname}:${port}/')  
})
```

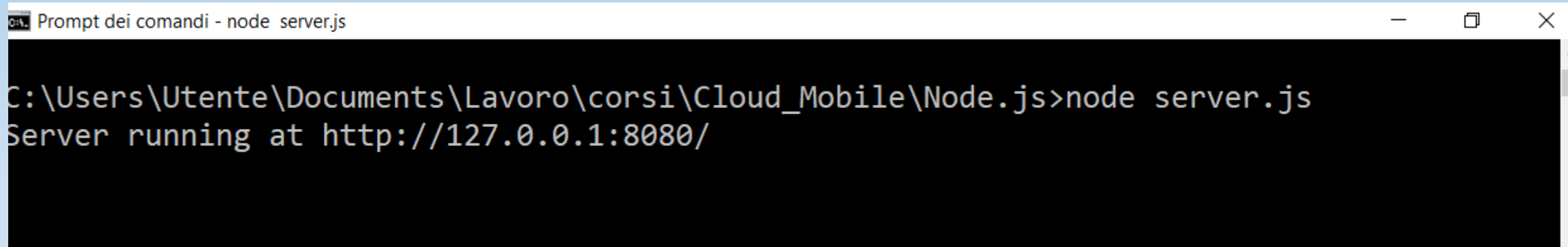

Esempio di Codice: Parte 2

- Viene creato il server, con `http.createServer`
- Il parametro è la funzione di callback da chiamare quando arriva una richiesta
- Il server viene assegnato alla variabile/costante `server`

Esempio di Codice: Parte 2

- Occorre associare il server all'host e alla porta
- `server.listen(port, hostname, function () { ... }) ;`
- Il metodo mette in ascolto il server sull'host e sulla porta specificata
- La funzione di callback viene chiamata quando il server è effettivamente in ascolto

Esempio di Codice: Parte 2



```
Prompt dei comandi - node server.js

C:\Users\Utente\Documents\Lavoro\corsi\Cloud_Mobile\Node.js>node server.js
Server running at http://127.0.0.1:8080/
```

- Il messaggio «Server Running at ...» è quindi generato dalla funzione di callback fornita al metodo «listen» di «server»

Callback del Server: Parte 1

```
function (req, res) {  
    console.log("Request Received\n");  
    ...  
}
```

Callback del Server: Parte 1

- Parametri:
 - **req**: oggetto che descrive la richiesta HTTP
Classe: `http.ClientRequest`
 - **res**: oggetto che gestisce la risposta
Classe: `http.ServerResponse`

Nel corpo

- `Console.log`
Manda in output sulla console un messaggio

Callback del Server: Parte 2

...

```
var myurl = new url.URL(  
    "http://" + req.headers.host +  
    req.url);  
var params = myurl.searchParams;  
var param_a = "";  
if( params.has("a") )  
    param_a = params.get("a");
```

...

Callback del Server: Parte 2

- Vogliamo ottenere il parametro «a» dall'URL della richiesta
- `req.url` fornisce l'URL senza dominio e protocollo, cioè dal carattere «/» in poi
- L'oggetto URL è capace di gestire i parametri, ma vuole un URL completo nel costruttore
- Per ottenere il dominio
`req.headers.host`

Callback del Server: Parte 2

- Il nuovo oggetto URL viene assegnato alla variabile `myurl`
- Il campo `myurl.searchParams` è un oggetto che gestisce i parametri della ricerca
- L'oggetto con i parametri viene assegnato alla variabile «params»

Callback del Server: Parte 2

- Il metodo `params.has`
verifica la presenza del parametro richiesto
- Il metodo `params.get`
restituisce una stringa con il valore del parametro richiesto
- Il valore viene assegnato alla variabile «param_a»

Callback del Server: Parte 3

...

```
var o = new Object();  
  o.param_a = param_a;  
var l = new Object();  
for (const [name, value] of params)  
  { l[name] = value; }  
o.list = l;
```

...

Callback del Server: Parte 3

- Prepariamo l'output: un documento JSON che contiene
 - un campo «param_a» (con il valore del parametro)
 - un campo «list», un documento annidato con tutti i parametri trovati nella richiesta
- La variabile «o» contiene l'oggetto da serializzare
- La variabile «l» contiene la lista completa dei parametri

Callback del Server: Parte 3

- Il ciclo for scandisce l'oggetto params
- Usa il concetto di «iteratore», cioè scandisce automaticamente la lista
- La scrittura [name, value] definisce l'iteratore
- Nel nostro caso, una coppia di variabili «name» e «value»
- Che diventano il contesto dell'azione del for

Callback del Server: Parte 4

...

```
res.statusCode = 200
res.setHeader('Content-Type',
              'application/json')
res.end(JSON.stringify(o));
});
```

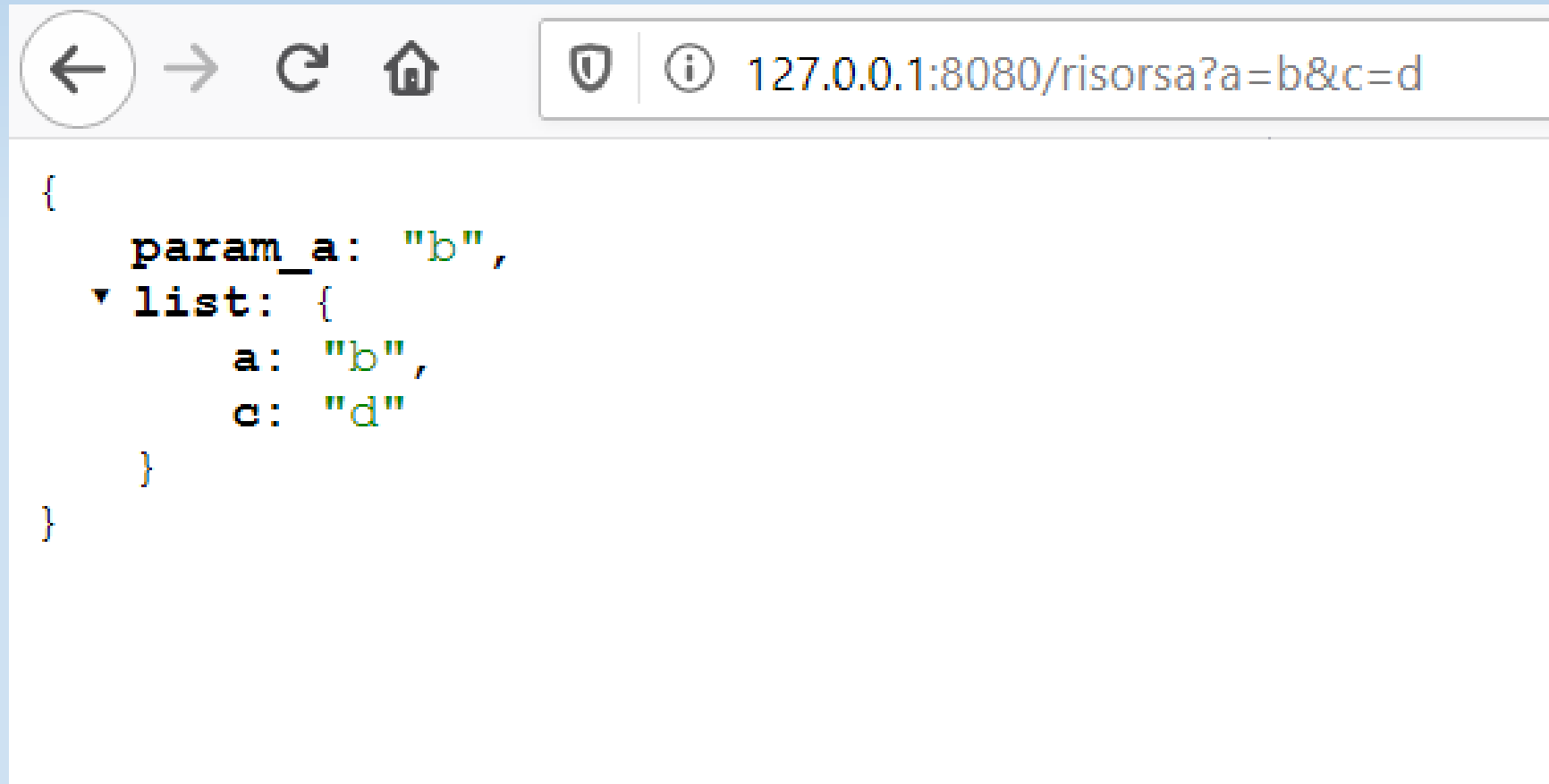
Callback del Server: Parte 4

- Si predispone l'header della risposta
 - Codice HTTP: 200 (tutto OK)
 - MIME Type del contenuto
`application/json`
- Invio della risposta:
- Serializzazione dell'oggetto
`JSON.stringify(o)`
- Chiusura e invio:
`res.end`

Invocazione

- Usiamo il seguente URL:
`http://127.0.0.1:8080/?a=b&c=d`
- Che invia due parametri
 - a=b
 - c=d

Output



A screenshot of a web browser window displaying a REST client response. The address bar shows the URL `127.0.0.1:8080/risorsa?a=b&c=d`. The response body contains a JSON object with the following structure:

```
{
  param_a: "b",
  list: {
    a: "b",
    c: "d"
  }
}
```

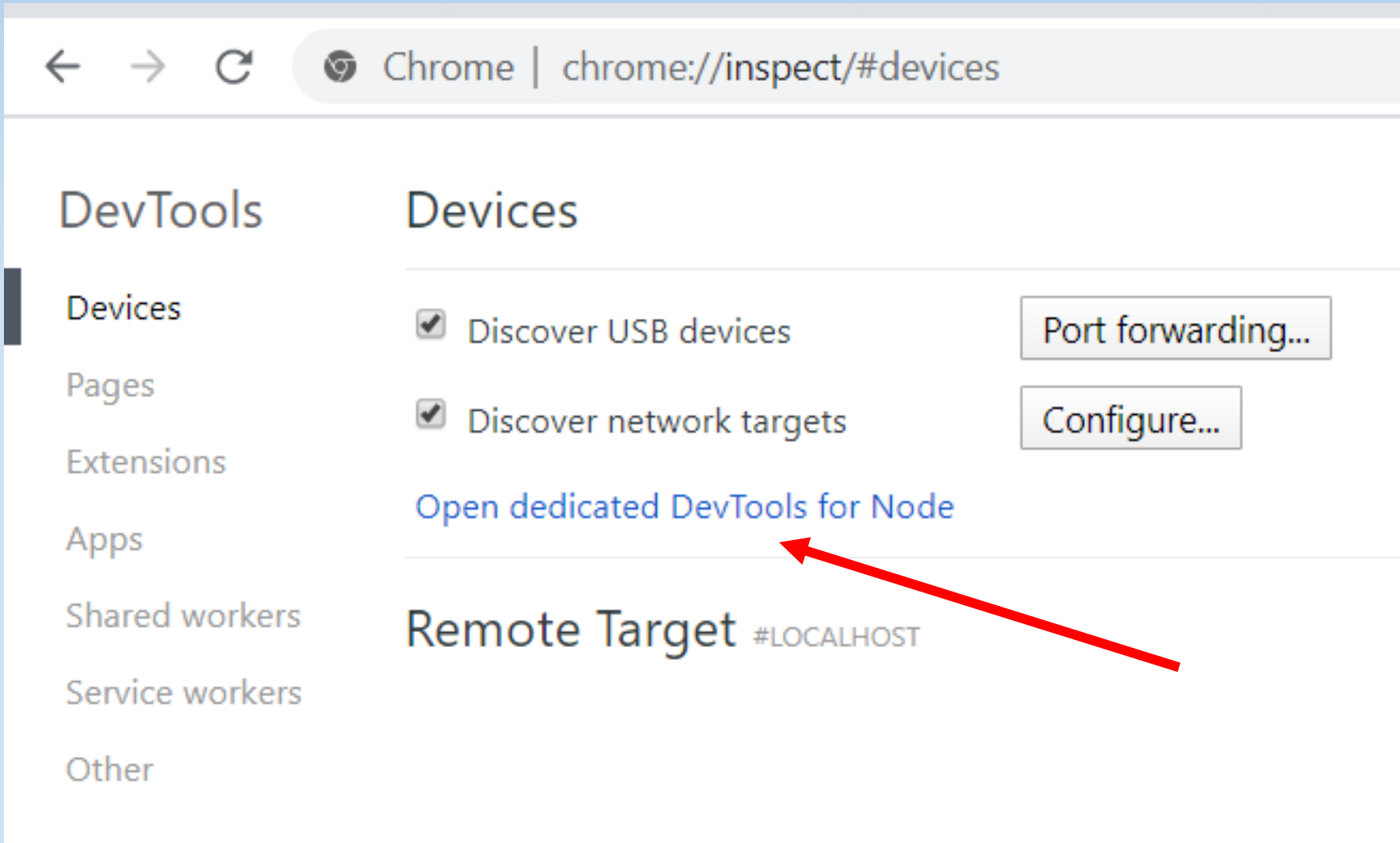

Debugging

- Si può fare debugging del codice JavaScript
- Attivando Node.js in modalità di debug
- Viene aperta una porta TCP (9229) specifica che dei tool esterni possono contattare
- Avviare con:
`node --inspect server.js`

Debugger

- Aprire Google Chrome
- Nella barra degli indirizzi, scrivere `chrome://inspect`
- Viene avviato lo strumento di debugging di Chrome

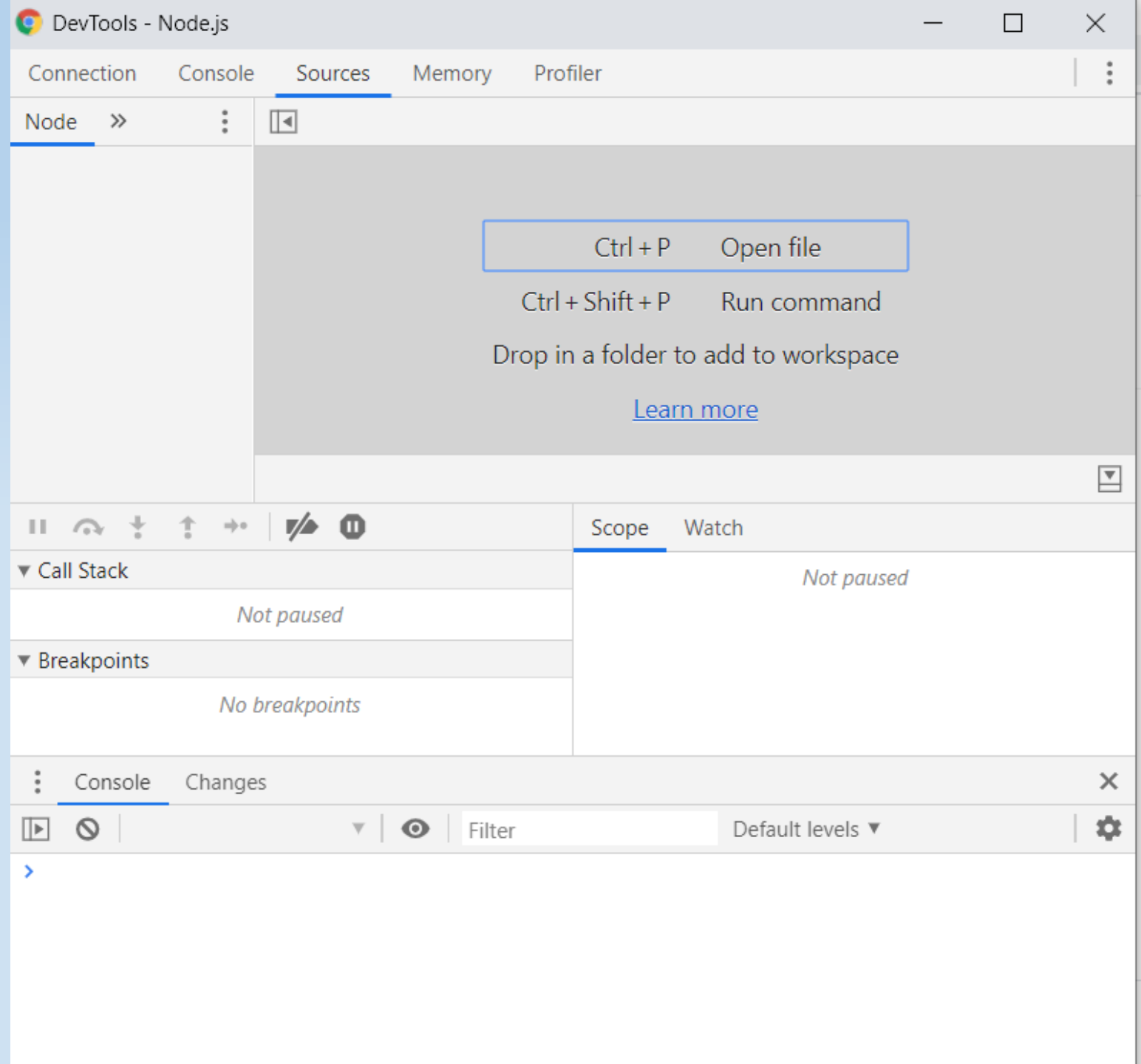
Debugger



Debugger

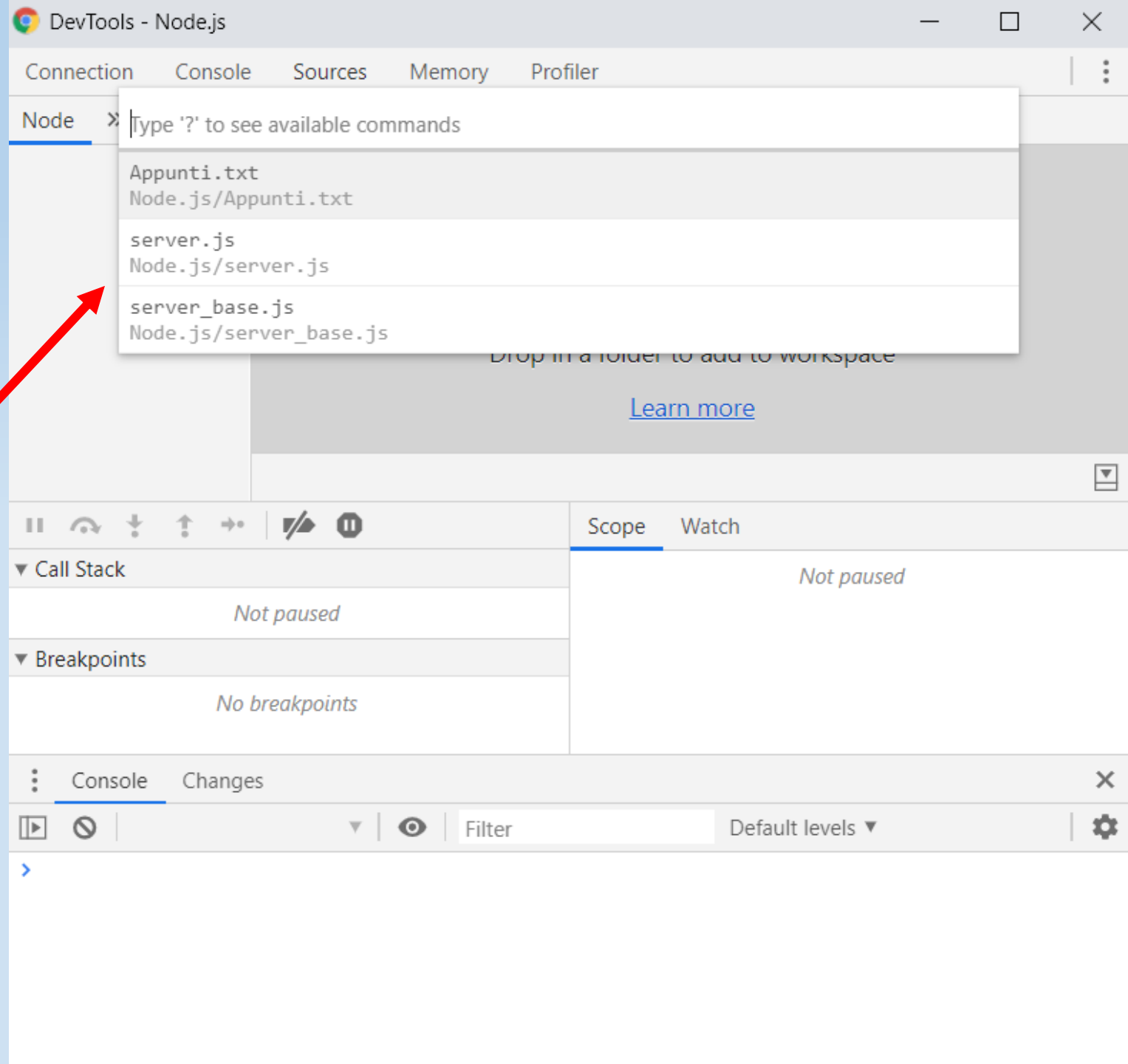
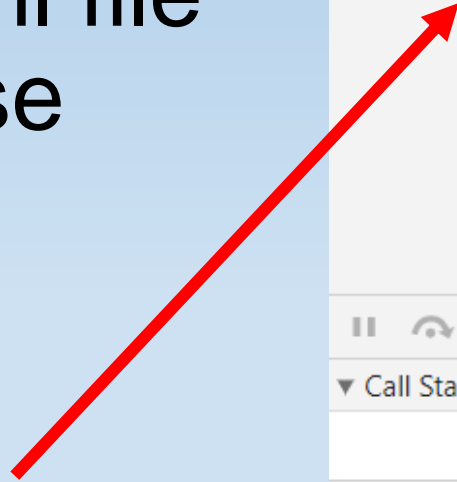
- Cliccando su «Open Dedicated Debugger for Node»
- Si apre la finestra del debugger
- Per aprire il file sorgente, premere Ctrl + P

Debugger



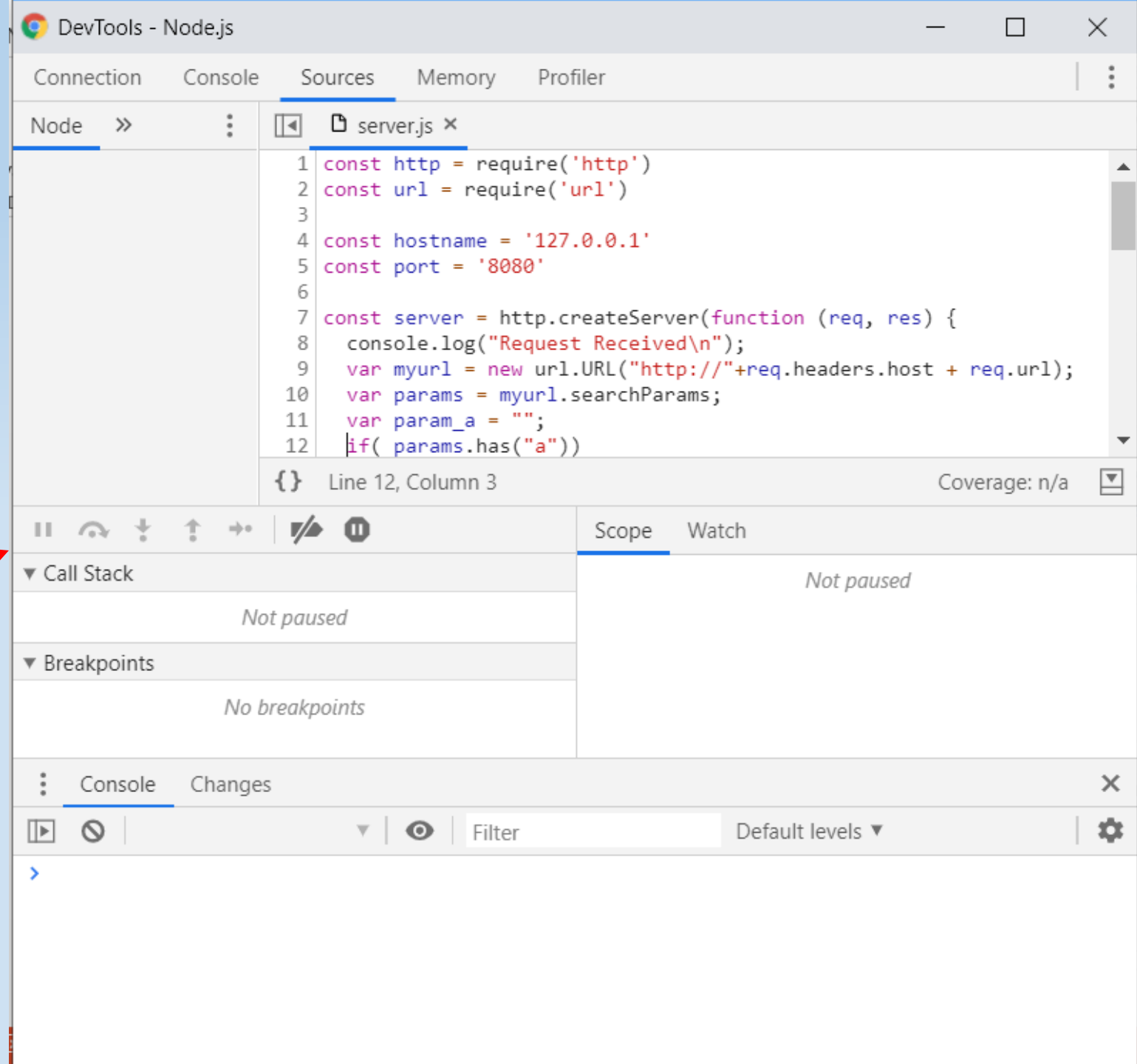
Debugger

- Si sceglie il file di interesse



Debugger

- Il codice è caricato:
le funzionalità del debugger sono attive



Debugger

- Breakpoint attivato

The screenshot displays the Chrome DevTools interface with the 'Sources' panel active. A breakpoint is set at line 15 of the file `server.js`. The code in the editor is as follows:

```
7 const server = http.createServer(function (req, res) { req = IncomingMessage;
8   console.log("Request Received\n");
9   var myurl = new url.URL("http://" + req.headers.host + req.url);
10  var params = myurl.searchParams; params = URLSearchParams {Symbol(Symbol.iterator): ...
11  var param_a = ""; param_a = "b"
12  if( params.has("a")) params = URLSearchParams {Symbol(query): Ar
13    param_a = params.get("a"); param_a = "b"
14
15  var o = new Object();
16  o.param_a = param_a;
17  var l = new Object();
18
```

The status bar at the bottom of the editor indicates 'Line 15, Column 11' and 'Coverage: n/a'.

Below the editor, the 'Paused on breakpoint' message is shown. The 'Call Stack' panel lists the following frames:

- (anonymous) server.js:15
- emit events.js:311
- parserOnIncoming _http_server.js:784
- parserOnHeaderComplete

The 'Scope' panel shows the following local variables:

- `l`: undefined
- `myurl`: URL {Symbol(context): URLContext, ...}
- `o`: undefined
- `param_a`: "b"
- `params`: URLSearchParams {Symbol(query): ...}
- `req`: IncomingMessage {readableState: Re...

The 'Console' panel at the bottom shows the following log messages:

- Server running at <http://127.0.0.1:8080/> server.js:30
- Request Received server.js:8

Problemi con la Porta

- Se la porta 9229 risulta occupata (come è successo a me)
- Si può cambiare la porta con l'opzione `--inspect-port=9228`
- Il comando diventa:
`node --inspect-port=9228 --inspect server.js`

Sul Prompt

```
C:\Users\Utente\Documents\Lavoro\corsi\Cloud_Mobile\Node.js>node --inspect-port=9228
--inspect server.js
Debugger listening on ws://127.0.0.1:9228/91bdfbde-50e5-408f-b05c-5e59e6b750a6
For help, see: https://nodejs.org/en/docs/inspector
Server running at http://127.0.0.1:8080/
Debugger attached.
```

Considerazioni

- Abbiamo visto una rapida introduzione a Node.js
- È uno strumento molto interessante
- Ma così come è, richiede uno sforzo di programmazione significativo se dobbiamo generare anche il codice HTML
- Se invece dobbiamo ricevere richieste e generare JSON (per esempio, AJAX), va benissimo

Application Servers

- Tuttavia, esistono moltissimi moduli e complementamenti per Node.js
- Per esempio, driver per connettersi con i database
- **Application Servers:** consentono di gestire in modo ingegneristico (per esempio, con il paradigma del Model-View-Controller, MVC) lo sviluppo e la manutenzione delle applicazioni web

Model-View-Controller

- È un paradigma architetturale molto diffuso
- Suddivide l'applicazione in tre parti:
 - Model: il gestore del modello dei dati, normalmente il DB
 - View: l'interfaccia utente,
 - Controller: il codice che controlla le trasformazioni dei dati, in base alle richieste ricevute dal view

Model-View-Controller

- In un applicativo web «classico», non basato su AJAX
 - Model: il DB
 - View: il codice HTML che viene inviato dal server, il cui contenuto viene generato a partire dai dati ottenuti dal DB
 - Controller: il codice lato server che esegue le query sul DB e comanda la rigenerazione della pagina (gestisce la «business logic» dell'applicazione)
- Un framework molto usato: Spring

Spring

- Sito: spring.io
- Spring ha circa 20 anni (forse qualcosa meno)
- Negli anni si è evoluto verso approcci moderni, tra cui anche i microservizi (di cui parleremo più avanti)
- Quindi, è una buona soluzione per realizzare web app usando Java sul lato server

Model-View-Controller

- In un applicativo web basata su AJAX
 - Model: il DB
 - View: il codice JavaScript che invia richieste al server e riceve i dati da visualizzare
 - Controller: il codice lato server che riceve le richieste AJAX, esegue le query sul DB e gestisce la business logic, invia i dati al client
- Un framework molto usato: Angular.js

Angular.js

- Sito: angularjs.org
- Supporta il lato view del paradigma MVC
- Con poco sforzo è possibile progettare l'interfaccia e legarla al lato server basato su AJAX

Angular

- Sito: angular.io
- Evoluzione di Angular.js
- Riscritto completamente con TypeScript
- Sempre compatibile con i browser moderni
- Che cosa è TypeScript?
- L'evoluzione di JavaScript proposta da Microsoft
- Un programma TypeScript viene compilato in JavaScript