1) What is node.js?

Node.js is a Server side scripting which is used to build scalable programs. Its multiple advantages over other server side languages, the prominent being non-blocking I/O.

2) How node.js works?

Node.js works on a v8 environment, it is a virtual machine that utilizes JavaScript as its scripting language and achieves high output via non-blocking I/O and single threaded event loop.

3) What do you mean by the term I/O ?

I/O is the shorthand for input and output, and it will access anything outside of your application. It will be loaded into the machine memory to run the program, once the application is started.

4) What does event-driven programming mean?

In computer programming, event driven programming is a programming paradigm in which the flow of the program is determined by events like messages from other programs or threads. It is an application architecture technique divided into two sections 1) Event Selection 2) Event Handling.

5) Where can we use node.js?

Node.js can be used for the following purposes.

- Web applications (especially real-time web apps)
- Network applications
- Distributed systems
- General purpose applications

6) What is the advantage of using node.js?

- It provides an easy way to build scalable network programs
- Generally fast
- Great concurrency
- Asynchronous everything
- Almost never blocks

7) What are the two types of API functions in Node.js ?

The two types of API functions in Node.js are

- Asynchronous, non-blocking functions
- Synchronous, blocking functions

8) What is control flow function?

A generic piece of code which runs in between several asynchronous function calls is known as control flow function.

9) Explain the steps how “Control Flow” controls the functions calls?

- Control the order of execution
- Collect data
- Limit concurrency

- Call the next step in program

10) Why Node.js is single threaded?

For async processing, Node.js was created explicitly as an experiment. It is believed that more performance and scalability can be achieved by doing async processing on a single thread under typical web loads than the typical
thread based implementation.

11) Does node run on windows?

Yes – it does. Download the MSI installer from https://nodejs.org/download/

12) Can you access DOM in node?

No, you cannot access DOM in node.

13) Using the event loop what are the tasks that should be done asynchronously?

- I/O operations
- Heavy computation
- Anything requiring blocking

14) Why node.js is quickly gaining attention from JAVA programmers?

Node.js is quickly gaining attention as it is a loop based server for JavaScript.
Node.js gives user the ability to write the JavaScript on the server, which has
access to things like HTTP stack, file I/O, TCP and databases.

15) What are the two arguments that async.queue takes?

The two arguments that async.queue takes

- Task function
- Concurrency value

16) What is an event loop in Node.js ?

To process and handle external events and to convert them into callback invocations an event loop is used. So, at I/O calls, node.js can switch from one request to another.

17) Mention the steps by which you can async in Node.js?

By following steps you can async Node.js

- First class functions
- Function composition
- Callback Counters
- Event loops

18) What are the pros and cons of Node.js?

Pros:

- If your application does not have any CPU intensive computation, you can build it in Javascript top to bottom, even down to the database level if you use JSON storage object DB like MongoDB.
- Crawlers receive a full-rendered HTML response, which is far more SEO friendly rather than a single page application or a websockets app

run on top of Node.js.

Cons:

- Any intensive CPU computation will block node.js responsiveness, so a threaded platform is a better approach.
- Using relational database with Node.js is considered less favourable.

19) How Node.js overcomes the problem of blocking of I/O operations?

Node.js solves this problem by putting the event based model at its core, using an event loop instead of threads.

20) What is the difference between Node.js vs Ajax?

The difference between Node.js and Ajax is that, Ajax (short for Asynchronous
Javascript and XML) is a client side technology, often used for updating the contents of the page without refreshing it. While,Node.js is Server Side Javascript, used for developing server software. Node.js does not execute in
the browser but by the server.

21) What are the Challenges with Node.js ?

Emphasizing on the technical side, it‟s a bit of challenge in Node.js to have one process with one thread to scale up on multi core server.

22) What does it mean “non-blocking” in node.js?

In node.js “non-blocking” means that its IO is non-blocking. Node uses “libuv”
to handle its IO in a platform-agnostic way. On windows, it uses completion ports for unix it uses epoll or kqueue etc. So, it makes a non-blocking request
and upon a request, it queues it within the event loop which call the JavaScript
„callback‟ on the main JavaScript thread.

23) What is the command that is used in node.js to import external libraries?

Command “require” is used for importing external libraries, for example, “var
http=require (“http”)”. This will load the http library and the single exported object through the http variable.

24) Mention the framework most commonly used in node.js?

“Express” is the most common framework used in node.js.

25) What is „Callback‟ in node.js?

Callback function is used in node.js to deal with multiple requests made to the
server. Like if you have a large file which is going to take a long time for a server to read and if you don‟t want a server to get engage in reading that large file while dealing with other requests, call back function is used. Call back function allows the server to deal with pending request first and call a

function when it is finished.

48+ Top Node.js Interview Questions and Answers in 2021

Lesson 7 of 7By Taha Sufiyan
Last updated on Apr 9, 2021129830
Previous

Node.js is a super popular server-side platform that more and more organizations are
using. If you are preparing for a career change and have an upcoming job interview, it"s
always a good idea to prepare and brush up on your interview skills beforehand. Although there are a few commonly asked Node.js interview questions that pop up during all types of interviews, we also recommend that you prepare by focusing on exclusive questions to your specific industry.

We have compiled a comprehensive list of common Node.js interview questions that come up often in interviews and the best ways to answer these questions. This will also
help you understand the fundamental concepts of Node.js.

The Node.js interview questions are grouped into the following categories:

1. Beginner Node.js Interview Questions
2. Intermediate Node.js Interview Questions
3. Advanced Node.js Interview Questions

EXPLORE COURSE

Beginner Node.js Interview Questions

1. What is Node.js? Where can you use it?

Node.js is an open-source, cross-platform JavaScript runtime environment and library to
run web applications outside the client"s browser. It is used to create server-side web
applications.

Node.js is perfect for data-intensive applications as it uses an asynchronous, eventdriven
model. You can use I/O intensive web applications like video streaming sites.

You can also use it for developing: Real-time web applications, Network applications, General-purpose applications, and Distributed systems.

2. Why use Node.js?

Node.js makes building scalable network programs easy. Some of its advantages include:

- It is generally fast
- It rarely blocks
- It offers a unified programming language and data type
- Everything is asynchronous
- It yields great concurrency

3. How does Node.js work?

A web server using Node.js typically has a workflow that is quite similar to the diagram

illustrated below. Let"s explore this flow of operations in detail.

- Clients send requests to the webserver to interact with the web application.

Requests
can be non-blocking or blocking:

- Querying for data
- Deleting data
- Updating the data
- Node.js retrieves the incoming requests and adds those to the Event Queue
- The requests are then passed one-by-one through the Event Loop. It checks if the requests are simple enough not to require any external resources
- The Event Loop processes simple requests (non-blocking operations), such as I/O Polling, and returns the responses to the corresponding clients

A single thread from the Thread Pool is assigned to a single complex request. This thread is responsible for completing a particular blocking request by accessing external
resources, such as computation, database, file system, etc.
Once the task is carried out completely, the response is sent to the Event Loop that sends that response back to the client

Why is Node.js Single-threaded?

Node.js is single-threaded for async processing. By doing async processing on a singlethread
under typical web loads, more performance and scalability can be achieved instead of the typical thread-based implementation.

5. Explain callback in Node.js.

A callback function is called after a given task. It allows other code to be run in the meantime and prevents any blocking. Being an asynchronous platform, Node.js heavily
relies on callback. All APIs of Node are written to support callbacks.

6. How would you define the term I/O?

- The term I/O is used to describe any program, operation, or device that transfers data to or from a medium and to or from another medium
- Every transfer is an output from one medium and an input into another. The medium

can be a physical device, network, or files within a system

7. How is Node.js most frequently used?

Node.js is widely used in the following applications:

1. Real-time chats
2. Internet of Things
3. Complex SPAs (Single-Page Applications)
4. Real-time collaboration tools
5. Streaming applications
6. Microservices architecture

8. Explain the difference between frontend and backend development?

Front-end Back-end
Frontend refers to the client-side of an application
Backend refers to the serverside
of an application
It is the part of a web application that users can see and
interact with

It constitutes everything that
happens behind the scenes
It typically includes everything that attributes to the
visual aspects of a web application
It generally includes a web
server that communicates with a
database to serve requests
HTML, CSS, JavaScript, AngularJS, and ReactJS are
some of the essentials of frontend development
Java, PHP, Python, and Node.js
are some of the backend
development technologies

9. What is NPM?

NPM stands for Node Package Manager, responsible for managing all the packages and modules for Node.js.
Node Package Manager provides two main functionalities:

- Provides online repositories for node.js packages/modules, which are searchable on
search.nodejs.org
- Provides command-line utility to install Node.js packages and also manages Node.js
versions and dependencies

10. What are the modules in Node.js?

Modules are like JavaScript libraries that can be used in a Node.js application to include
a set of functions. To include a module in a Node.js application, use
the require() function with the parentheses containing the module's name.
Node.js has many modules to provide the basic functionality needed for a web application. Some of them include:

Core Modules	Description
HTTP	Includes classes, methods, and events to create a Node.js HTTP server
util	Includes utility functions useful for developers
fs(file system)	Includes events, classes, and methods to deal with file I/O operations
url	Includes methods for URL parsing query string

Includes methods to work with

11. Why is Node.js preferred over other backend technologies like Java and PHP?

Some of the reasons why Node.js is preferred include:

- Node.js is very fast
- Node Package Manager has over 50,000 bundles available at the developer‟s disposal
- Perfect for data-intensive, real-time web applications, as Node.js never waits for an API to return data
- Better synchronization of code between server and client due to same code base
- Easy for web developers to start using Node.js in their projects as it is a JavaScript Library

What is the package.json file?

The package.json file is the heart of a Node.js system. This file holds the metadata for a particular project. The package.json file is found in the root directory of any Node application or module

What is REPL in Node.js?

REPL stands for Read Eval Print Loop, and it represents a computer environment. It‟s similar to a Windows console or Unix/Linux shell in which a command is entered. Then, the system responds with an output

React JS Questions

1. What is React?

React is a front-end and open-source JavaScript library which is useful in developing user interfaces specifically for applications with a single page. It is helpful in building complex and reusable user interface(UI) components of mobile and web applications as it follows the component-based approach.

The important features of React are:

- It supports server-side rendering.
- It will make use of the virtual DOM rather than real DOM (Data Object Model) as RealDOM manipulations are expensive.
- It follows unidirectional data binding or data flow.
- It uses reusable or composable UI components for developing the view.

2. What is useState() in React?

The useState() is a built-in React Hook that allows you for having state variables in functional components. It should be used when the DOM has something that is dynamically manipulating/controlling

What are keys in React?

A key is a special string attribute that needs to be included when using lists of elements.

Example of a list using key -

```
const ids = [1,2,3,4,5];
const listElements = ids.map((id)=>{
return(
<li key={id.toString()}>
  {id}
</li>
)
})
```

Importance of keys -

- Keys help react identify which elements were added, changed or removed.
- Keys should be given to array elements for providing a unique identity for each element.
- Without keys, React does not understand the order or uniqueness of each element.
- With keys, React has an idea of which particular element was deleted, edited, and added.
- Keys are generally used for displaying a list of data coming from an API.

. What is JSX?

JSX stands for JavaScript XML. It allows us to write HTML inside JavaScript and place them in the DOM without using functions like appendChild( ) or createElement( ).

Note- We can create react applications without using JSX as well.

Let's understand **how JSX works**:

Without using JSX, we would have to create an element by the following process:

```
const text = React.createElement('p', {}, 'This is a text');
const container = React.createElement('div','{}',text );
ReactDOM.render(container,rootElement);
```

Using JSX, the above code can be simplified:

```
const container = (
<div>
  <p>This is a text</p>
</div>
);
ReactDOM.render(container,rootElement);
```

As one can see in the code above, we are directly using HTML inside JavaScript.

7. What are the differences between functional and class components?

Before the introduction of Hooks in React, functional components were called stateless components and were behind class components on a feature basis. After the introduction of Hooks, functional components are equivalent to class components.

Although functional components are the new trend, the react team insists on keeping class components in React. Therefore, it is important to know how these components differ.

Functional components are nothing but JavaScript functions and therefore can be declared using an arrow function or the function keyword:

```
  function card(props){
   return(
      <div className="main-container">
        <h2>Title of the card</h2>
      </div>
    )
   }
   const card = (props) =>{
    return(
      <div className="main-container">
        <h2>Title of the card</h2>
      </div>
    )
   }
```

Class components, on the other hand, are declared using the ES6 class:

```
class Card extends React.Component{
 constructor(props){
    super(props);
  }
   render(){
     return(
       <div className="main-container">
         <h2>Title of the card</h2>
       </div>
     )
   }
  }
```

- **Handling props**

Let's render the following component with props and analyse how functional and class components handle props:

```
<Student Info name="Vivek" rollNumber="23" />
```

In functional components, the handling of props is pretty straightforward. Any prop provided as an argument to a functional component can be directly used inside HTML elements:

```
function StudentInfo(props){
  return(
    <div className="main">
      <h2>{props.name}</h2>
      <h4>{props.rollNumber}</h4>
    </div>
  )
}
```

In the case of class components, props are handled in a different way:

```
class StudentInfo extends React.Component{
  constructor(props){
    super(props);
   }
   render(){
     return(
       <div className="main">
         <h2>{this.props.name}</h2>
         <h4>{this.props.rollNumber}</h4>
       </div>
     )
   }
  }
```

As we can see in the code above, **this** keyword is used in the case of class components.

- **Handling state**

Functional components use React hooks to handle state. It uses the useState hook to set the state of a variable inside the component:

```
function ClassRoom(props){
  let [studentsCount,setStudentsCount] = useState(0);
   const addStudent = () => {
     setStudentsCount(++studentsCount);
  }
   return(
     <div>
       <p>Number of students in class room: {studentsCount}</p>
       <button onClick={addStudent}>Add Student</button>
     </div>
   )
  }
```

Since useState hook returns an array of two items, the first item contains the current state, and the second item is a function used to update the state.

In the code above, using array destructuring we have set the variable name to studentsCount with a current value of "0" and setStudentsCount is the function that is used to update the state.

For reading the state, we can see from the code above, the variable name can be directly used to read the current state of the variable.

We cannot use React Hooks inside class components, therefore state handling is done very differently in a class component:

Let's take the same above example and convert it into a class component:

```
class ClassRoom extends React.Component{
        constructor(props){
            super(props);
            this.state = {studentsCount : 0};

            this.addStudent = this.addStudent.bind(this);
         }

            addStudent(){
            this.setState((prevState)=>{
               return {studentsCount: prevState.studentsCount++}
            });
         }

            render(){
             return(
               <div>
                 <p>Number of students in class room:
{this.state.studentsCount}</p>
                 <button onClick={this.addStudent}>Add Student</button>
               </div>
             )
           }
         }
```

In the code above, we see we are using **this.state** to add the variable studentsCount and setting the value to "0".

For reading the state, we are using **this.state.studentsCount**.

For updating the state, we need to first bind the addStudent function to **this**. Only then, we will be able to use the **setState** function which is used to update the state.

.

What is the use of useEffect React Hooks?

The useEffect React Hook is used for performing the side effects in functional components. With the help of useEffect, you will inform React that your component requires something to be done after rendering the component or after a state change. The function you have passed(can be referred to as "effect") will be remembered by React and call afterwards the performance of DOM updates is over. Using this, we can perform various calculations such as data fetching, setting up document title, manipulating DOM directly, etc, that don't target the output value. The useEffect hook will run by default after the first render and also after each update of the component. React will guarantee that the DOM will be updated by the time when the effect has run by it.

The useEffect React Hook will accept 2 arguments: `useEffect(callback,[dependencies]);`

Where the first argument callback represents the function having the logic of side-effect and it will be immediately executed after changes were being pushed to DOM. The second argument dependencies represent an optional array of dependencies. The useEffect() will execute the callback only if there is a change in dependencies in between renderings.

Example:

```
import { useEffect } from 'react';
function WelcomeGreetings({ name }) {
 const msg = `Hi, ${name}!`;     // Calculates output
 useEffect(() => {
   document.title = `Welcome to you ${name}`;    // Side-effect!
 }, [name]);
 return <div>{msg}</div>;          // Calculates output
}
```

The above code will update the document title which is considered to be a side-effect as it will not calculate the component output directly. That is why updating of document title has been placed in a callback and provided to useEffect().

Consider you don't want to execute document title update each time on rendering of WelcomeGreetings component and you want it to be executed only when the name prop changes then you need to supply name as a dependency to `useEffect(callback, [name])`.

. Explain React Hooks.

What are Hooks? Hooks are functions that let us “hook into” React state and lifecycle features from a **functional component.**

React Hooks **cannot** be used in class components. They let us write components without class.

Why were Hooks introduced in React?

React hooks were introduced in the 16.8 version of React. Previously, functional components were called stateless components. Only class components were used for state management and lifecycle methods. The need to change a functional component to a class component, whenever state management or lifecycle methods were to be used, led to the development of Hooks.

Example of a hook: **useState hook:**

In functional components, the useState hook lets us define a state for a component:

```
function Person(props) {
// We are declaring a state variable called name.
// setName is a function to update/change the value of name
let [name, setName] = useState('');
}
```

The state variable “name” can be directly used inside the HTML.

17. What are the rules that must be followed while using React Hooks?

There are 2 rules which must be followed while you code with Hooks:

- React Hooks must be called only at the top level. It is not allowed to call them inside the nested functions, loops, or conditions.
- It is allowed to call the Hooks only from the React Function Components.