

Node JS

Questionnaire

- [Node JS Questionnaire](#)

Node JS Questionnaire

1. Since the node is a single-threaded process, how to make use of all CPUs?

The node js is single-threaded but the node name itself suggests that it is a topology, which means there is communication between two nodes. This communication between the two nodes makes use of all CPUs.

2. Syntax to install modules using NPM.

Syntax : `$ npm install`

Example:

`$ npm install express`

`var express = ('express');`

Where express is the Name of the module,

express is a JS file that can be used in your module.

3. Mention types of npm modules available that are used very often. # S

Express, connect, socket.io and SockeJS, pug, MongoDB and Mongojs, Redis, Lodash, forever, bluebird, moment are some of the npm modules.

4. What are the attributes of package.json

Name, version, description, homepage, author, contributors, dependencies ? list of dependencies, repository ? repository type and URL of the package, the main ? entry point of the package, keywords.

5. What kind of applications can be built by Node.JS.

DIRT, JSON, I/O Bound, Single Page applications are preferable to use Node.JS.

6. NodeJS is a framework. True or False.

False. It is a runtime or environment, but not a framework.

7. What is the cluster?

Cluster is a process to handle thread execution load while working with multi-core systems.

8. How does the URL module work?

This module helps URL to parse it into host, pathname, search, query, etc.

Example:

```
var url = require('url');  
  
var adr = 'https://mindmajix.com/?s=node+js+training';  
  
var q = url.parse(adr, true);  
  
console.log(q.host); //returns 'mindmajix.com'  
  
console.log(q.search); //returns '?s=node+js+training'
```

9. What are the types of versions available? #S

Patch_Version, Minor_Version, Major_Version.

10. What are event emitters in NodeJS? #S

Objects in NodeJS will have to trigger events in order to maintain the asynchronous execution of the core API used. These objects that emit events are known as Event Emitters.

11. 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.

12. 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

13. What is the Express.js package?

Express is a flexible Node.js web application framework that provides a wide set of features to develop both web and mobile applications

14. What is the control flow function?

The control flow function is a piece of code that runs in between several asynchronous function calls.

15. What are some of the flags used in the read/write operations in files?

flags not found or type unknown

16. Describe Node.js exit codes.

exit codes not found or type unknown

17. Briefly explain the working of Node.js.

Node.js is an entity that runs in a virtual environment, using JavaScript as the primary scripting language. It uses a simple V8 environment to run on, which helps in the provision of features like the non-blocking I/O and a single-threaded event loop.

18. What are the different API functions supported by Node.js? # S

There are two types of API functions. They are as follows:

Synchronous APIs: Used for non-blocking functions

Asynchronous APIs: Used for blocking functions

19. Are there any disadvantages to using Node.js?

A multi-threaded platform can run more effectively and provide better responsiveness when it comes to the execution of intensive CPU computation, and the usage of relational databases with Node.js is becoming obsolete already.

20. How can you import external libraries into Node.js?

External libraries can be easily imported into Node.js using the following command:

```
var http=require ("http")
```

This command will ensure that the HTTP library is loaded completely, along with the exported object.

Next among the Node JS questions, you need to know about event-driven programming.

21. What is the framework that is used majorly in Node.js today?

Node.js has multiple frameworks, namely:

Hapi.js

Express.js

Sails.js

Meteor.js

Derby.js

Adonis.js

22. What is the meaning of a test pyramid?

A test pyramid is a methodology that is used to denote the number of test cases executed in unit testing, integration testing, and combined testing (in that order). This is maintained to ensure that an ample number of test cases are executed for the end-to-end development of a project.

23. What is Libuv?

Libuv is a widely used library present in Node.js. It is used to complement the asynchronous I/O functionality of Node.js. It was developed in-house and used alongside systems such as Luvit, Julia, and more.

Following are some of the features of Libuv:

File system event handling

Child forking and handling

Asynchronous UDP and TCP sockets

Asynchronous file handling and operations

24. What is the difference between spawn and fork methods in Node.js?

The `spawn()` function is used to create a new process and launch it using the command line. What it does is that it creates a node module on the processor. Node.js invokes this method when the child processes return data.

The following is the syntax for the `spawn()` method:

```
child_process.spawn(command[, args][, options])
```

Coming to the `fork()` method, it can be considered as an instance of the already existing `spawn()` method. Spawning ensures that there is more than one active worker node to handle tasks at any given point in time.

The following is the syntax for the `fork()` method:

```
child_process.fork(modulePath[, args][, options])
```

25. What are global objects in Node.js?

Global objects are objects with a scope that is accessible across all of the modules of the Node.js application. There will not be any need to include the objects in every module. One of the objects is declared as global. So, this is done to provide any functions, strings, or objects access across the application.

Next among the Node JS coding questions, you need to take a look at the usage of assets in Node JS.

26. What is the use of REPL in Node.js?

REPL stands for Read-Eval-Print-Loop. It provides users with a virtual environment to test JavaScript code in Node.js.

To launch REPL, a simple command called 'node' is used. After this, JavaScript commands can be typed directly into the command line.

27. Where is package.json used in Node.js?

The 'package.json' file is a file that contains the metadata about all items in a project. It can also be used as a project identifier and deployed as a means to handle all of the project dependencies.

28. What is the use of the crypto module in Node.js?

The crypto module in Node.js is used to provide users with cryptographic functionalities. This provides them with a large number of wrappers to perform various operations such as cipher,

decipher, signing, and hashing operations.

29. What is a passport in Node.js?

Passport is a widely used middleware present in Node.js. It is primarily used for authentication, and it can easily fit into any Express.js-based web application.

With every application created, it will require unique authentication mechanisms. This is provided as single modules by using passport, and it becomes easy to assign strategies to applications based on requirements, thereby avoiding any sort of dependencies.

30. How does the DNS lookup function work in Node.js? #

The DNS lookup method uses a web address for its parameter and returns the IPv4 or IPv6 record, correspondingly.

There are other parameters such as the options that are used to set the input as an integer or an object. If nothing is provided here, both IPv4 and IPv6 are considered. The third parameter is for the callback function.

The syntax is:

```
dns.lookup(address, options, callback)
```

31. What is the use of EventEmitter in Node.js?

Every single object in Node.js that emits is nothing but an instance of the EventEmitter class. These objects have a function that is used to allow the attachment between the objects and the named events.

Synchronous attachments of the functions are done when the EventEmitter object emits an event.

32. What are Data Structures?

Data structures are the methods and techniques used to maintain data in an organized fashion. This is primarily done to ensure that data can be manipulated and accessed in an efficient manner. Data dependency and relationships between two or more entities of data also play a vital role in the concept of data structures.

33. What is a linked list?

A linked list is a data structure that consists of individual entities called nodes. These nodes have the capability to connect to other nodes and create a chain in the process. This continuous chain structure forms a linked list, as the name suggests.

34. Where are Data Structures primarily used?

Data structures are very much needed in almost all of the fields that you can think of. Algorithms are the primary requirement in every data handling situation. Following are some of the scenarios where data structures are widely used:

Numerical computation

Operating system design

Artificial Intelligence

Compiler design

Database handling

Graphical processing

Lexical analysis

Statistics

35. How do you manage packages in your node.js project?

It can be managed by a number of package installers and their configuration file accordingly. Out of them mostly use npm or yarn. Both provide almost all libraries of javascript with extended features of controlling environment-specific configurations. To maintain versions of libs being installed in a project we use package.json and package-lock.json so that there is no issue in porting that app to a different environment.

36. Explain the steps how “Control Flow” controls the functions calls?

Control the order of execution

Collect data

Limit concurrency

Call the following step in the program.

37. What is fork in node JS?

A fork in general is used to spawn child processes. In node it is used to create a new instance of v8 engine to run multiple workers to execute the code.

38. List down the two arguments that async.queue takes as input?

Task Function

Concurrency Value

39. What tools can be used to assure consistent code style?

ESLint can be used with any IDE to ensure a consistent coding style which further helps in maintaining the codebase.

40. Differentiate between `process.nextTick()` and `setImmediate()`?

Both can be used to switch to an asynchronous mode of operation by listener functions.

`process.nextTick()` sets the callback to execute but `setImmediate` pushes the callback in the queue to be executed. So the event loop runs in the following manner

timers→pending callbacks→idle,prepare→connections(poll,data,etc)→check→close callbacks

In this `process.nextTick()` method adds the callback function to the start of the next event queue and `setImmediate()` method to place the function in the check phase of the next event queue.

41. What are node.js buffers?S

In general, buffers are temporary memory that is mainly used by streams to hold on to some data until consumed. Buffers are introduced with additional use cases than JavaScript's `Unit8Array` and are mainly used to represent a fixed-length sequence of bytes. This also supports legacy encodings like ASCII, utf-8, etc. It is a fixed(non-resizable) allocated memory outside the v8.

42. Explain what a Reactor Pattern is in Node.js? S

Reactor pattern again a pattern for nonblocking I/O operations. But in general, this is used in any event-driven architecture.

There are two components in this: 1. Reactor 2. Handler.

Reactor: Its job is to dispatch the I/O event to appropriate handlers

Handler: Its job is to actually work on those events

43. Why should you separate Express app and server?S

The server is responsible for initializing the routes, middleware, and other application logic whereas the app has all the business logic which will be served by the routes initiated by the server. This ensures that the business logic is encapsulated and decoupled from the application logic which makes the project more readable and maintainable.

44. Explain the concept of stub in Node.js?

Stubs are used in writing tests which are an important part of development. It replaces the whole function which is getting tested.

This helps in scenarios where we need to test:

External calls which make tests slow and difficult to write (e.g HTTP calls/ DB calls)

Triggering different outcomes for a piece of code (e.g. what happens if an error is thrown/ if it passes)

For example, this is the function:

```
const request = require('request');

const getPhotosByAlbumId = (id) => {

const requestUrl = `https://jsonplaceholder.typicode.com/albums/${id}/photos?_limit=3`;

return new Promise((resolve, reject) => {

  request.get(requestUrl, (err, res, body) => {

    if (err) {

      return reject(err);

    }

    resolve(JSON.parse(body));

  });

});

};

module.exports = getPhotosByAlbumId;
```

To test this **function this is the stub**

```
const expect = require('chai').expect;

const request = require('request');

const sinon = require('sinon');

const getPhotosByAlbumId = require('./index');

describe('with Stub: getPhotosByAlbumId', () => {

  before(() => {

    sinon.stub(request, 'get')

      .yields(null, null, JSON.stringify([

        {


```

```

        "albumId": 1,

        "id": 1,

        "title": "A real photo 1",

        "url": "https://via.placeholder.com/600/92c952",

        "thumbnailUrl": "https://via.placeholder.com/150/92c952"
    },

    {

        "albumId": 1,

        "id": 2,

        "title": "A real photo 2",

        "url": "https://via.placeholder.com/600/771796",

        "thumbnailUrl": "https://via.placeholder.com/150/771796"
    },

    {

        "albumId": 1,

        "id": 3,

        "title": "A real photo 3",

        "url": "https://via.placeholder.com/600/24f355",

        "thumbnailUrl": "https://via.placeholder.com/150/24f355"
    }
    ]));

});

after(() => {

    request.get.restore();

});

it('should getPhotosByAlbumId', (done) => {

```

```

getPhotosByAlbumId(1).then((photos) => {

  expect(photos.length).to.equal(3);

  photos.forEach(photo => {

    expect(photo).to.have.property('id');

    expect(photo).to.have.property('title');

    expect(photo).to.have.property('url');

  });

  done();

});

});

});

```

45. What is an Event Emitter in Node.js?

EventEmitter is a Node.js class that includes all the objects that are basically capable of emitting events. This can be done by attaching named events that are emitted by the object using an `eventEmitter.on()` function. Thus whenever this object throws an event the attached functions are invoked synchronously.

```
const EventEmitter = require('events');
```

```
class MyEmitter extends EventEmitter {}
```

```
const myEmitter = new MyEmitter();
```

```
myEmitter.on('event', () => {
```

```
  console.log('an event occurred!');
```

```
});
```

```
myEmitter.emit('event');
```

46. What is WASI and why is it being introduced?

Web assembly provides an implementation of WebAssembly System Interface specification through WASI API in node.js implemented using WASI class. The introduction of WASI was done by keeping in mind its possible to use the underlying operating system via a collection of POSIX-like functions thus further enabling the application to use resources more efficiently and features that require system-level access.

47. Why does Node.js prefer *Error-First Callback*?

Answer

The usual pattern is that the callback is invoked as `callback(err, result)`, where only one of `err` and `result` is non-null, depending on whether the operation succeeded or failed. Without this convention, developers would have to maintain different signatures and APIs, without knowing where to place the error in the arguments array.

```
fs.readFile(filePath, function(err, data) {  
  
  if (err) {  
  
    //handle the error  
  
  }  
  
  // use the data object  
  
});
```

48. How does *concurrency* work in Node.js?

Answer

The thing with node.js is that *everything runs concurrently, except for your code*.

So, what that means is that there are actually lots of threads running inside Node.js virtual machine (or a thread pool if you wish), and those threads are utilized whenever you call an async function like performing i/o operations on files, accessing databases, requesting urls, etc.

However, for your code, there is only a single thread, and it processes events from an event queue. So, when you register a callback its reference is actually passed to the background worker thread, and once the async operation is done, new event is added to the event-queue with that callback

When Node gets I/O request it creates or uses a thread to perform that I/O operation and once the operation is done, it pushes the result to the event queue. On each such event, event loop runs and checks the queue and if the execution stack of Node is empty then it adds the queue result to execution stack.

This is how Node manages concurrency.

49. Explain usage of **NODE_ENV**

Node encourages the convention of using a variable called `NODE_ENV` to flag whether we're in production right now. This determination allows components to provide better diagnostics during development, for example by disabling caching or emitting verbose log statements. Setting `NODE_ENV` to production makes your application roughly 3 times faster.

```
// Setting environment variables in bash before starting the node process
```

```
$ NODE_ENV=development
```

```
$ node
```

And reading variable:

```
if (process.env.NODE_ENV === "production")
```

```
    useCaching = true;
```

50. Provide some example of config file separation for dev and prod environments

Answer

A perfect and flawless configuration setup should ensure:

keys can be read from file AND from environment variable

secrets are kept outside committed code

config is hierarchical for easier findability

Consider the following config file:

```
var config = {  
  production: {  
    mongo : {  
      billing: '*****'  
    }  
  },  
  default: {  
    mongo : {  
      billing: '*****'  
    }  
  }  
}
```

```
exports.get = function get(env) {
```

```
    return config[env] || config.default;
}
```

And it's usage:

```
const config = require('./config/config.js').get(process.env.NODE_ENV);

const dbconn = mongoose.createConnection(config.mongo.billing);
```

51. What are LTS releases of Node.js? Why should you care?

Answer

An LTS(Long Term Support) version of Node.js receives all the critical bug fixes, security updates and performance improvements.

LTS versions of Node.js are supported for at least 18 months and are indicated by even version numbers (e.g. 4, 6, 8). They're best for production since the LTS release line is focussed on stability and security, whereas the *Current* release line has a shorter lifespan and more frequent updates to the code. Changes to LTS versions are limited to bug fixes for stability, security updates, possible npm updates, documentation updates and certain performance improvements that can be demonstrated to not break existing applications.

52. Explain Node.js web application architecture?

A web application distinguishes into 4 layers:

Client Layer: The Client layer contains web browsers, mobile browsers or applications which can make an HTTP request to the web server.

Server Layer: The Server layer contains the Web server which can intercept the request made by clients and pass them the response.

Business Layer: The business layer contains application server which is utilized by the web server to do required processing. This layer interacts with the data layer via database or some external programs.

Data Layer: The Data layer contains databases or any source of data.

53.What do you understand by the first class function in JavaScript?

When functions are treated like any other variable, then those functions are called first-class functions. Apart from JavaScript, many other programming languages, such as Scala, Haskell, etc. follow this pattern. The first class functions can be passed as a param to another function (callback), or a function can return another function (higher-order function). Some examples of higher-order functions that are popularly used are `map()` and `filter()`.

54. Why is Node.js Single-threaded?

Node.js is a single-threaded application with event looping for async processing. The biggest advantage of doing async processing on a single thread under typical web loads is that you can achieve more performance and scalability than the typical thread-based implementation.

55. What are some commonly used timing features of Node.js?

Following is a list of some commonly used timing features of Node.js:

setTimeout/clearTimeout: This timing feature of Node.js is used to implement delays in the code execution.

setInterval/clearInterval: The setInterval or clearInterval timing feature is used to run a code block multiple times in the application.

setImmediate/clearImmediate: This timing feature of Node.js is used to set the execution of the code at the end of the event loop cycle.

nextTick: This timing feature sets the execution of code at the beginning of the next event loop cycle.

56. What do you understand by the term fork in Node.js?

Generally, a fork is used to spawn child processes. In Node.js, it is used to create a new instance of the V8 engine to run multiple workers to execute the code.

57. What are buffers in Node.js?

In general, a buffer is a temporary memory mainly used by the stream to hold on to some data until it is consumed. Buffers are used to represent a fixed-size chunk of memory allocated outside of the V8 JavaScript engine. It can't be resized. It is like an array of integers, which each represents a byte of data. It is implemented by the Node.js Buffer class. Buffers also support legacy encodings like ASCII, utf-8, etc.

58. What is a control flow function?

Control flow function is a generic piece of code that runs in between several asynchronous function calls.

59. How "Control Flow" controls the functions calls?

The control flow does the following job:

Control the order of execution

Collect data

Limit concurrency

Call the next step in a program

60. Is it possible to access DOM in Node?

No, it is not possible to access DOM in Node.

61. Explain the tasks of terms used in Node REPL.

Following are the terms used in REPL with their defined tasks:

Read: It reads user's input; parse the input into JavaScript data-structure and stores in memory.

Eval: It takes and evaluates the data structure.

Print: It is used to print the result.

Loop: It loops the above command until user press ctrl-c twice to terminate.

62. What is the difference between operational and programmer errors?

Operational errors are not bugs, but create problems with the system like request timeout or hardware failure. On the other hand, programmer errors are actual bugs.

63. What tools can be used to assure a consistent style in Node.js?

Following is a list of tools that can be used in developing code in teams, to enforce a given style guide and to catch common errors using static analysis.

JSLint

JSHint

ESLint

JSCS

64. What is the role of assert in Node.js?

The Node.js Assert is a way to write tests. It provides no feedback when running your test unless one fails. The assert module provides a simple set of assertion tests that can be used to test invariants. The module is intended for internal use by Node.js, but can be used in application code via require ('assert'). For example:

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

```
function add (a, b) {
```

```
    return a + b;
```

```
}
```

```
var expected = add(1,2);
```

```
assert( expected === 3, 'one plus two is three');
```

65. What does Node.js TTY module contains?

The Node.js TTY module contains `tty.ReadStream` and `tty.WriteStream` classes. In most cases, there is no need to use this module directly. You have to use `require('tty')` to access this module.

66. What is the concept of Punycode in Node.js?

In Node.js, the concept of Punycode is used for converting one type of string into another type. Punycode is an encoding syntax used for converting Unicode (UTF-8) string of characters into a basic ASCII string of characters. Now, the hostnames can only understand the ASCII characters so, after the Node.js version 0.6.2 onwards, it was bundled up with the default Node package.

To use it with any previous versions, you have to use the following code:

Syntax:

```
punycode = require('punycode');
```

67. 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.

68. 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.

69. What is typically the first argument passed to a Node.js callback handler?

Node.js core modules, as well as most of the community-published ones, follow a pattern whereby the first argument to any callback handler is an optional error object. If there is no error, the argument will be null or undefined.

A typical callback handler could therefore perform error handling as follows:

```
function callback(err, results) {
```

```
// usually we'll check for the error before handling results

if(err) {

    // handle error somehow and return

}

// no error, perform standard callback handling

}
```

70. What is libuv?

Answer

libuv is a C library that is used to abstract non-blocking I/O operations to a consistent interface across all supported platforms. It provides mechanisms to handle file system, DNS, network, child processes, pipes, signal handling, polling and streaming. It also includes a thread pool for offloading work for some things that can't be done asynchronously at the operating system level.

71. Why we always require modules at the top of a file? Can we require modules inside of functions?

Answer

Yes, we can but we shall never do it.

Node.js always runs require *synchronously*. If you require an external module from within functions your module will be synchronously loaded when those functions run and this can cause two problems:

If that module is only needed in one route handler function it might take some time for the module to load synchronously. As a result, several users would be unable to get any access to your server and requests will queue up.

If the module you require causes an error and crashes the server you may not know about the error.

72. What is Chaining in Node?

Chaining is a mechanism to connect output of one stream to another stream and create a chain of multiple stream operations. It is normally used with piping operations.

73. How will you delete a directory?

Parameters

Here is the description of the parameters used:

path ? This is the directory name including path.

callback ? This is the callback function which gets no arguments other than a possible exception are given to the completion callback.

74. What's the difference between operational and programmer errors?

Answer: Operation errors are not bugs, but problems with the system, like *request timeout* or *hardware failure*. On the other hand programmer errors are actual bugs.

75. What is the difference between Nodejs, AJAX, and jQuery?

Answer: The one common trait between Node.js, AJAX, and jQuery is that all of them are the advanced implementation of JavaScript. However, they serve completely different purposes.

Node.js –It is a server-side platform for developing client-server applications. For example, if we've to build an online employee management system, then we won't do it using client-side JS. But the Node.js can certainly do it as it runs on a server similar to Apache, Django not in a browser.

AJAX (aka Asynchronous Javascript and XML) –It is a client-side scripting technique, primarily designed for rendering the contents of a page without refreshing it. There are a no. of large companies utilizing AJAX such as Facebook and Stack Overflow to display dynamic content.

jQuery –It is a famous JavaScript module which complements AJAX, DOM traversal, looping and so on. This library provides many useful functions to help in JavaScript development. However, it's not mandatory to use it but as it also manages cross-browser compatibility, so can help you produce highly maintainable web applications.

76. How you can monitor a file for modifications in Node.js ?

Answer: We can take advantage of File System watch() function which watches the changes of the file.

77. What is libuv?

Answer: libuv is a C library that is used to abstract non-blocking I/O operations to a consistent interface across all supported platforms. It provides mechanisms to handle file system, DNS, network, child processes, pipes, signal handling, polling and streaming. It also includes a thread pool for offloading work for some things that can't be done asynchronously at the operating system level.

78. What is Chaining in Node?

Answer: Chaining is a mechanism to connect output of one stream to another stream and create a chain of multiple stream operations. It is normally used with piping operations.

79. What is the purpose of the setTimeout function?

Answer: The setTimeout(cb, ms) global function is used to run callback cb after at least ms milliseconds. The actual delay depends on external factors like OS timer granularity and system load. A timer cannot span more than 24.8 days.

80. How does Node.js handle child threads?

Answer: Node.js, in its essence, is a single thread process. It does not expose child threads and thread management methods to the developer. Technically, Node.js does spawn child threads for certain tasks such as asynchronous I/O, but these run behind the scenes and do not execute any application JavaScript code, nor block the main event loop.

If threading support is desired in a Node.js application, there are tools available to enable it, such as the ChildProcess module.

81. What is the preferred method of resolving unhandled exceptions in Node.js? ???

Answer: Unhandled exceptions in Node.js can be caught at the Process level by attaching a handler for uncaughtException event.

```
process.on('uncaughtException', function(err) {  
  
    console.log('Caught exception: ' + err);  
  
});
```

However, uncaughtException is a very crude mechanism for exception handling and may be removed from Node.js in the future. An exception that has bubbled all the way up to the Process level means that your application, and Node.js may be in an undefined state, and the only sensible approach would be to restart everything.

The preferred way is to add another layer between your application and the Node.js process which is called the [domain](#).

Domains provide a way to handle multiple different I/O operations as a single group. So, by having your application, or part of it, running in a separate domain, you can safely handle exceptions at the domain level, before they reach the Process level.

82. What are the global objects of Node.js?

Answer: These objects are available in all modules:

process - The process object is a global that provides information about, and control over, the current Node.js process.

console - Used to print to stdout and stderr.

buffer - Used to handle binary data.

83. Name some of the events fired by streams. Answer: Each type of Stream is an EventEmitter instance and throws several events at different instance of times. For example, some of the commonly used events are:

data - This event is fired when there is data is available to read.

end - This event is fired when there is no more data to read.

finish - This event is fired when all data has been flushed to underlying system

Answer: The `__filename` represents the filename of the code being executed. This is the resolved absolute path of this code file. For a main program this is not necessarily the same filename used in the command line. The value inside a module is the path to that module file.

Answer: Run the application on any port above 1024, then put a reverse proxy like [nginx](#) in front of it.

JSLint by Douglas Crockford

JSHint

ESLint

JSCS

Answer: Yes, Node.js would run on a multi-core system without any issue. But it is by default a single-threaded application, so it can't completely utilize the multi-core system.

However, Node.js can facilitate deployment on multi-core systems where it does use the additional hardware. It packages with a Cluster module which is capable of starting multiple Node.js worker processes that will share the same port.

Answer: We can use Node.js for a variety of applications. But it is a single threaded framework, so we should not use it for cases where the application requires long processing time. If the server is doing some calculation, it won't be able to process any other requests. Hence, Node.js is best when processing needs less dedicated CPU time.

Answer: Buffer is used to process binary data, such as pictures, mp3, database files, etc. Buffer supports a variety of encoding and decoding, binary string conversion.

90. Why to use Buffers instead of binary strings to handle binary data ?

Answer: Pure JavaScript does not able to handle straight binary data very well. Since Node.js servers have to deal with TCP streams for reading and writing of data, binary strings will become problematic to work with as it is very slow and has a tendency to break. That's why it is always advisable to use Buffers instead of binary strings to handle binary data.

91. How do you debug Node.js applications?

Answer: Node has its own built in GUI debugger as of version 6.3 (using Chrome's DevTools).

92. What is Piping in Node?

Answer: Piping is a mechanism to connect output of one stream to another stream. It is normally used to get data from one stream and to pass output of that stream to another stream. There is no limit on piping operations.

93. How would you scale Node application? ?????

Answer: We can scale Node application in following ways:

cloning using *Cluster* module.

decomposing the application into smaller services – i.e micro services.

94. What is the difference between process.nextTick() and setImmediate() ?

Answer: The difference between process.nextTick() and setImmediate() is that process.nextTick() defers the execution of an action till the next pass around the event loop or it simply calls the callback function once the ongoing execution of the event loop is finished whereas setImmediate() executes a callback on the next cycle of the event loop and it gives back to the event loop for executing any I/O operations.

95.Explain what is Reactor Pattern in Node.js?

Answer: Reactor Pattern is an idea of non-blocking I/O operations in Node.js. This pattern provides a handler(in case of Node.js, a *callback function*) that is associated with each I/O operation. When an I/O request is generated, it is submitted to a *demultiplexer*.

This *demultiplexer* is a notification interface that is used to handle concurrency in non-blocking I/O mode and collects every request in form of an event and queues each event in a queue. Thus, the demultiplexer provides the *Event Queue*.

At the same time, there is an Event Loop which iterates over the items in the Event Queue. Every event has a callback function associated with it, and that callback function is invoked when the Event Loop iterates.