Learning Outcome

**Able to develop the real time scenarios based on Node JS applications.**

# Angular JS Pipe

## What is Pipe?

The pipe symbol (|) is used for applying filters in AngularJS. A filter is a function that is invoked for handling model transformations. Its basically just a global function that doesn't require registration of functions on a scope, and offers more convenient syntax to regular function calls.

A pipe class must implement the PipeTransform interface. For example, if the name is "myPipe", use a template binding expression such as the following:

{{ exp | myPipe }}

The result of the expression is passed to the pipe's transform() method.

A pipe must belong to an NgModule in order for it to be available to a template. To make it a member of an NgModule, list it in the declarations field of the NgModule metadata.

Angular Pipes are used to transform data on a template, without writing a boilerplate code in a component.

## Why Pipe?

A pipe takes in data as input and transforms it to the desired output. It is like a filter in Angular 1 (AngularJS).

Generally, If we need to transform data, we write the code in the component, For example, we want to transform today’s date into a format like '16 Apr 2018' or '16-04-2018', We need to write separate code in the component.

So instead of writing separate boilerplate code, we can use the built-in pipe called DatePipe which will take input and transform it into the desired date format.

We can use the pipe on a template using Pipe Operator | .

As shown below,

{{today | date : ‘fullDate’}}

This interpolation give output in Monday, April 16, 2018 date format.

here,

* today is the component variable, which specifies the current date.
* date represent DataPipe
* fullDate is an optional parameter or argument which specifies the date format.

Angular comes with a set of built-in pipes such as DatePipe, UpperCasePipe, LowerCasePipe. Other than this, We can also create our own custom pipe.

## Built-In Pipes

Angular comes with a collection of built-in pipes such as:

1. DatePipe
2. UpperCasePipe
3. LowerCasePipe
4. CurrencyPipe
5. DecimalPipe
6. PercentPipe

Built-in Angular Pipes are defined in @angular/common package.

## Custom Filter:

Sometimes the built-in filters in Angular cannot meet the needs or requirements for filtering output. In such a case, an AngularJS custom filter can be created, which can pass the output in the required manner.

Similarly, for numbers, you can use other filters. During this tutorial, we will see the different standard built-in filters available in Angular.

## Pipe Chaining

**What Is Pipe Channing?**

The chaining Pipe is used to perform the multiple operations within the single expression. This chaining operation will be chained using the pipe (I).

In the following example, to display the birthday in the upper case- will need to use the inbuilt date-pipe and upper-case-pipe.

In the following example –

{{ birthday | date | uppercase}}

<!-- The output is - MONDAY, MARCH 10, 1984 -->

**What Is Parameterizing Pipe?**

A pipe can accept any number of optional parameters to achieve output. The parameter value can be any valid template expressions. To add optional parameters follow the pipe name with a colon (:). Its looks like- currency: 'INR'

In the following example –

<h2>The birthday is - {{ birthday | date:"MM/dd/yy" }} </h2>

<!-- Output - The birthday is - 10/03/1984 -->

# Angular js Routing

Routing is a core feature in AngularJS. This feature is useful in building SPA (Single Page Application) with multiple views. In SPA application, all views are different Html files and we use Routing to load different part of application and its help to divide application logically and make it manageable. In other words, Routing helps us to divide our application in logical views and bind them with different controllers.

## Introduction to ngRoute Module

This module provides routing in AngularJS application and also provides deep linking services and directives. To achieve routing in AngularJS, we need to include the library file of ngRoute.

### **Component of ngRoute Module**

There are four main components of ngRoute module:

1. ngView: ngView is directive and creates new scope. It is used to load html templates.
2. $routeProvider: It is used to configure routes.
3. $route: It is used to make deep linking URLs between controllers and view. $route watches $location.url() and tries to map the path to an existing route defined by $routeProvider
4. $routeParams: This is Angular service which allows us to retrieve the current set of route parameters.

$routeProvider used for configuring the routes in AngularJS application. It is depends on ngRoute module. All application routes are defined via $routeProvider and it is the provider of the $route Service. It is very easy to wire up controllers, view templates and browser URL location using $route service. This service also help us to implement deep linking that utilize the browser back and forward navigation (browser’s history).

**Hello World Example**

Routing in AngularJS is used to load different templates at runtime. In the following example we will elaborate it more step by step.

**Step 1: Create basic structure of application (SPA).**

In this step, I have created demo.html and it includes all the required AngularJS library and bootstrap library. Here I have also created structure for application. Also defined two links: page1 and page2. Each link loads respective template.

**Project structure**

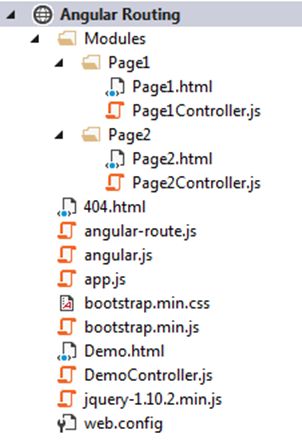


Image: Project structure

Reference: <https://csharpcorner-mindcrackerinc.netdna-ssl.com/UploadFile/ff2f08/routing-in-angularjs/Images/Project%20structure.jpg>

Demo.html

<!DOCTYPEhtml>

<htmldata-ng-apphtmldata-ng-app="AngularApp">

<head>

<metacontentmetacontent="IE=edge, chrome=1" http-equiv="X-UA-Compatible" />

<title>AngularJS - Routing</title>

<scriptsrcscriptsrc="angular.js">

</script>

<scriptsrcscriptsrc="angular-route.js">

</script>

<scriptsrcscriptsrc="app.js">

</script>

<scriptsrcscriptsrc="jquery-1.10.2.min.js">

</script>

<scriptsrcscriptsrc="DemoController.js">

</script>

<scriptsrcscriptsrc="Modules/Page1/Page1Controller.js">

</script>

<scriptsrcscriptsrc="Modules/Page2/Page2Controller.js">

</script>

<scriptsrcscriptsrc="bootstrap.min.js">

</script>

<linkhreflinkhref="bootstrap.min.css" rel="stylesheet" /> </head>

<body>

<div>

<div ng-controller="demoController" class="container">

<p><b>Hello World - Routing Example</b></p>

<divclassdivclass="row">

<divclassdivclass="col-md-3">

<ulclassulclass="nav">

<li>

<ahrefahref="#/page1"> Page 1 </a>

</li>

<li>

<ahrefahref="#/page2"> Page 2 </a>

</li>

</ul>

</div>

<divclassdivclass="col-md-9">

<divng-view>

</div>

</div>

</div>

</div>

</div>

</body>

</html>

Here I have divided screen in two sections: Left contains the menu and in the right pane respective template will be loaded.

The ngView directive is responsible to render the template of current route into the main layout file passed by $route service. I have also defined ng-app directive once. The ngView become place holder for views. Every view render by the route is loaded into this section.

**Step 2: Add Routing**

In the Demo.html, I have included app.js file which hold the definition of AngularJS application. The $routeProvider definition contain by the module is called "ngRoute". In app.js file, I have defined an angular app using “angular. Module” method. After creating module, we need to configure the routes. The "config" method is used to configure $routeProvider. Using "when" and "otherwise" method of $routeProvider, we can define the route for our AngularJS application.

**app.js**

var app = angular.module("AngularApp", ['ngRoute']);

app.config(['$routeProvider',

function ($routeProvider)

{

$routeProvider.

when('/page1',

{

templateUrl: 'Modules/Page1/page1.html',

controller: 'Page1Controller'

})

.

when('/page2',

{

templateUrl: 'Modules/Page2/page2.html',

controller: 'Page2Controller'

})

.

otherwise(

{

redirectTo: '/page1'

});

}

]);

In the above code, I have defined two routes: "page1" and "page2" and mapped them with template view "Modules/Page1/page1.html" and "Modules/Page2/page2.html" respectively. I have set default page using "otherwise" method.

**Step 3: Define HTML Template and controller**

In this example, I have added two html templates: page1.html and page2.html and created two controllers files: page1controller.js and page2controller.js. The following are the definition of both HTML template and controller.

**Page1.html**

<divng-controller="Page1Controller">

<h2>Page 1</h2> Hi, {{myName}} </div>

**Page1Controller.js**

app.controller("Page1Controller", ['$scope', function ($scope)

{

$scope.myName = "Tejas Trivedi";

}]);

Page2.html

<divng-controller="Page2Controller">

<h2>Page 2</h2> Hi, {{myName}} </div>

**Page1Controller.js**

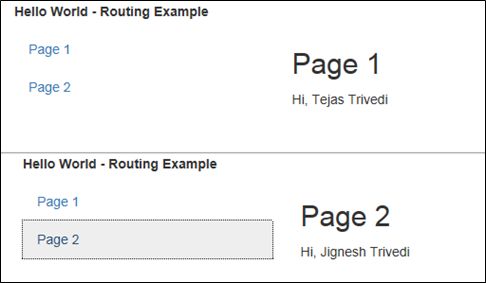
app.controller("Page2Controller", ['$scope', function ($scope)

{

$scope.myName = "Jignesh Trivedi";

}]);

**Output**



## Pass parameters to route URL

We can also define parameters in the route URL. Parameter name should define by colon (:) after the route path. The $routeParams service allow us to retrieve the route parameters in controller.

This feature is useful when we want to use same view for two or more different purpose and it can be identified by the parameter value or we want to display detail of any master record based on master id and master id passed as parameter. In Angular, route parameter can be defined using parameter name in URL. For example,

**app.js**

.when('/page3/:id',

{

templateUrl: 'Modules/Page3/page3.html',

controller: 'Page3Controller'

})

We can read parameter value in controller by using $routeParams. Note that do not forget to inject $routeParam service in controller.

**page3Controller.js**

app.controller("Page3Controller", ['$scope', '$routeParams', function ($scope, $routeParams)

{

$scope.myName = "Jignesh Trivedi";

$scope.id = $routeParams.id;

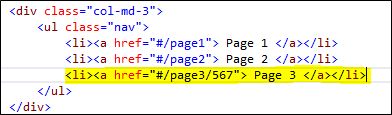
}]);

**page3.html**

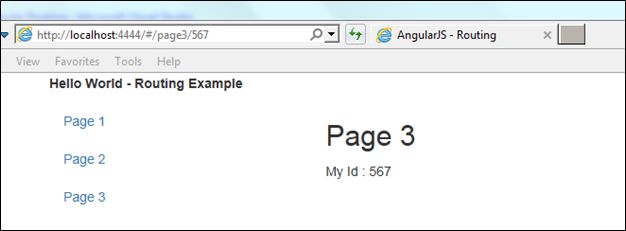
<divng-controller="Page2Controller">

<h2>Page 3</h2> My Id : {{id}} </div>

Demo.html



**Output**



## Load local views

This is not always required to load view from the different html files because view templates are very small and we might want to keep them within main html file instead of creating separate html files. The ng-template can be used to define small templates in the main html file.

Syntax

<script type="text/ng-template" id="page4.html">

//define definition of html here...

</script>

Here I have defined a template "page4.html" and "page5.html" inside the script tag. AngularJS will automatically load these templates in ng-view when "page4.html" or "page5.html" is referred in route.

DemoLocalView.html defines the structure of the application. The appLocalView.js is very similar to previous example. Here I have used same controller js files.

**appLocalView.js**

var app = angular.module("AngularApp", ['ngRoute']);

app.config(['$routeProvider',

function ($routeProvider)

{

$routeProvider.

when('/page4',

{

templateUrl: 'page4.html',

controller: 'Page1Controller'

})

.

when('/page5',

{

templateUrl: 'page5.html',

controller: 'Page2Controller'

})

.

otherwise(

{

redirectTo: '/page4'

});

}

]);

**DemoLocalView.html**

<!DOCTYPE html>

<html data-ng-app="AngularApp">

<head>

<meta content="IE=edge, chrome=1" http-equiv="X-UA-Compatible" />

<title>AngularJS - Routing</title>

<script src="angular.js"></script>

<script src="angular-route.js"></script>

<script src="appLocalView.js"></script>

<script src="jquery-1.10.2.min.js"></script>

<script src="Modules/Page1/Page1Controller.js"></script>

<script src="Modules/Page2/Page2Controller.js"></script>

<script src="bootstrap.min.js"></script>

<link href="bootstrap.min.css" rel="stylesheet" /> </head>

<body>

<div>

<div class="container">

<p><b>Load local view Example</b></p>

<div class="row">

<div class="col-md-3">

<ul class="nav">

<li><a href="#page4"> Page 4 </a></li>

<li><a href="#page5"> Page 5 </a></li>

</ul>

</div>

<div class="col-md-9">

<div ng-view></div>

</div>

</div>

<script type="text/ng-template" id="page4.html">

<h2> Page 4 </h2> Hi, {{myName}} </script>

<script type="text/ng-template" id="page5.html">

<h2> Page 5 </h2> Hi, {{myName}} </script>

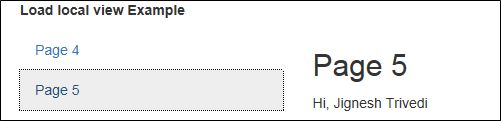
</div>

</div>

</body>

</html>

**Output**



# Angular js Services

## What are AngularJS Services?

AngularJs service is a function, which can use for the business layer of an application. It is like a constructor function that will invoke only once at runtime with new. Services in AngularJS are stateless and singleton objects because we get only one object in it regardless of which application interface created it.

We can use it to provide functionality in our web application. Each service performs a specific task.

## When to use Services in AngularJS?

We can use AngularJS services when we want to create things that act as an application interface. It can be used for all those purposes for which constructor is used.

## Types of AngularJS Services

There are two types of services in angular:

* Built-in services – There are approximately 30 built-in services in angular.
* Custom services – In angular if the user wants to create its own service he/she can do so.

### **Built-in Services in AngularJS**

They are pre-built services in AngularJS. These services get registered automatically at runtime with the dependency injector. Therefore, by using dependency injector we can easily incorporate these built-in services in our angular application.

The various built-in services are as follows:

1. $http: It is a service to communicate with a remote server. It makes an ajax call to the server.
2. **$interval:** It is a wrapper in angular for window.setInterval.
3. **$timeout:** It is the same as setTimeout function in javascript. To set a time delay on the execution of a function $timeout is used.
4. **$anchorscroll:** The page which specifies by an anchor in $location.hash() scrolls using $anchorscroll.
5. **$animate:** It consists of many DOM (Document Object Model) utility methods that provide support for animation hooks.
6. **$animateCss:** It will perform animation only when ngAnimate includes, by default.
7. **$cacheFactory**: It is a factory that constructs cache objects. It puts the key-value pair and retrieves the key-value pair. Also, it can provide access to other services.
8. **$templateCache:** Whenever a template is used for the first time, it is loaded in template cache. It helps in quick retrieval.
9. **$compile:** We can compile HTML string or DOM in the template by it. Also, it produces a template function which will use to link template and scope together.
10. **$controller:** By using a $controller, we can instantiate Angular controller components.
11. **$document:** J-query wrapped the reference to the window. Document element is specified by it.
12. **$exceptionHandler:** Any uncaught exception in an angular expression is sent to this service.
13. **$filter:** Use to format the data for displaying to the user.
14. **$httpParamSerializer:** It converts an object to a string.
15. **$httpParamSerializerJQLike:** We can sort Params in alphabetic order. It follows j-query’s param() method logic.
16. **$xhrFactory:** XMLHttpRequest objects is created using factory function.
17. **$httpBackend:** Browser incompatibilities can be handled using it.
18. **$inerpolate:** Use for data binding by HTML $compile service.
19. **$locale :** For various angular components, localization rules can be provided using $locale.
20. **$location**: URL in the address bar of a browser is parsed by it and then the URL is made available to your application. Changes to the URL in the address bar is reflected in $location service and vice-versa.
21. **$log:**It is a console logger.
22. **$parse:** Use to convert Angular expression into a function using $parse.
23. **$q:** A function can run asynchronously using $q and its return value, which will use when they finish the processing.
24. $**rootElement:** It is a root element of the angular application.
25. **$rootScopeUse** in an angular application.
26. **$sceDelegate:** Use in the backend by $sce.

### **Customs Services in AngularJS**

We can create our own service by connecting it with a module in AngularJS. And to use it add it as a dependency while defining controller.

# Angular js Http Request

AngularJS provides $http control which works as a service to read data from the server. $http is an AngularJS service for reading data from remote servers. The $http is a core AngularJS service that is used to communicate with the remote HTTP service via browser’s XMLHttpRequest object or via JSONP.

Syntax:

$http({

method: 'Method\_Name',

url: '/someUrl'

}).then(function successCallback(response) {

//when the response is available, this callback will be called asynchronously

}, function errorCallback(response) {

// this method will called when server returns response with an error status.

});

The $http service is function that takes a configured object to generate a HTTP request and return the response. This response contains data, status code, header, configuration object and status text. In $http the first function executes on successful callback and the second function xeecutes on error.

**Example 1:**

<html>

<head>

<title>Angular JS Includes</title>

<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.2.15/angular.min.js">

</script>

<script src="angular.min.js">

</script>

</head>

<body>

<h2>AngularJS Ajax Demo</h2>

<div ng-app="app" ng-controller="Employee"> <span>{{Message}}</span><br/> <span>{{Status}}</span><br/> <span>{{Headers}}</span><br/> <span>{{Config}}</span><br/> <span>{{StatusText}}</span><br/> </div>

<script>

var obj = angular.module('app', []);

obj.controller('Employee', function ($scope, $http)

{

$http(

{

method: 'GET',

url: 'index.html'

}).then(function successCallback(response)

{

$scope.Message = response.data;

$scope.Status = response.status;

$scope.Headers = response.headers;

$scope.Config = response.config;

$scope.StatusText = response.statusText;

}, function errorCallback(response)

{

alert("UnsuccessFull call!");

});

});

</script>

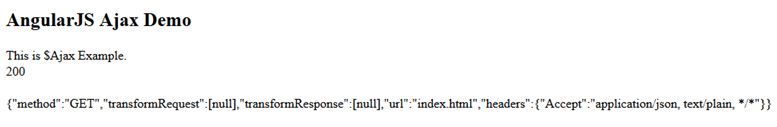
</body>

</html>

**Index.HTML**

This is $Ajax Example.

**Output**



In the above example we use a $ajax service and define the “url” and “method” property of $ajax. On successful Callback method we are showing data of “index.html” page and on error we define the alert message of failure.

## Methods

In the above example we used the .get shortcut method for $ajax service . There are also other shortcut methods.

* $http.get
* $http.post
* $http.head
* $http.put
* $http.delete
* $http.patch
* $http.jsonp

## Property

The response from the server is retrieved as an object and this object contains the following properties:

|  |  |
| --- | --- |
| **Property** | **Description** |
| .config | The configuration object that was used to generate the request. |
| .status | Status number defining the HTTP status. |
| .data | The response body transformed with the transform functions. |
| .headers | Function to use to get header information. |
| .statusText | HTTP status text of the response. |

# Node Introduction

## What is Node.js?

Node.js is a server-side platform built on Google Chrome's JavaScript Engine (V8 Engine). Node.js was developed by Ryan Dahl in 2009 and its latest version is v0.10.36. The definition of Node.js as supplied by its official documentation is as follows –

* Node.js is a platform built on Chrome's JavaScript runtime for easily building fast and scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

Node.js is an open source, cross-platform runtime environment for developing server-side and networking applications. Node.js applications are written in JavaScript, and can be run within the Node.js runtime on OS X, Microsoft Windows, and Linux.

Node.js also provides a rich library of various JavaScript modules which simplifies the development of web applications using Node.js to a great extent.

Node.js = Runtime Environment + JavaScript Library

## Features of Node.js

Following are some of the important features that make Node.js the first choice of software architects.

* Asynchronous and Event Driven − All APIs of Node.js library are asynchronous, that is, non-blocking. It essentially means a Node.js based server never waits for an API to return data. The server moves to the next API after calling it and a notification mechanism of Events of Node.js helps the server to get a response from the previous API call.
* Very Fast − Being built on Google Chrome's V8 JavaScript Engine, Node.js library is very fast in code execution.
* Single Threaded but Highly Scalable − Node.js uses a single threaded model with event looping. Event mechanism helps the server to respond in a non-blocking way and makes the server highly scalable as opposed to traditional servers which create limited threads to handle requests. Node.js uses a single threaded program and the same program can provide service to a much larger number of requests than traditional servers like Apache HTTP Server.
* No Buffering − Node.js applications never buffer any data. These applications simply output the data in chunks.
* License − Node.js is released under the MIT license.

## Who Uses Node.js?

Following is the link on github wiki containing an exhaustive list of projects, application and companies which are using Node.js. This list includes eBay, General Electric, GoDaddy, Microsoft, PayPal, Uber, Wikipins, Yahoo!, and Yammer to name a few.

Projects, Applications, and Companies Using Node

## Concepts

The following diagram depicts some important parts of Node.js which we will discuss in detail in the subsequent chapters.



Image: Node.js Concepts

Reference: https://www.tutorialspoint.com/nodejs/images/nodejs\_concepts.jpg

## Where to Use Node.js?

Following are the areas where Node.js is proving itself as a perfect technology partner.

* I/O bound Applications
* Data Streaming Applications
* Data Intensive Real-time Applications (DIRT)
* JSON APIs based Applications
* Single Page Applications

## Where Not to Use Node.js?

It is not advisable to use Node.js for CPU intensive applications.

## Advantages of NodeJS:

## Here are the benefits of using Node.js

* Easy Scalability: Developers prefer to use Node.js because it is easily scaling the application in both horizontal and vertical directions. We can also add extra resources during the scalability of the application.
* Real-time web apps: If you are building a web app you can also use PHP, and it will take the same amount of time when you use Node.js, But if I am talking about building chat apps or gaming apps Node.js is much more preferable because of faster synchronization. Also, the event loop avoids HTTP overloaded for Node.js development.
* Fast Suite: NodeJs runs on the V8 engine developed by Google. Event loop in NodeJs handles all asynchronous operation so NodeJs acts like a fast suite and all the operations can be done quickly like reading or writing in the database, network connection, or file system
* Easy to learn and code: NodeJs is easy to learn and code because it uses JavaScript. If you are a front-end developer and have a good grasp of JavaScript you can easily learn and build the application on NodeJS
* Advantage of Caching: It provides the caching of a single module. Whenever there is any request for the first module, it gets cached in the application memory, so you don’t need to re-execute the code.
* Data Streaming: In NodeJs HTTP request and response are considered as two separate events. They are data stream so when you process a file at the time of loading it will reduce the overall time and will make it faster when the data is presented in the form of transmissions. It also allows you to stream audio and video files at lightning speed.
* Hosting: PaaS (Platform as a Service) and Heroku are the hosting platforms for NodeJS application deployment which is easy to use without facing any issue.
* Corporate Support: Most of the well-known companies like Walmart, Paypal, Microsoft, Yahoo are using NodeJS for building the applications. NodeJS uses JavaScript, so most of the companies are combining front-end and backend Teams together into a single unit.

## Application of NodeJS:

NodeJS should be preferred to build:

* Real-Time Chats,
* Complex Single-Page applications,
* Real-time collaboration tools,
* Streaming apps
* JSON APIs based application

## Installing Node and using It:

### **Using Website:**

1. You can visit the link Download Node and download LTS version.
2. After installing the node you can check your node version in command prompt using command..

~ $node --version

1. After that, you can just create a folder and add a file here for example app.js. To run this file you need to execute command…

first app $node app.js

1. Node Modules: There are some built-in modules that you can use to create your applications. Some popular modules are- OS, fs, events, HTTP, URL and then you can include these modules in your file using these lines.

var fs = require('fs');

1. Here is an example of how to include an HTTP module to build the server…

var http = require('http');

// Create a server object:

http.createServer(function (req, res) {

// Write a response to the client

res.write('GeeksForGeeks');

// End the response

res.end();

// The server object listens on port 8080

}).listen(8080);

This will listen to the server on port 8080. Once you will run your file in command prompt it will execute your file and listen to the server on this port. You can also create your own module and include it in your file.

### **Using NPM:**

NPM is a Node Package Manager that provides packages to download and use. It contains all the files and modules that you require in your application. To install any package you need to execute a command…

npm install

This is an example of using the Events module.

var events = require('events');

var eventEmitter = new events.EventEmitter();

// Create an event handler:

var myEventHandler = function () {

console.log('Welcome to GeeksforGeeks');

}

// Assign the event handler to an event:

eventEmitter.on('geeks', myEventHandler);

// Fire the 'geeks' event:

eventEmitter.emit('geeks');

So this is how you can start with node and build your own applications. There are some frameworks of the node which you can use to build your applications. Some popular frameworks of node are…Express.js, Socket.io, Koa.js, Meteor.js, Sail.js.

# Blocking & Nonblocking code

There are two types of execution of our code, synchronous and asynchronous. The code is executed in sequence synchronously, and execution is awaited when the function is called. In asynchronous execution, the line is not necessarily followed, and completion of the operation is not necessary. Based on this, we have methods that are blocking and non-blocking in Node.js.

## Blocking:

It refers to the blocking of further operation until the current operation finishes. Blocking methods are executed synchronously. Synchronously means that the program is executed line by line. The program waits until the called function or the operation returns.

Example: Following example uses the readFileSync() function to read files and demonstrate Blocking in Node.js

**Index.js**

const fs = require('fs');

const filepath = 'text.txt';

// Reads a file in a synchronous and blocking way

const data = fs.readFileSync(filepath, {encoding: 'utf8'});

// Prints the content of file

console.log(data);

// This section calculates the sum of numbers from 1 to 10

let sum = 0;

for(let i=1; i<=10; i++){

sum = sum + i;

}

// Prints the sum

console.log('Sum: ', sum);

Run the index.js file using the following command:

node index.js

**Output:**

This is from text file.

Sum: 55

## Non-Blocking:

It refers to the program that does not block the execution of further operations. Non-Blocking methods are executed asynchronously. Asynchronously means that the program may not necessarily execute line by line. The program calls the function and move to the next operation and does not wait for it to return.

Example: Following example uses the readFile() function to read files and demonstrate Non-Blocking in Node.js

**Index.js**

const fs = require('fs');

const filepath = 'text.txt';

// Reads a file in a asynchronous and non-blocking way

fs.readFile(filepath, {encoding: 'utf8'}, (err, data) => {

// Prints the content of file

console.log(data);

});

// This section calculates the sum of numbers from 1 to 10

let sum = 0;

for(let i=1; i<=10; i++){

sum = sum + i;

}

// Prints the sum

console.log('Sum: ', sum);

Run the index.js file using the following command

node index.js

**Output:**

Sum: 55

This is from text file.

**Note:** In the non-blocking program the sum actually prints before the content of the file. This is because the program does not wait for the readFile() function to return and move to the next operation. And when the readFile() function returns it prints the content.

## Comparison between Blocking and Non-Blocking in Node.js:

Non-blocking execution is usually faster if we compare blocking and non-blocking code in node.js, allowing concurrency. It will enable the event loop to execute JavaScript callback functions after completing other operations.

Suppose, if a program takes 40ns to execute and 30ns of that work can be done asynchronously, that frees up 30ns per request handle in which the system can handle other requests.

It is not usually a good idea to mix blocking and non-blocking code in node.js:

const fs = require('fs');

fs.readFile('/uploads/sample.txt', (err, data) => {

if (err) throw err;

console.log(data);

});

fs.unlinkSync('/uploads/sample.txt');

In this code, ‘fs.unlinksync()’ is likely to run first which would delete the sample.txt file before it is read. To avoid this, code can be written in clear non-blocking way:

const fs = require('fs');

fs.readFile('/uploads/sample.txt', (readFileErr, data) => {

if (readFileErr) throw readFileErr;

console.log(data);

fs.unlink('/uploads/sample.txt', (unlinkErr) => {

if (unlinkErr) throw unlinkErr;

});

});

In this, fs.unlink() is within the callback of fs.callback(), the order of operations will be definite.

References

1. <https://www.ngdevelop.tech/angular/pipes/>
2. <https://angular.io/api/core/Pipe#description>
3. <https://www.code-sample.com/2018/05/angular-5-6-7-chaining-pipes.html>
4. <https://www.c-sharpcorner.com/UploadFile/ff2f08/routing-in-angularjs/>
5. <https://data-flair.training/blogs/angularjs-services/>
6. <https://www.c-sharpcorner.com/UploadFile/f0b2ed/what-is-http-service-in-angularjs815/>
7. <https://www.tutorialspoint.com/nodejs/nodejs_introduction.htm>
8. <https://www.geeksforgeeks.org/blocking-and-non-blocking-in-node-js/>
9. <https://www.geeksforgeeks.org/introduction-to-node-js/>
10. <https://www.geeksforgeeks.org/blocking-and-non-blocking-in-node-js/>
11. <https://www.codingninjas.com/codestudio/library/blocking-and-non-blocking-in-node-js>