How to Learn and Master Angular easily – Part1

quarta-feira, 6 de julho de 2016 16:22

How to Learn and Master Angular easily – Part1

The objective of "How to Learn and Master Angular easily – Part1" tutorial series is to easily master the basics of Angular and understand the foundation.

The initial inception of the Front End Java Script Frameworks have been almost over and we have few frameworks that most of the big corporations are evaluating for a full scale implementation. The challenge now is how to adapt this new concept and train mass resources to be more productive.

The industry went through very similar adaptation with the advent of Java or HTML. Now we are dealing with Angular, Ember, Backbone, React and many similar frameworks.

"Google" being there behind Angular JS has immensely helped it to become one of the most popular one. While there are many books, videos and tutorials are available on Angular JS many developers are still facing challenges to learn Angular in an efficient way.

I have seen developers facing challenges to grasp Angular JS. There are though few reasons:

1. Concept wise Angular is very different than JQuery, Dojo, ExtJS or similar library available in the market.
2. It's very easy to start with Angular on simple tasks however it gets complicated when you are trying to finish a complete production ready application in Angular. The learning curve is not smooth, means in order to get started you really need to read at least 80%-90% of a book or tutorial. Learning AngularJS is not like JQuery or ExtJS where you can read two chapters and get some coding completed.
3. Directives are complicated to understand for entry-intermediate level programmers.
4. In case you are not a hands-on person on HTML5, CSS3 and JavaScript and coming from a Spring MVC, JSF, Struts, ASP background you may face challenges on understanding Angular.

I don't want this to be just another tutorial for AngularJS. I want to address the challenges most of the developers are facing and help them learn Angular in a much faster and efficient way.

Before we get started, here are few important direction I am going to follow in this series, so that anyone can become familiar with Angular in a fast and efficient way.

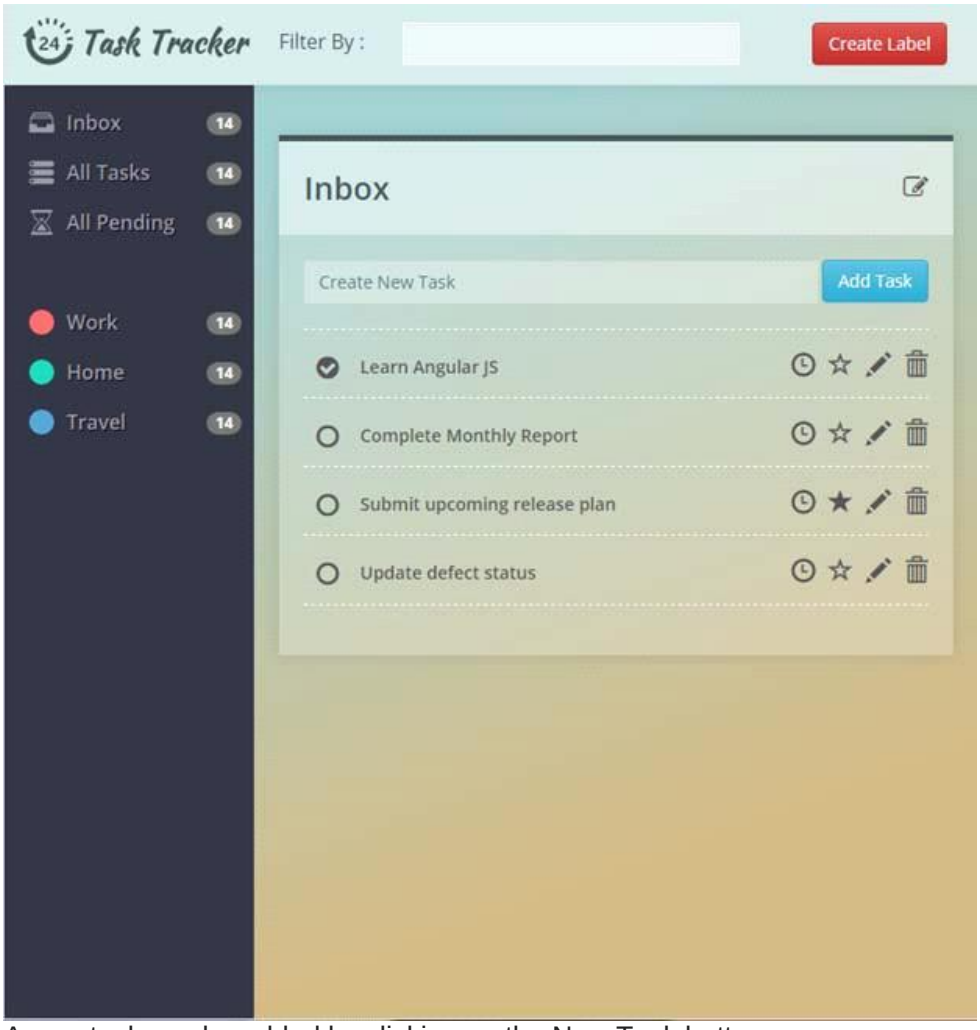
- Don't try to learn everything at once.
- I will not be talking about directive at all in this first series (find more on this below).
- Learn by example. Our objective will be to create an application and understand the concept of Angular.
- Understand the HTML on which we are going to add the Angular code.

I am not going to talk about how to create Angular `directive` in this series, since custom directives are the most complicated concept to understand and beginners shouldn't worry about it while starting.I have been working on real time production application for one of the largest corporation and based on my experience, directive is something not used in a daily basis, it should be learned once you have understood everything else about AngularJS. I will talk about directive, security, build tools etc in the next series.

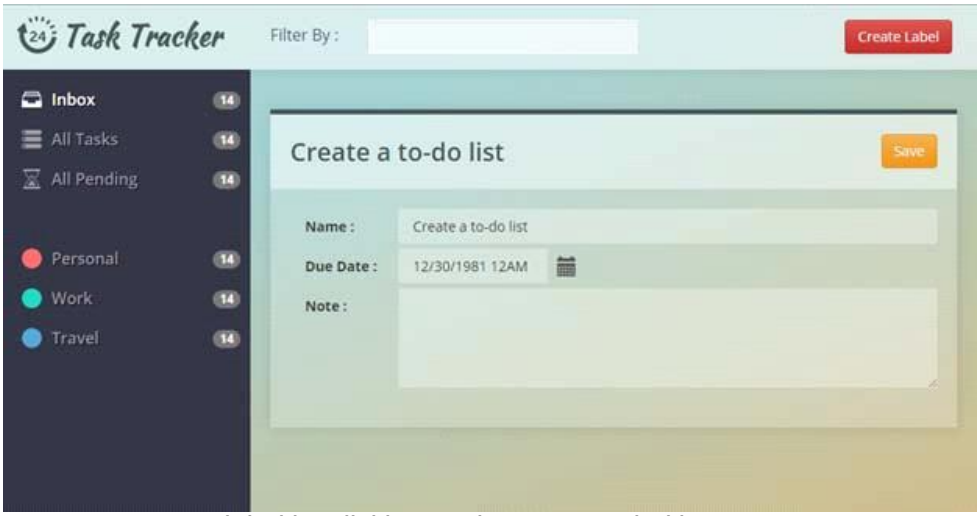
Learn by example is a very efficient way of mastering any topic; we will also follow the same. We will develop a TaskTracker application in Angular by end of the series.

In order to focus more on Angular JS, rather on HTML/CSS, I have already created the HTML & CSS required for this tutorial.I would recommend you to download the “chapter 1” from the github repository and follow along.

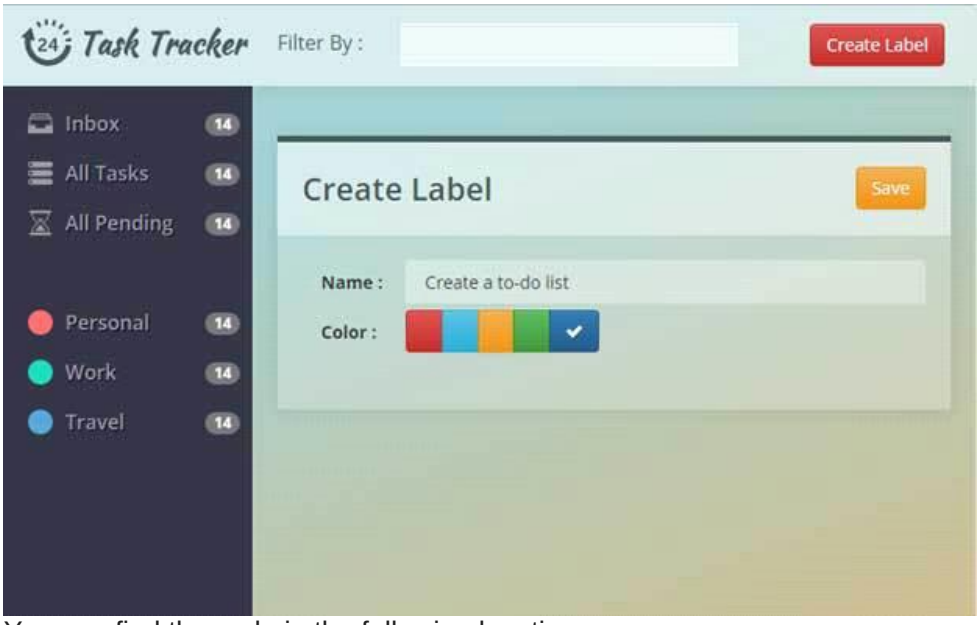
Here are the screen prints below. You will see the list of labels at the left side. The right side content area will display the list of task associated with the Label. It’s basically a ToDo Application.



A new task can be added by clicking on the New Task button.



You can create a label by clicking on the create Label button.



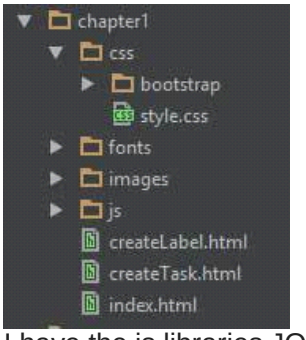
You can find the code in the following location:

https://github.com/adeveloperdiary/angular-for-web-developers/tree/master/chapter1

Note: The HTML is not fully responsive. I didn't want to spend too much time on that since responsiveness is not something we want to achieve here.

Project File Structure:

Here is the project file structure. We will be using bootstrap, however you don't need to know bootstrap.



I have the js libraries JQuery & Bootstrap inside the js folder. The style.css has all the custom css. Once you have downloaded the source, go ahead and open the index.html in Chrome.

Note: I will be using Chrome developer tools throughout this series.I suggest you the same.

Structure of the HTML:

As of now there are 3 HTML files – `index.html`, `createTask.html`,`createLabel.html`, all of them have the same structure. Each for one dynamic content. So even if now we have 3 static html files, we will end up having one base index.html and we will use angular to dynamically replace contents inside the `#page-body`.

NOTE: I have omitted some of the css classes here in the pseudo code below so that you can understand easily. Please refer the github repository for the actual content.

The `#wrapper` is the parent div which has two sections, the `#sidebar-wrapper` and `#page-content-wrapper`. The `#sidebar-wrapper` is for showing the left side menu and the `#page-content-wrapper` is for the right side content & header.

XHTML

```
1  <div id="wrapper">
2     <div id="sidebar-wrapper">
3     </div>
```

```
    <div id="page-content-wrapper">

    </div>
</div>
```

The `#sidebar-wrapper` has a `ul` to display the left side menu list. The header on the top of our Task Tracker application is not one div, the logo and name is actually the first `li` element of the `ul` element.

XHTML

```
<div id="sidebar-wrapper">
    <ul class="sidebar-nav">
        <li>Logo + App Name</li>

        <li>Inbox</li>
        <li>All Tasks</li>
    </ul>
</div>
```

The `#page-content-wrapper` has two sections, `#page-header` & `#page-body`. The `#page-header` has a bootstrap panel component to display the right section of the application header which includes `Filter` & `New Label`button.

XHTML

```
<div id="page-content-wrapper">
    <div class="panel" id="page-header">

        <div class="panel-body">
        </div>
    </div>
    <div id="page-body">
    </div>
</div>
```

The `#page-body` has another bootstrap panel which holds the dynamic contents like the Task Lists, New Task Creation etc. We have a custom panel header inside the `#panel-body`, we will not be using bootstrap panel header here.

XHTML

```
<div id="page-body">
  <div class="panel panel-default">

      <div class="panel-body">
          <div class="panel-top-header">

          </div>

          <!-- Panel Content Start -->

          <!-- Panel Content End -->

      </div>
    </div>
</div>
```

It is very important to understand the structure of the HTML since we will be making many changes on top of this.

Tips: I strongly recommend to use a proper IDE than notepad or eclipse. You can use `brackets.io`,this is open source and works on both windows and mac.

We will create a Single Page Application (SPA) using these 3 html templates in Angular. All the contents including the html templates will be loaded asynchronously. Once the index.html has been loaded, it will load Angular JS and other libraries, which will then bootstrap the application and load all required components asynchronously from the server whenever required.

You don’t have to name the initial page as index.html, you can name anything like app.jsp or app.asp.

In Part 2 we will start with Angular architecture and then change the HTML pages to make an application.

De <http://www.adeveloperdiary.com/angular-js/angular-1-x/the-best-way-to-master-angular-js-part-1/>

How to Learn and Master Angular easily – Part2

Welcome to “How to Learn and Master Angular easily – Part2” !

In this chapter, we will start with understanding Angular Architecture, then move on to individual components such as `Service`, `Controller` & `View`. We will work on live demo and full working code base. We will also learn about few important concepts like `Dependency Injection` and `Two Way Binding`.

In Part 1 of the “The Best Way to Master Angular JS” we read about the background, objective etc and then started with the HTML template of the TaskTracker application.

Why Angular JS ?

Question: With the help of JQuery & vanilla JavaScript we can easily develop an application, so why do we even need Angular or any other such framework?

Here are few good points to be noted, I am not going to have the full list here since many of it may not make sense to you at this moment.

1. Maintainability : Angular helps to structure the code so that the maintainability becomes very easy. This is the most important point you need to keep in mind when you start your development.
2. MVC : Angular supports the MVC framework at the front end. This makes our life much easier.
3. Reusability : Most of the Angular JS components are reusable. Anything you develop,even the views are fully reusable
4. Rapid Development : Angular itself is performing many heavy lifting and since everything is reusable in Angular, the development timeframe can be shorten significantly by experienced Angular developers.

There are many other functional advantages, we will discuss about them as you follow along.

We will talk about the architecture of Angular JS in a moment, but lets understand few important points on modern web development. You may know that with Angular we will always develop a Single Page Application (SPA) and we don’t have the refresh the entire application. So what else has been changed since the edge of Struts,JSF,Spring MVC?

1. Better Standard & Fast Browser : The browser is getting more and more powerful everyday. Before Chrome & Firefox became popular we used to develop web applications in IE4, IE5. At that time we had to render the entire page at server end. However now the ECMA5 & ECMA6 standards are powerful enough to do the heavy lifting.
2. MVC : We all learnt the popular MVC framework at some point and most of us used that at the server end, since the server used to render all the html content. However the modern Java Script Libraries like Angular supports the MVC framework which made the server side development of the presentation layer light, easy and service oriented.

3. JSON : JSON (Java Script Object Notation) is another very powerful element. JavaScript now can read/parse the JSON very easily, the communication with the server has become the most simple task.
4. Debugging Tool : Earlier the browser didn't have any development tool, with Fire Bug and Chrome Development tools it has become very easy to debug the Java Script.

Angular Architecture:

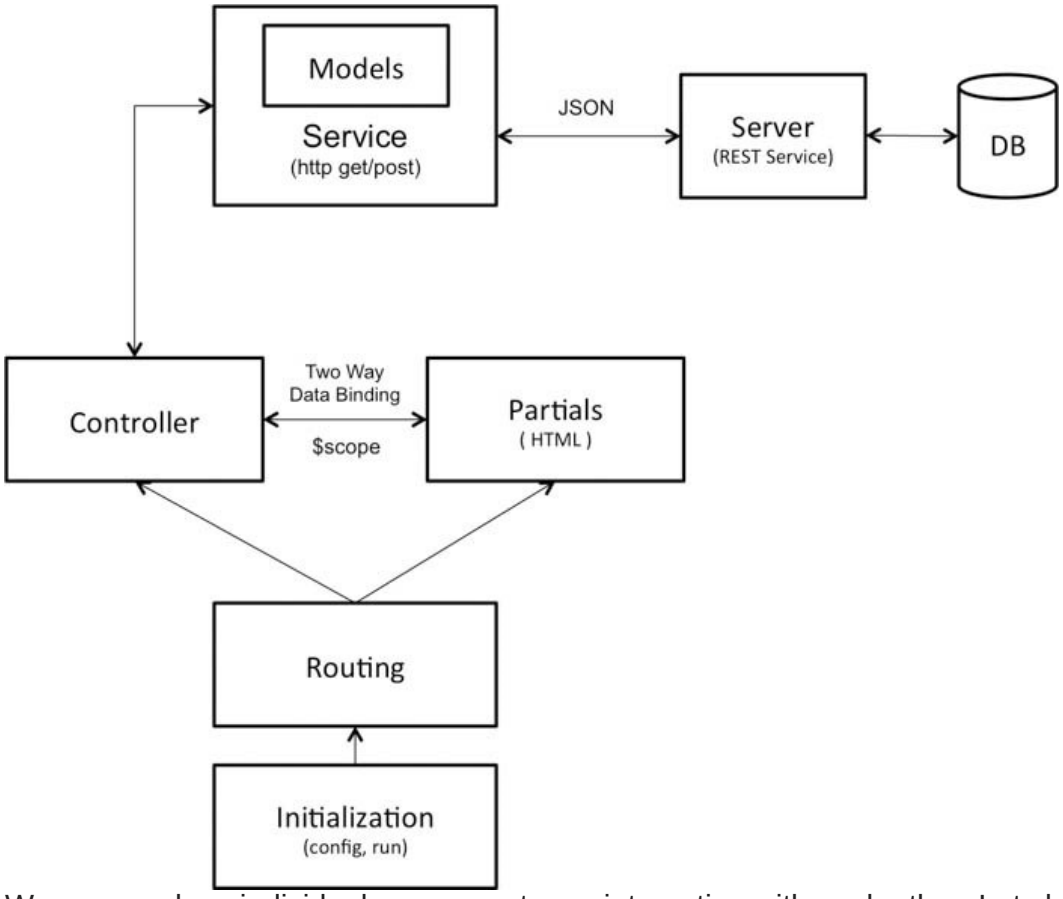
One thing I want to make sure is not to explain everything at once, the below picture has components which you need to understand now and not everything.

Angular has few integral parts which we need to understand to get started.

1. Service
2. Controller
3. View (Partials)
4. Routing

Again, the above list does not have everything, however sufficient for our purpose for time being.

Here is the high level architecture diagram of Angular JS 1.x, based on above components.



We can see how individual components are interacting with each other. Lets learn more about them. We will take the top down approach.

Service:

We need to get/push the DATA loaded from/to the server. The Angular Service is meant for that. The only important characteristics to be mentioned about `Service` is its persistent behavior. The model objects inside the service are static in nature so once the data is loaded into the model you can access it from anywhere. In nutshell, service acts like a data store and its encapsulates the server side interaction.

What to have in service?

1. HTTP POST/GET methods
2. Model Objects

Now, Lets see an example on how to create a service in angular.

angular-service.js

JavaScript

```
var module = angular.module('TaskTracker',[]);

module.service('TaskService',function($http){

    var tasks=null;

    this.getTasks=function(){
      if(tasks!=null){
        return tasks;
      }else{
        $http.get('data/data.json')
          .success(function(response){
            tasks=response;
            return tasks;
          })
          .error(function(data){
            return "Error";
          });
      }
    };
});
```

At the first line we have created a module then added the service to the module. We will learn about module in the next chapter.

We can create a Service just by calling the `service()` function of any module. The first parameter is the name of the Service and 2nd parameter is a function.

The name of the Service is `TaskService`.In order to expose any method as part of the service we can use `this` keyword and add the function to this.Then that function will be available to be executed from outside of the service.

The `getTasks()` method is exposed to fetch the list of tasks from a json file. If the tasks variable is null then service will fetch the data from backend.

`$http` object was used to make the http get service to call backend.The $http service is a core Angular service that facilitates communication with the remote HTTP servers via the browser's XMLHttpRequest object or via JSONP. If the remote call is successful then we will store that in our `tasks` variable otherwise it will just return "Error".

Remember, anything starts with a `$` is a native angular object/service.

In order to invoke the `getTasks()` method from the `TaskService` service you need to execute following line.

JavaScript

```
var tasks=TaskService.getTasks();
```

It is that simple! The variable tasks is the model object which is going to hold the data.

Just that you know, there are 4 ways to create a Service in Angular. Factory is another popular one. Essentially all of them kind of do the same thing.

In MVC the M=Model is the service here.

Dependency Injection

Notice, while we are calling the `TaskService.getTasks()` we are not passing the object named `$http` into the method `getTasks()`.

Question: So how the service can get the instance of $http service?

Angular can automatically detect a dependent Angular Component (Service, Module, Controller etc) and inject it automatically to the specific Component. This is called `Dependency Injection`.

If you need to pass any specific data like id or name you still need to pass that as part of the getTasks() method. If you are confused don't worry, we have many more examples to come in order to understand this.

Controller:

The Controller is responsible to interact with the Service (Model) and the View (Partials/HTML). In simple words, Controller makes service call to get/push data and passes that data to the HTML in order to render the view. You can refer the diagram above to get a clear view.

So Controller in Angular JS is mainly responsible for:

1. Make Service Calls
2. Interact with the View
3. Initiate Routing Change

You have already seen how we can make a service call from Controller in the Service section. Lets get that code here and see how to create an Angular Controller.

JavaScript

```
1  var module = angular.module('TaskTracker',[]);
2
3  module.controller('TaskController',function(TaskService){
4
5
6    var tasks=getTaskList.getTasks();
7
   });
```

I have given `TaskService` as the parameter in the `TaskController`, since the TaskService is an Angular Component, it will be automatically injected in the Controller (Dependency Injection).

So now you know how to invoke a service from a controller. Next we will learn how to interact with the View (HTML) from the Controller.

In order to share the variables between Controller & Partials (HTML) Angular JS has a default variable named `$scope`.

`$scope` is like a global Object which is accessible to the View (Partials). Anything you put in `$scope` will be accessible to the View. If you are coming to JEE background, `$scope` is exactly same as the request object we use in our JSP. So from Servlet/ActionClass we had to load everything into the request object

so that the data can be accessible from the JSP.

In our previous example, we have stored the task list in a local variable called tasks in the Controller, however var tasks wont be available to the View. So we need to store it in the `$scope` variable. Lets jump into the code.

JavaScript

```
module.controller('TaskController',function($scope,TaskService){


  $scope.tasks=getTaskList.getTasks();

});
```

We had to add `$scope` as the function argument so that Angular can inject it in our `TaskController`.

Now the tasks variable will be automatically available to the View. Similar way any function call from view (HTML) needs to be implemented in the Controller as well. We will see an example of that in the View section.

View :

You might have noticed, I am using the name View, Partials, HTML interchangeably since all of them carries almost same meaning. If you have worked with handlebars or mustache then you can correlate Partials with the HTML template. In case you are from the Java world, the View (Partial) is nothing but the JSP where we manipulate the HTML with the help of scriptlet.

Lets see an example of the View.

Note: All the examples here in this chapter are on view, since partials generally stays in another file and in runtime they get added dynamically. (Something like a template we use in JQuery) Partials are not really different than a simple view, so lets understand View here and then in later chapters will learn more on partials.

XHTML

```
<head>
  <script>
    var module = angular.module('TaskTracker',[]);

    module.controller('MyController',function($scope){


      $scope.message="Hello World";

    });
  </script>
</head>
<body ng-app="TaskTracker">
  <div ng-controller="MyController">
    {{message}}
  <div>
</body>
```

In the example above, I have added message to the `$scope` variable which will be available to the view. `{{}}` is the syntax for the template you can use to access the data from the `$scope` variable. So `{{message}}` will be replaced by Hello World.See the demo below:

`ng-controller` is an angular element where we need to specify the name of the Controller for this view.Since there will be many Controllers, the View needs the Controller name to bind the `$scope` object.I have assigned `MyController`as `ng-controller`, so any code inside the `div` will reference the `$scope`variable of the `MyController` Controller.

NOTE: Any Angular specific element like `ng-controller` is called as`Directive`.We can create our own custom directive.

Two Way Binding :

The `$scope` will be automatically updated from both Controller & View, means if you change any value of `$scope` in the Controller it will be automatically be updated in the View and vice versa. This is called `Two Way Binding`, another very important function of Angular. Lets see how it actually works.

XHTML

```
<head>
  <script>
  var module = angular.module('TaskTracker',[]);

  module.controller('MyController',function($scope){


    $scope.message="Hello World";

  });
  </script>
</head>
<body ng-app="TaskTracker">
  <div ng-controller="MyController">
    <input type="text" ng-model="message">
    <label>{{message}}</label>
  <div>
</body>
```

The `ng-model` is another directive through which you can bind an HTML element to its $scope variable.

Here in the above example we have assigned the `message` to the `ng-model`. So whenever we change the text it will automatically update the`$scope.message`.

Since our `label` is displaying the `$scope.message` you can see the updated value as it changes.

Initially the Text and the Label both will display Hello World, once you change the text the label will automatically be updated. Check out the demo below.

Two Way Binding is a very important functionality, see how we can update data without any onChange() function or onClick() function. It has its own drawback, however most of the cases its very useful.

Going back to View, now lets see how to invoke a Controller function from View.

XHTML

```
<head>
  <script>
  var module = angular.module('TaskTracker',[]);

  module.controller('MyController',function($scope){

```

```
    $scope.name="World";
    $scope.message="";

    $scope.showMessage=function(){
      $scope.message="Hello "+$scope.name + " !";
    };

  });
  </script>
</head>
<body ng-app="TaskTracker">
  <div ng-controller="MyController">
    <input type="text" ng-model="name">
    <input type="button" value="Show Message" ng-click="showMessage()"/>

    <label>{{message}}</label>
  <div>
</body>
```

`ng-click` is a directive which calls a function when the attached element has been clicked. Here we have attached a function name `showMessage()` with the $scope of the `MyController`, then called it using `ng-click()` from our view.

We are updating the `$scope.message` from showMessage() which will automatically be displayed in the `label` (Two Way Binding)

`ng-change`, `ng-dblclick`, `ng-keyup` are also similar directives like `ng-click`. They also take an expression or function with arguments.

Here is the live demo:

I will discuss about routing later, once we get to know more about the service, controller and partials. In next chapter we will learn about Angular module, Project Folder Structure etc.

How to Learn and Master Angular easily – Part3

Welcome to the “How to Learn and Master Angular easily – Part3” !

In chapter 3 we will learn on how to create Modules in Angular js. Then we will start developing our TaskTracker application and learn few Angular native directive such as ng-repeat, ng-class etc. We will slowly build the foundation that is required. It is very important to start slowly and understand each concept.

In Part2 we have learned about Angular Architecture, Service , Controller, View, Two Way Binding etc.

Module:

Modules are like plugins, where all of the related logics are placed together. Module helps if you are creating a common functionality which can be used across multiple application, then you can add the common module you created as a dependency to your app.You will create at least one Module for any application, in Part 2 the code snippets had the module created.

Question: You might have many modules created, but how to tell angular which module is the parent application? or using different words, how to bootstrap an Angular JS Application ?

Answer: There is a directive named `ng-app` where you specify the module name of your application.

Here is the code snippet from Chapter 2. Notice the module name`TaskTracker` was given in `ng-app`.

test

XHTML

```
<head>
  <script>
    var module = angular.module('TaskTracker',[]);

    module.controller('MyController',function($scope){


      $scope.message="Hello World";

    });
  </script>
</head>
<body ng-app="TaskTracker">
  <div ng-controller="MyController">
    {{message}}
  <div>
</body>
```

Now lets see how to add a custom module as a dependency to the application and use it.

Module Dependency:

Here is an example of how we can create custom module and use them across the application using `Dependency Injection`. In case you had noticed the `[]`in the second argument when we were defining the module name, this is where we need to specify the dependent module name.

In this example we created a custom module name `MyModule` and declared that as a dependency in our application named `TaskTracker`. Then we just had to mention the name of the service in our controller in order to call the`callSomething()` method.

XHTML

```
<head>
  <script>
    var myModule = angular.module('MyModule',[]);

      myModule.service('MyService',function(){
        this.callSomething=function(){
          return "Hello World !";
        };
      });

      var module = angular.module('TaskTracker',['MyModule']);


      module.controller('MyController',function($scope,MyService){

        $scope.message=MyService.callSomething();
      });

  </script>
</head>
<body ng-app="TaskTracker">
  <div ng-controller="MyController">
    {{message}}
  <div>
```

```
</body>
```

Here is the live example:

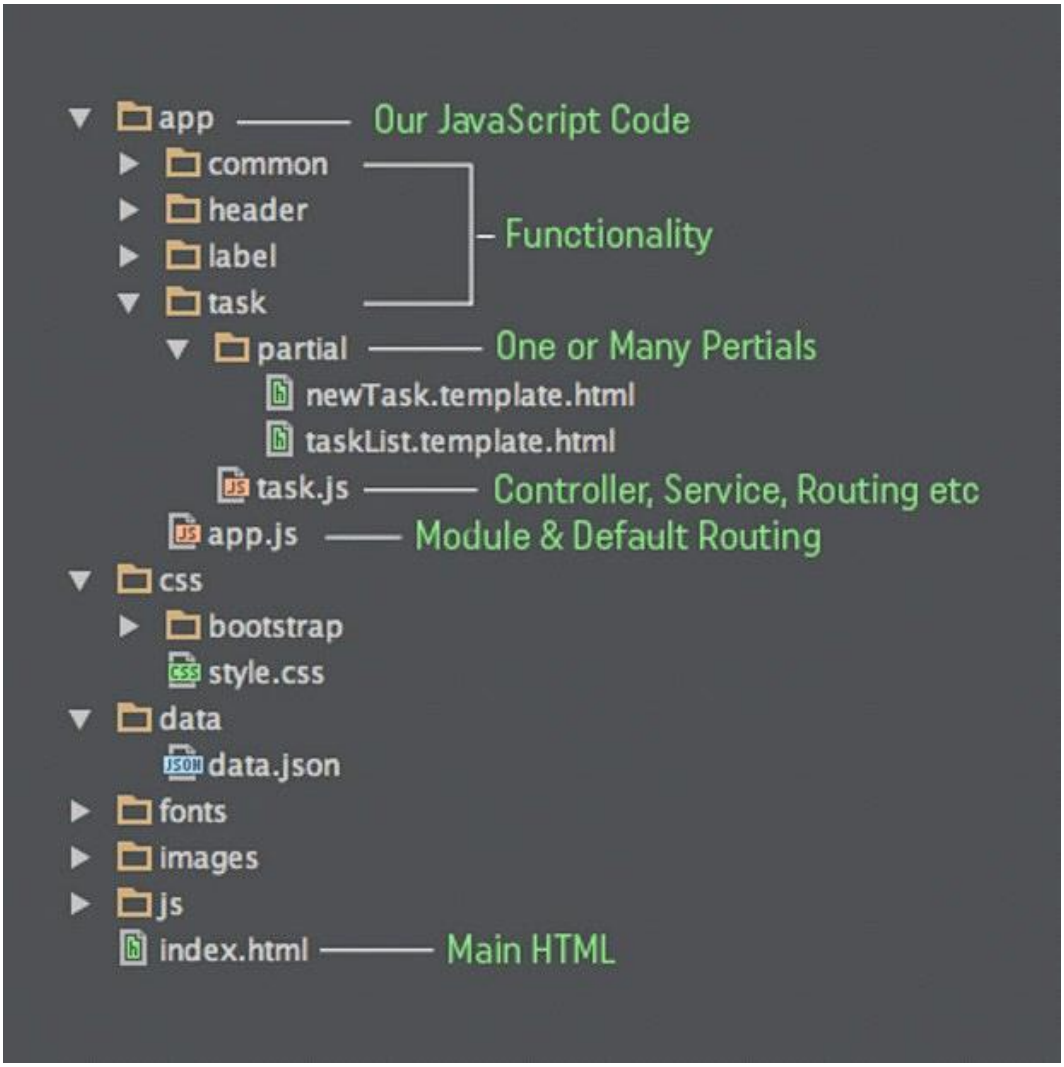
This is a very simple example, in real life mostly reusable directives are used as module, like UI Grid, UI Router , Bootstrap UI etc. You can think modules are just like .DLL or .JAR or .SO files which can be easily plugged into any application.

Note : The `ng-app` can be defined in `html` element also rather than in the`body` element.

Folder Structure :

Its very important to maintain a proper folder structure in Angular for any enterprise project since at runtime all of them belogs to the same context and it will be very difficult to maintain.

In our TaskTracker Application we will follow the following structure.



Inside the app folder we have the app.js and then different folders for each functionally.Then each of them will have a folder named partials for the partials(view/template) and one js file for all the necessary angular component such as Controller, Service, Routing, Filter etc.

In case you are working on a very complex project you can create one js file for each Angular component like task.js, task-controller.js, task-service.js, task-directive.js etc.

Lets make changes to the TaskTracker application. Download the chapter 1 html code from github from the following link and start from there.

https://github.com/adeveloperdiary/angular-for-web-developers

Add the `angular_1.4.js` before closing the head element after the other libraries. The angular 1.4 library is already included added in the chapter 1 js folder.

Add the `ng-app="taskTracker"` in the body element in `index.html`. This will bootstrap the application.

index.html

XHTML

```
<body ng-app="taskTracker">


....

</body>
```

Now, add a folder name app and create a file named `app.js`. Create an Angular module name `TaskTracker`;

app.js

JavaScript

```
var module=angular.module('taskTracker',[]);
```

Add the `app/app.js` at the end of the body element.

Now run the index.html in Chrome. If you are using bracket.io or webstrom then it will automatically have the web server. Otherwise you need a webserver or appserver to be installed.

Open the developer tools are make sure you are not seeing any error in the console.

Let me know in case you face issue with the setup and I will help you.

Now create a controller in the `app.js` named `labelController` and add the label data in the `$scope`.

JavaScript

```
var module=angular.module('taskTracker',[]);

module.controller('labelController',function($scope){

  $scope.data={
    "labels": [
      {
        "name": "Work",
        "color": "color_red"
      },
      {
        "name": "Home",
        "color": "color_green"
      },
      {
        "name": "Personal",
        "color": "color_blue"
      }
    ]};
```

```
});
```

Open the index.html, now we will update the `sidebar-nav` `ul`. The first `li` is the application name and next three of them are the fixed label named Inbox, All Tasks and All Pending.

In our TaskTracker application user can add any custom label, so lets add them dynamically from the JSON object we created in the controller.

JavaScript

```
<div id="sidebar-wrapper" ng-controller="labelController">
  <ul class="sidebar-nav">
      <li class="sidebar-brand"><a href="#">Task Tracker</a></li>
      <li><a href="#" class="inbox_icon">Inbox</a><div class="todo-badge">14</div></li>
      <li><a href="#"  class="all_icon">All Tasks</a><div class="todo-badge">14</div></li>
      <li><a href="#"  class="pending_icon">All Pending</a><div class="todo-badge">14</div>
</li>
      <li><br></li>

      <li ng-repeat="label in data.labels">
        <div class="circle" ng-class="label.color"></div>
        <a href="#">{{label.name}}</a>
        <div class="todo-badge">14</div>
      </li>
  </ul>
</div>
```

Here the ng-controller was added to attach with the View as learned in Part/Chapter 2. There are few new directives were added in the code, before we go though them, lets reload the index.html and you should be able to see the custom Labels added.

ng-repeat :

`ng-repeat` is the for loop in angular which takes an array and loops through it. Here the `data.labels` is the array as we have defined in the labelController's $scope object. The label is the local variable for each of the object.

Lets see few more example of this to understand fully.

XHTML

```
<head>
  <script>
  var module = angular.module('MyApp',[]);
  module.controller('MyController',function($scope){

    $scope.list=[{
      name:"John Doe",
      age: 25
    },{
      name:"Sean Lee",
      age: 22
    },{
      name:"Mary Evans",
      age:23
    }];
  });
  </script>
</head>

```

```
<body ng-app="MyApp">
  <div ng-controller="MyController">
    <ul style="list-style:none">
      <li ng-repeat="person in list">
        [ {{$index+1}} ] Name :  {{person.name}}
        , Age : {{person.age}}
      </li>
    </ul>
  <div>
</body>
```

list is an array of persons with name & age. It has been assigned to the `$scope`in the controller.

Now we are iterating through the array using `person in list` where each person represents one person object.

Then we are accessing the name and age using `person.name` and`person.age`. We are doing one more thing extra here, the `$index` represent the index of the element in the given array. We are accessing it and showing it in the list. We are not using `$index` in our TaskTracker application, however it can be useful in many scenarios.

Here is the live demo.

ng-class:

This is another directive to specify a css class name for any element. In our TaskTracker application we are dynamically setting the color of the custom Labels using `ng-class`. The `label.color` has the css class name for the different color.

Notice, we didn't have to use the `{{}}` in the `ng-class` or `ng-repeat`directive. Any angular directive (Native or Custom) can take an expression and the expression does not need any the curly braces since they are already Angular component.We will see more example of expression later.

Now lets load the list of tasks. Here is the JSON for tasks.

```
{"tasks":[
    {
        "id":1,
        "name":"Send Status Report",

        "dueDate":"12/30/1981",
        "note":"",
        "completed":false,
        "labelName":"Inbox"
    },
    {
        "id":2,
        "name":"Learn Angular JS 1.x",

        "dueDate":"12/30/1981",
        "note":"",
        "completed":false,
        "labelName":"Work"
    },
    {
        "id":3,
```

```
        "name":"Purchase Grocery",

        "dueDate":"12/30/1981",
        "note":"",
        "completed":true,
        "labelName":"Home"
    }
]}
```

Each task has an id, name, completed status and labelName. Rest of the elements we will not be implementing in this series,this is something that you should complete integrating by yourself at the end of the series.

Lets define this in our controller and name it as `data1` then assign it to the`$scope` object. This should look like `$scope.data1={ "tasks": [ ... ] }`

Now open the `index.html` and make following changes.

In the `list-group` `ul` element, keep only the first `li` element and delete rest of them.

Add the following code.

app.js

XHTML

```
<ul class="list-group">
    <li class="list-group-item" ng-repeat="task in data1.tasks">
        <input id="{{task.id}}" type="checkbox" checked>
        <label for="{{task.id}}" class='checkbox-label'>{{task.name}}</label>
        <span class="glyphicon glyphicon-trash pull-right action-icon"  aria-hidden="true"></span>

        <span class="glyphicon glyphicon-pencil pull-right action-icon"  aria-hidden="true">
</span>
        <span class="glyphicon glyphicon-star-empty pull-right action-icon"  aria-hidden="true">
</span>
        <span class="glyphicon glyphicon-time pull-right action-icon"  aria-hidden="true"></span>

    </li>
</ul>
```

We are looping through the `data1.tasks` array and populating the task name in the label element. Open & test the `index.html` in Chrome.

Note : We need to use `{{}}` in the id attribute of the input element since id is not an angular related and it’s a default html attribute.

You will not see any tasks getting displayed !!! There are also no error in the console. So whats wrong here ?

As we learnt in Part 2, every view needs to be tied to at least one controller so that the view can access the `$scope object` of the controller. If you look closely, the `labelController` we added in the `#sidebar-wrapper` `div` is already closed before `#page-content-wrapper` `div`. So `labelController`will not be accessible by our code written outside of the `#sidebar-wrapperdiv` element.

Now we shall create a new controller named `taskListController` and define it to the `#page-body` `div` element. Lets do that in our `app.js` file. Move the`data1` in this `taskListController`.

JavaScript

```
module.controller('taskListController',function($scope){

    $scope.data1={
        "tasks":[
            {...

        ]};
});
```

Add the `taskListController` to the `#page-body` `div` element.

XHTML

```
<div class="padding-20" id="page-body"
  ng-controller="taskListController"> .. </div>
```

Now if you refresh the index.html page you can see all 3 tasks getting displayed there.

Before we finish this part, lets clean up some of the code.

If we keep adding controllers in the `app.js` then very soon the app.js will be very big and difficult to maintain. Lets create few more folders.

- Create two folders named `label` and `task` inside our `app` folder.
- Create a js file named `label.js` inside the label folder and move the`labelController` inside this file.
- Create a js file named `task.js` inside the label folder and move the`taskListController` inside this file.
- Add the new js files after the app.js in the index.html.

Here are the files:

JavaScript

```
var module=angular.module('taskTracker');

module.controller('labelController',function($scope){

    $scope.data={"labels": [ ... ]};
});
```

JavaScript

```
var module=angular.module('taskTracker');

module.controller('taskListController',function($scope){

    $scope.data1={"tasks":[ ... ]};
});
```

JavaScript

```
var module=angular.module('taskTracker',[]);
```

XHTML

```
...
</div>
<script src="app/app.js"></script>
<script src="app/label/label.js"></script>

<script src="app/task/task.js"></script>
```

```
7
```
```
</body>
</html>
```

If we don’t have the 2nd argument while creating a module then Angular will not create a new module, rather it will assume that the module already exists and it will add the new components to the existing module. In `label.js` and`task.js` we didn’t define the 2nd argument in the `angular.module()`function so Angular will just add the controllers to the taskTracker module we created in our app.js.

The code should work as it already was. This will be the end of Part 3.

How to Learn and Master Angular easily – Part4

We will learn some of the very important concepts in How to Learn and Master Angular easily – Part4.

We will start with updating our TaskTracker Application, then we will learn and implement Promises, Expression and then complete part of the application using the knowledge we have gained so far.

In Part3 we have learned about Modules, Angular Folder Structure, more native directives etc. We will also take one more step towards completing our TaskTracker application.

We have the JSON data embedded in the Controllers. Lets move that to a Service and fetch the `data/data.json` using http GET. We will have only one service to fetch the data at once from the json file. Our Labels and Tasks will be inside single JSON file. Create a folder named common inside app and create a file named `common-service.js`.

common-service.js

JavaScript

```
var module=angular.module('taskTracker');


module.service('dataService',function($http) {


    var arrLabels = [];
    var arrAllTasks = [];

    this.loadTaskData = function () {

        $http.get('data/data.json')
            .success(function (data) {

                arrLabels = data.labels;
                arrAllTasks = data.tasks;

            })
            .error(function (data) {
                console.log('Error : '+data);
            });
    };
});
```

First we are adding `$http` in the argument of the `dataService` service so that it will be injected by Angular. Then we have the `loadTaskData()` function where we are loading the `data/data.json` file and upon success we are assigning the label data to `arrLabels` and tasks data to `arrAllTasks` array. In case of error we are just logging the error in the console.

Now all we need to do is call `dataService.loadTaskData()` from our controller to load the data to the arrays. Lets also write the getter function for accessing the arrays.

common-service.js

JavaScript

```
this.getAllTasks=function(){

  return arrAllTasks;
};

this.getAllLabels=function(){

    return arrLabels;
};
```

Once the service `dataService.loadTaskData()` has been invoked we can then call either `getAllTasks()` or `getAllLabels()` in order to get the data. Our Controller code will look like:

JavaScript

```
var module=angular.module('taskTracker');

module.controller('labelController',function($scope,dataService){

    dataService.loadTaskData();

    $scope.data=dataService.getAllLabels();
});
```

Eveything looks good so far, however, there is a small problem. The`$http.get()` as an asynchronous method call so, there will be a delay to load the data from backend and when the `dataService.getAllLabels()` gets called from the controller the data may not been loaded. As a result you will get an empty array when you call `dataService.getAllLabels()`.

We need to find a way to wait till the `$http.get()` asynchronous call has been completed and then call `dataService.getAllLabels()` to get the data.

We are going to use Angular `$q` service.

Promises ($q):

"`$q` is a service that helps you run functions asynchronously, and use their return values (or exceptions) when they are done processing". There are two API as part of $q, deferred and promise.

`Promise` will be mostly be used along with `$http` service. Promise is a generic design pattern and `$q` used in our code just an implementation that Angular provides.

1. Once you get the `defer` object by calling `$q.defer()`.
2. Return the `promise` object by calling `defer.promise`.
3. Mean while after the external/asynchronous call has completed (`$http.get()` in our case), use either the `defer.resolve()` or`defer.reject()` to send the data/error back.

See the example below, this is the signature`function().then(success,error)`. Once the external call has been completed, the success or error function will be called based on`defer.resolve()` or `defer.reject()`

JavaScript

```
function getData(){
  var defer=$q.defer();
  var promise=$http.get('some url')
    .success(function(data){
defer.resolve(data);
})
.error(function(data){
defer.reject("Error");
});
  return defer.promise;
}

getData().then(function(data){
//This will be called in case of defer.resolve

console.log(data);
},function(error){
//This will be called in case of defer.reject

console.log(error);
});
```

Here is a full working example of a `$http` call with and with-out the Promises.

XHTML

```
<head>
  <style>
    .green{
      color:green;
    }

    .red{
        color:red;
    }

    div {
        padding:10px 0px 0px 30px;
    }
  </style>
  <script>
    var module = angular.module('MyApp',[]);
    module.service("MyService1",function($http){

    var arrData=[];

    this.loadData = function () {
        $http.get('https://api.github.com/users/adeveloperdiary/repos')
          .success(function (data) {
            arrData=data;
          })
          .error(function (data) {
            console.log('Error : '+data);
          });
      };

    this.getData=function(){
      return arrData;
    };
```

```
34
35      });
36      module.service("MyService2",function($http,$q){
37
38        var arrData=[];
39
40        this.loadData = function () {
41            var defered=$q.defer();
42            var promise=$http.get('https://api.github.com/users/adeveloperdiary/repos')
43
44              .success(function (data) {
45                defered.resolve(data);
46                arrData=data;
47              })
48              .error(function (data) {
49                console.log("Error");
50                defered.reject("Error");
51              });
52
53              return defered.promise;
54
55          };
56
57        this.getData=function(){
58          return arrData;
59        };
60
61      });
62
63      module.controller('MyController',function($scope,MyService1,MyService2){
64
65
66        MyService1.loadData();
67        $scope.data1=MyService1.getData();
68
69        MyService2.loadData()
70       .then(function(data){
71          $scope.data2=MyService2.getData();
72       },function(error){
73          console.log(error);
74       });
75      });
76  </script>
77  </head>
78  <body ng-app="MyApp" ng-controller="MyController">
79    <div>
80      Length of the Request 1:
81      <label ng-class="data1.length>0 ? 'green' : 'red'">
82  {{data1.length}}
83  </label>
84    </div>
85     <div>
86         Length of the Request 2:
87      <label ng-class="data2.length>0 ? 'green' : 'red'">
88  {{data2.length}}
89  </label>
    </div>
  </body>
```

In the example above we are having two services. You can see the live demo below. We are fetching the

the repository details for adeveloperdiary from github. Here is the URL : https://api.github.com/users/adeveloperdiary/repos. This returns a JSON file and we are going to print the number of repositories available.

In the first service `MyService1`, we have not used the promise. So the by the time we call `MyService1.getData()` the github service hasn't returned the response. As a result the data will be empty when we print the size of the repository. However in `MyService2`, are returning the Promise and upon receiving the response the function passed in the `then()` method from the`MyController` will be invoked. This is instance we shall re the appropriate size of the repository and not zero.

Expression:

In the previous chapter we spoke about expressions. In the example above you can see an example of that expression. We are setting the color of the label based on the array size. Pretty cool huh !

Going back to our TaskTracker Application. Here is the revised version of our`loadTaskData()` service using Promises.

common-service.js

JavaScript

```
var module=angular.module('taskTracker');

module.service('dataService',function($http,$q) {


    var arrLabels = [];
    var arrAllTasks = [];

    this.loadTaskData = function () {

        var defer=$q.defer();
        $http.get('data/data.json')
            .success(function (data) {
                defer.$$resolve(data);
                arrLabels = data.labels;
                arrAllTasks = data.tasks;
            })
            .error(function (data) {
                defer.reject("Error");
            });
        return defer.promise;
    };
});
```

Now we shall call the service from our Controllers.

Once the data has been loaded we need to populate the `$scope.data` in our`labelController`. We need a getter method in the to get the `arrLabels` from the `dataService`.

Lets write that.

common-service.js

JavaScript

```
this.getAllLabels=function(){

```

```
return arrLabels;
};
```

Now lets complete the `labelController`.

label.js

JavaScript

```
module.controller('labelController',function($scope,dataService){

    dataService.loadTaskData()
        .then(function(data){
            $scope.data=dataService.getAllLabels();
        },function(error){
            console.log(error);
        });
});
```

We need a similar getter method for getting all the tasks also.

common-service.js

JavaScript

```
this.getAllTasks=function(){

return arrAllTasks;
};
```

Here is the updated `taskListController`.

task.js

JavaScript

```
module.controller('taskListController',function($scope,dataService){

    $scope.data1=dataService.getAllTasks();
});
```

Now add the `app/common/common-service.js` after the app.js in index.html.

We need to make one more small change to make our code working, in index.html we are iterating through `data.labels`, however since we are already assigning the labels to $scope we need to directly iterate the $scope.data. So change the `data.labels` to just `data` in `index.html`.

Need to make same change for the task array too.

index.html

XHTML

```
<li ng-repeat="label in data">
...
</li>

...
<li class="list-group-item" ng-repeat="task in data1">

...
</li>
```

Save the file, go ahead and open the index.html in the browser. The labels should be working fine however the tasks list won't load, since we are getting all the Tasks before even the data.json has been loaded. We will fix it later.We should load the labels first and then only load the tasks, so we need some

way to load the task controller from the label controller. Right now both of the controllers are getting loaded at the same time.We will fix this when we learn about routing.

Next we need to display tasks count at the right side of each labels. Let's add the required functions in the `dataService`.

For all the custom labels, we need to get the list of tasks by the label name. Each tasks already has a label name in the json file. So we need a function which will take the label name and return all the tasks associated with that label.

common-service.js

JavaScript

```
this.getTasksForLabel=function(label){
var tasks=[];

angular.forEach(arrAllTasks,function(obj,key){

if(obj.labelName===label){
tasks.push(obj);
}
});

return tasks;
};
```

angular.forEach() : Invokes the iterator function once for each item in obj collection, which can be either an object or an array. This is basically a for loop on an Array or Object.

If the label name matches we are adding the label object to a local tasks array variable. At the end returning the tasks array.

Note : Probably in real application we will make another backend server call here and get the list of tasks for any label from DataBase itself. However since we are just learning Angular here, we will have our service itself to calculate and return the data.

We need to show all the pending tasks under "All Pending" label. So would need another function in the service to just do that.

common-service.js

JavaScript

```
this.getTaskByCompletionStatus=function(status){

        var tasks=[];
        angular.forEach(arrAllTasks, function(task, key) {

            if(task.completed===status){
                tasks.push(task);
            }
        });

        return tasks;
    };
```

Here we are verifying the `completed` value to the status (true or false).

The div with `class="todo-badge"` displays the count of the tasks. We will call a function named `getTasksLengthForLabel()` from View to load the counts. We need to pass the name of the label here.

index.html

XHTML

```
<li ng-repeat="label in data">
<div class="circle" ng-class="label.color"></div>
<a ng-click="setLabel(label.name)">{{label.name}}</a>
<div class="todo-badge">{{getTasksLengthForLabel(label.name)}}</div>

</li>
```

Now let's create the `getTasksLengthForLabel()` function in our`getTasksLengthForLabel` Controller.

label.js

JavaScript

```
$scope.getTasksLengthForLabel=function(label){

return dataService.getTasksForLabel(label).length;

};
```

We are calling the `getTasksForLabel()` service that we wrote, this function returns an array so we can use the `length` attribute to get the size.

Load the index.html and now the real count will be displayed for all the custom labels.

For Inbox, All Tasks and All Pending we need a separate function thought.

Add the following to the Controller.

label.js

JavaScript

```
$scope.getPendingTasksLength=function(){
return dataService.getTaskByCompletionStatus(false).length;

};

$scope.getAllTasksLength=function(){
return dataService.getAllTasks().length;
};
```

Now call them from the `index.html`.

index.html

XHTML

```
<li><a href="#" class="inbox_icon">Inbox</a><div class="todo-badge">
{{getTasksLengthForLabel('Inbox')}}</div></li>
<li><a href="#"  class="all_icon">All Tasks</a><div class="todo-badge">{{getAllTasksLength()}}
</div></li>
<li><a href="#"  class="pending_icon">All Pending</a><div class="todo-badge">
{{getPendingTasksLength()}}</div></li>
```

Re-load the `index.html` and now you can see all the correct counts.

This is the end of Part 4, we will again continue further in Part 5.

Find the source for this part 4 in the github repository.

How to Learn and Master Angular easily – Part5

This How to Learn and Master Angular easily – Part5 is the most important article in our "The Best Way to Learn and Master Angular JS series". We will mostly concentrate on Routing and then restructure our TaskTracking application.

In the Previous Part 4 we have learned about Promises, Expressions and Native Angular Directive.

Routing :

Since using Angular we are creating Single Page Application(SPA), there should be a way to display dynamic content in our application.Generally the header,menu etc stays as is, the content area loads different view each time you click on a specific functionality.

In Angular we can define which view and controller to load based on any specific rule. Angular has a native library named `ngRoute`, however `ngRoute` is not very robust and in real application we use another library named ui-router.

ui-router:

Lets create a small application to understand how to use ui-router. You can download the js file from the above github repository or refer the cdn repository. Here is the URL of that :`https://cdnjs.cloudflare.com/ajax/libs/angular-ui-router/0.2.15/angular-ui-router.min.js`

Lets complete the code and make it working, sometime hands-on experience is much better then words/explanations.

index.html

XHTML

```
<html>
<head>
<link rel="stylesheet"
href="//maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap.min.css">
<script src="//cdnjs.cloudflare.com/ajax/libs/angular.js/1.3.14/angular.min.js"></script>
<script src="//cdnjs.cloudflare.com/ajax/libs/angular-ui-router/0.2.15/angular-ui-
router.min.js"></script>
<style>
.button{
  margin-left:10px;
```

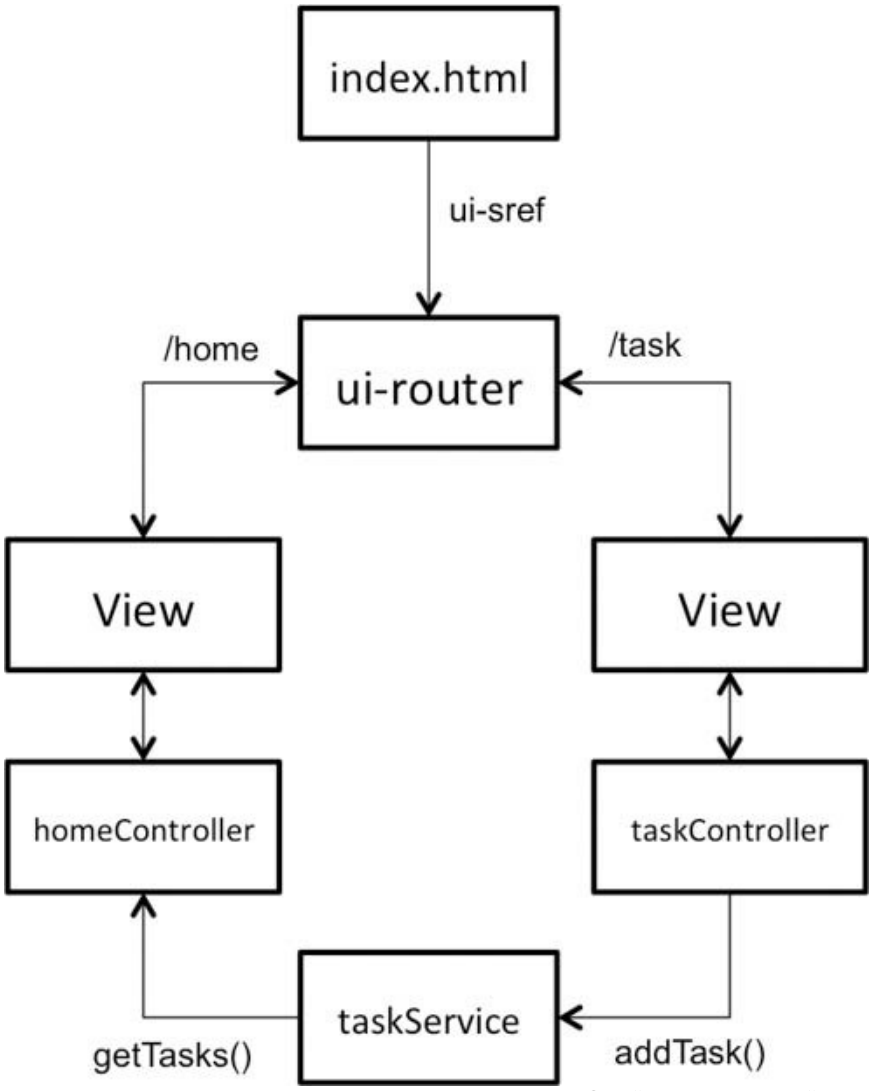
```
  height:28px;
  line-height:15px;
}

.jumbotron{
  padding:20px !important;
}
</style>
<script>
angular.module('MyApp',['ui.router'])
.config(function($stateProvider, $urlRouterProvider){

  //set default to home
  $urlRouterProvider.otherwise('/home');

  $stateProvider
  .state('home', {
  url: '/home',
  template: "<div class='jumbotron'>"+
  "<h4>List of Tasks :</h4>"+
  "<ol>"+
   "<li ng-repeat='task in tasks'>{{task}}</li>"+
  "</ol>"+
  "</div>",
  controller:'homeController'
  })
  .state('new_task', {
  url: '/new_task',
  template: "<div class='jumbotron'>"+
  "Task Name : <input type='text' ng-model='task'>"+
  "<input type='button' ng-click='addTask()' "+
   "class='btn btn-default button' value='Add task'>"+
  "</div>",
  controller:'taskController'

  })
})
.service('taskService',function(){
  var tasks=[];
  this.addTask=function(task){
tasks.push(task);
  };

  this.getTasks=function(){
return tasks;
  };
})
.controller('homeController',
function($scope,taskService){
  $scope.msg="Hello";
  $scope.tasks=taskService.getTasks();
})
.controller('taskController',
function($scope,taskService
  ,$state,$stateParams){
  $scope.task="";
  $scope.addTask=function(){
taskService.addTask($scope.task);
```

```
$scope.task="";

$state.transitionTo('home'
,$stateParams,{reload:true});

  };
})

</script>
</head>
<body ng-app="MyApp">
<nav class="navbar navbar-default" role="navigation">
    <ul class="nav navbar-nav">
        <!-- the ui-sref is the replacement for href -->
        <li><a ui-sref="home">Home</a></li>
        <li><a ui-sref="new_task">New Task</a></li>
    </ul>
</nav>
<div class="container">
  <!-- All the dynamic code will be placed here-->
  <div ui-view></div>
</div>
</body>
</html>
```

Here is the live demo. Check it out.I have few extra css to make it look better here, you can ignore them.

The home page displays number of available tasks, which will be empty at the beginning. Click on the Add Task menu and enter a Task Name, then click on Add Task. The Home page will now will display the task you have added.

I have created a simple diagram to explain whats happening. Let’s go through it.



Here we are dynamically loading the html for “Home” and “New Task” menu. The configuration of `ui-router` has been added in the `angular.config()`. Based on the ui-router configuration the appropriate html and controller will be loaded.Again both the `homeController` and `taskController` are using the`taskService` as a data store.

In order to add ui-router to our application add `ui.router` in the second parameter when we create the module.

JavaScript

```
1 angular.module('MyApp',['ui.router'])
```

Next we will configure the routing configuration inside `angular.config()`function. The angular.config() will be executed during bootstraping of the application. We will always define our ui-router configuration here.You can add as many as config() in any module.

Lets start by injecting the `$stateProvider`, `$urlRouterProvider` in the`config()` function.

We need to configure a set of View (HTML) and Controller for each routing state. We cannot change the base URL of index.html since it will then reload the entire page, rather we will define a state and assign a html template and a controller for each state.We will also have a URL for each state the URL will be appended after index.html with a #, ex –`context_root/index.html#/state_url`

We need to use `$stateProvider` service to define the states. Here is the syntax:

JavaScript

```
$stateProvider
.state('home', {
url: '/home', // URL for the state.
template: "<div>Home</div>", //HTML template
controller:'homeController' //Name of the controller

})
```

We can define as many as state using the `$stateProvider.state()` function. Lets find the usability of each of the details entered here:

- State Name: The first parameter is the name of the state. We can use this state name to programmatically change the state if required.
- url: The URL can be used to manually access a specific state.
- template: We need to provide the html template for the specific view, which will be loaded once the state has been activated.
- controller: Name of the Controller for the specific html template (View). Notice, we don't have to define `ng-controller` in the template since the state configuration is already defining the name of the controller.

We have defined two states, `home` and `new_task`. Each has a controller named `homeController` and `taskController`. Each View associated with the template will be tied with the controller defined in each state.

Next use the `$urlRouterProvider` to set a default state. Incase we access an URL which does not exists it will fall back to the default one.

JavaScript

```
$urlRouterProvider.otherwise('/home');
```

Now in the body of the html, instead of using href we will be using `ui-sref` (ui state ref) to change the state. Since the default state is home the template defined in home state will be displayed. If we click on New Task then the`new_task` state will be activated and the associated template will be loaded.

XHTML

```
<li><a ui-sref="home">Home</a></li>
<li><a ui-sref="new_task">New Task</a></li>
```

Question is where the state will load the template? for that we need to use a ui-router specific directive named `ui-view`. The template defined in the state will be placed inside the `div` containing the `ui-view` directive.

XHTML

```
<div ui-view></div>
```

We are adding the task name to the `tasks` array in the `taskService` and accessing that from the `homeController` and displaying the task list.

Once the task has been added in `taskController` we need to change the state programmatically to display the home page. We need to use `$state` and`$stateParams` service for this. Use the function named `transitionTo()` of the `$state` service. The name of the state is the first parameter. The `$stateParams` would be the 2nd parameter, you can also use the $stateParams to pass any parameter, like the task name here, so that you wont need the taskService.The $stateParams will append the

parameters in the URL. The 3rd argument is configuration details, in this instance we want to reload the `home` state every time so that the controller gets invoked every time, which will call the taskService to get the updated list of tasks.

XHTML

```
$state.transitionTo('home',$stateParams,{reload:true});
```

$stateParams:

The $stateParams could be used to set any URL Parameters. You can add primitive object (string ,int) to $stateParams and retrieve that in another controller. So $stateParams is also another way one controller pass data to another controller. This approach is helpful when we have to pass only the id or name, however in case we need to pass an entire complex object, probably an Angular service is more recommended option.

Through we will be using `$stateParams` in our TaskTracker application since we need to keep the data beyond one screen, `$stateParams` often becomes helpful to get simple tasks done.

Let's take our previous example and implement using `$stateParams`. There won't be any change to the html, find the only change required in the js below.

All the parameters we will be setting we need to define that in the URL of the state (e.g – `url: '/home:taskName'`).

JavaScript

```
.config(function($stateProvider, $urlRouterProvider){
//Add the task variable to the url with a :
url: '/home:taskName',
})

.controller('homeController',function($scope,taskService,$stateParams){

//Access the task parameter using $stateParams
if($stateParams.taskName!="")
    taskService.addTask($stateParams.task);
  $scope.tasks=taskService.getTasks();
})
.controller('taskController',function($scope,taskService,$state,$stateParams){

  $scope.task="";
  $scope.addTask=function(){
   //Assign the parameter to the $stateParams
    $stateParams.taskName=$scope.task;
    $state.transitionTo('home',$stateParams,{reload:true});
    $scope.task="";
  };
})
```

Here is the same demo using `$stateParams`

I hope by now you probably have an idea on how routing works. There are few more concepts still needs to be discussed, however we will continue working on our TaskTracker application. I will write a separate article on Angular Routing in future.

If you are interested to learn more on ui-router, continue reading my another article on How use ui-

router with Angular JS

Let's use the `ui-router` in our TaskTracker application.We will use the same implementation as we learnt above, except one change. The `template`attribute takes html content in the `$stateProvider` service. In real application the html will be probably very big and it will be difficult to put all that in the template attribute.So ui-router provides another attribute named `templateUrl`which accepts a html file as the view, so we can create the html file separately and just give the path of that file here.We call this html fragment/template as`partials`.

Lets put the tasks html in a partial and define the routing rules. The `angular-ui-router.js` should already be there in the js folder. Add it in your`index.html` after angular js.

Open `app.js` and add the module dependency. Since we can create the`angular.config()` as many times we can, we will just have the exception rule defined in app.js.

app.js

JavaScript

```
var module=angular.module('taskTracker',['ui.router']);

module.config(function($stateProvider, $urlRouterProvider){

    $urlRouterProvider.otherwise('/taskList');
});
```

Here we are defaulting to `/taskList`, which will be the default view. Now create a folder named `partial` inside the `task` folder.Create a file named`taskList.template.html`.

Open `index.html` then cut the `div` with id `page-body` and paste the entire`div` in the `taskList.template.html`.

Remove the `ng-controller="taskListController"` from the taskList.template.html. We don't have to define it here, since we will configure the controller in the routing rules.

Open the index.html again and add the `ui-view` at the same place you have removed the `page-body` div.

index.html

XHTML

```
<div ui-view></div>
```

Now add the routing rule for the task list. Open the `task.js` file and paste the following.

task.js

JavaScript

```
module.config(function($stateProvider){

    $stateProvider.state('showTasks',{
        url:'/taskList',
        templateUrl:'app/task/partial/taskList.template.html',

        controller:'taskListController'
    });
});
```

Notice, we have to provide the entire path in the `templateUrl`.

Go ahead and open the index.html in browser, the taskList partial should be loaded now. You can see the page is displaying all the tasks now !!!

Minification :

In production we will always have our javascript code minified. However in Angular, so far we are having the exact dependency name e.g – $stateProvider, dataService, $scope in the function argument. If you minify the code all these will be also minified and the code will no longer work.

task.js

JavaScript

```
1  //change this to
2  angular.controller('taskListController', function($scope,dataService){});
3
4
5  //after minification - This code will not work
   angular.controller('taskListController', function(a,b){});
```

There is a way to fix it. Look at the example below:

task.js

JavaScript

```
1  angular.controller('taskListController',
2  ['$scope','dataService',function($scope,dataService){}]);
```

The 2nd argument of the controller (and other angular functions) also accepts an array of objects where the last argument should be a function. We need to define the dependency name as string in the array and then refer them in the function argument in same sequence.

Since strings won't get minifed, the name of the dependency will never change, however the function argument name will be changed. See an example below, both the lines should work fine.

JavaScript

```
1  //before minification
2  angular.controller('taskListController',
3  ['$scope','dataService',function($scope,dataService){}]);
4
5  //after minification - This will also work since a is still referring $scope
6
7  angular.controller('taskListController',
   ['$scope','dataService',function(a,b){}]);
```

Now lets change this across our code base.You can define any name of the variables in the function argument,however the standard is to have the same name as the dependency name e.g- $scope should $scope, in order to avoid confusion.Test TaskTracker once to make sure everything is working as expected.

Going forward we will always take this a standard way of writing angular code.

Next, we will load the appropriate taskList based on the selection in the Label.

We will capture the click using `ng-click` in our View (index.html) and set the selected Label in `dataService`, then we will load `/taskList`programmatically, which will get the selected Label from

the `dataService` and load the list of tasks for that Label.

Open index.html, remove the href="#" and add this to the label.

index.html

XHTML

```
<a ng-click="setLabel(label.name)">{{label.name}}</a>
```

Now lets create a method inside `labelController` named `setLabel`. We need a variable in our `dataService` to hold the name of the selected label. Lets create one.

Note : We can also use `$stateParams` here to pass the selected label name to the `taskListController`.

Add this to the `dataService` Service in `common-service.js`

common-service.js

JavaScript

```
var selectedLabel="";

this.setSelectedLabel=function(label){

selectedLabel=label;
};

this.getSelectedLabel=function(){
return selectedLabel;
};
```

Here is the setLabel() function inside `labelController`. You need to add the `$state` and `$stateParams` dependency.

common-service.js

JavaScript

```
module.controller('labelController',
    ['$scope','dataService','$state','$stateParams'
        ,function($scope,dataService,$state,$stateParams){


...
$scope.setLabel=function(label){
dataService.setSelectedLabel(label);
$state.transitionTo('showTasks',$stateParams,{reload:true});

};
}]);
```

Here we are setting the label name in the `dataService`, then loading the /taskList state by the state name `showTasks`.

Now, delete the existing line from the taskListController. Here is the full code:

task.js

JavaScript

```
module.controller('taskListController',['$scope','dataService',function($scope,dataService){


    $scope.label=dataService.getSelectedLabel();
    $scope.tasks=dataService.getTasksForLabel($scope.label);

}]);
```

There is one more change, notice we have changed our array from `$scope.data1` to `$scope.tasks`. So lets change that in `taskList.template.html`.

taskList.template.html

XHTML

```
...
<li class="list-group-item" ng-repeat="task in tasks">


...
```

Save all the files and open index.html in browser. Click on Work,Home or Personal and now you can see the appropriate tasks are getting displayed for each Label.

Now lets make it work for the other labels “Inbox”,”All Tasks” and “All Pending”. Lets call the `setLabel()` function by passing the related string.

index.html

XHTML

```
<li>
<a href="#" class="inbox_icon" ng-click="setLabel('Inbox')">Inbox</a>
...
</li>
<li>
<a href="#"  class="all_icon" ng-click="setLabel('All Tasks')">All Tasks</a>
...
</li>
<li>
<a href="#"  class="pending_icon" ng-click="setLabel('All Pending')">All Pending</a>

...
</li>
```

In `taskListController` lets call the appropriate service to get the data.

task.js

JavaScript

```
$scope.label=dataService.getSelectedLabel();

if($scope.label=="All Pending"){
$scope.tasks=dataService.getTaskByCompletionStatus(false);

}else if($scope.label=="All Tasks"){
$scope.tasks=dataService.getAllTasks();
}
else {
$scope.tasks=dataService.getTasksForLabel($scope.label);
```

```
}
```

Open the `index.html`, now the top three Labels should work fine.

When you load the page, we expect to load the Index taskList by default, however that's not working now. In order to fix it just add the following in the`common-service.js`.

common-service.js

JavaScript

```
this.getSelectedLabel=function(){

if(selectedLabel=="") {
selectedLabel = "Inbox";
}
return selectedLabel;
};
```

When we load index.html the `selectedLabel` variable would be empty so it wont find any matching tasks.Here we are defaulting the `selectedLabel` to`Inbox` if its already empty.

Now load the index.html and you should see the task getting displayed for Inbox.

Next we will work on changing the status of a task if it has been completed.If you click on any task you will see that the checkbox is working fine so far. We need to make two changes:

1. Bind the status with the checkbox
2. Cross the text if the task has been completed

In out `task` object we already have an attribute named `completed`. Let's bind it with our Checkbox.

taskList.template.html

XHTML

```
...
<input id="{{task.id}}" type="checkbox" ng-model="task.completed">

...
```

Now the checkbox should be working as expected. Notice, the moment you check or uncheck a checkbox the count displayed in the labels are also changing.

Next we should update our model object when the status changes. In order to do that, call `updateTask()` method on ng-change.

taskList.template.html

XHTML

```
...
<input id="{{task.id}}" type="checkbox" ng-model="task.completed" ng-
change="updateTask(task)">
...
```

We will pass the `task` object. Since we the checkbox and `task.completed`are already binded together the `task` will have the updated value.

We will call a service to update the task.

task.js

JavaScript

```
$scope.updateTask=function(task){

dataService.updateTask(task);
};
```

Here is the function to update the task object. In real application, you will be making a backend call from here to update the status in database.

common-service.js

JavaScript

```
this.updateTask=function(updatedTask){
angular.forEach(arrAllTasks, function(task, key) {

    if(task.id===updatedTask.id){
       arrAllTasks[key]=updatedTask;
}
});
};
```

We already have a `crossed` css class in our stylesheet. Lets use that to cross the text if the task has already completed. Add the `ng-class` to the `label`element next to the checkbox.

taskList.template.html

XHTML

```
<label for="{{task.id}}" class='checkbox-label'

ng-class="task.completed==true?'crossed':none">


{{task.name}}
</label>
```

In the `taskList.template.html` file replace the hardcoded “Inbox” inside the`panel-top-header` with `{{label}}` in order to display the selected label name.

We have covered many important topics in this part. We should be able to complete our application in next part.

Find the source for this part 5 in the github repository.

How to Learn and Master Angular easily – Part6

quarta-feira, 6 de julho de 2016 16:35

How to Learn and Master Angular easily – Part6

This is the last part of our series How to Learn and Master Angular easily – Part6. We will complete our TaskTracker application and will also learn about Angular filter, $rootScope, $watch etc.

In Part 5 we went through Routing in detail using ui-route and implemented that in the app.

Let's try to finish our application with the knowledge we have gained so far. We will start with creating the New Label.

At first we need to create a partial (View/Template) for the New Label.

1. Create a new file named `newLabel.template.html` inside`\app\label\partial`.
2. Copy the element with `id='page-body'` from the `createLabel.html` to the `newLabel.template.html` file.
3. Now you can delete the createLabel.html file.

We will create a new controller for the header named `headerController`.

4. Create another js file named `header.js` inside `/app/header` and create a controller named `headerController`.
5. Create a new function named `createNewLabel()` and inside that change the route to `newLabel`.
6. Add the `header.js` in `index.html`.

header.js

JavaScript

```
angular.module('taskTracker')
    .controller('headerController',
    ['$scope', '$state',
        function ($scope, $state, $stateParams,$rootScope) {
            $scope.createNewLabel=function(){
                $state.transitionTo('newLabel',$stateParams,{reload:true});

            };
}]);
```

7. Now define the `headerController` as `ng-controller` inside the element with id `page-content-wrapper` in the index.html.
8. Call the `createNewLabel()` function in `#btnCreateLabel` using `ng-click` in order to capture the click event.

index.js

XHTML

```
...
<div id="page-content-wrapper" ng-controller="headerController" >



...
<button type="button" id="btnCreateLabel"
class="btn btn-danger pull-right btn-task-category"
ng-click="createNewLabel()">Create Label</button>

</div>
...
```

In summary all we have done is to change the state to `newLabel` from the`headerController` when the `#btnCreateLabel` is clicked.However we don't have the state defined for showing the `newLabel.template.html`. Let's work on that.

Add new state named newLabel in the label.js.

label.js

JavaScript

```
module.config(['$stateProvider',
function($stateProvider){
$stateProvider.state('newLabel',{
url:'/newLabel',
templateUrl:'app/label/partial/newLabel.template.html',

controller:'newLabelController'
});
}])
```

Add the new controller named `newLabelController` in the label.js.

Now if you open index.html in browser and click on "New Label", the New Label partial (view) should be displayed.

In this New Label we can enter a Label Name and select a color for the New Label then save it.Here are the changes we need to make in the`newLabel.template.html`.

1. Add the `ng-model="name"` in the text field to bind the field with the model named `name`.
2. Then replace `onclick()` function in the `Save` button with `ng-click="saveLabel()"`
3. In order to store the color value we will define a model named `color` in our `newLabelController` and based on the selected color we will display the tick icon. Let's call a method named `setColor()` from each of the buttons for setting the color and show/hide the tick icon using an expression in `ng-show` directive. Here is the changed code. We will pass the name of the css class in the `setColor()` function.I already have a css class defined for each color in the `style.css`
4. So once you click on the color button, thd `setColor()` will be called to set the `$scope.color` value, then the tick will be displayed based on the new `$scope.color` value. This will work automatically since Angular has Two Way Binding !

newLabel.template.html

XHTML

```
<div class="col-sm-12">
<label class="col-sm-2 form-label">Color :</label>
<div class="btn-group" role="group" aria-label="...">
<button type="button" class="btn btn-danger" ng-click="setColor('color_red1')">

<i class="fa fa-check" ng-show="color=='color_red1'"></i> 
</button>
```

```
<button type="button" class="btn btn-info" ng-click="setColor('color_sky')">
<i class="fa fa-check" ng-show="color=='color_sky'"></i> 
</button>
<button type="button" class="btn btn-warning" ng-click="setColor('color_yellow')">

<i class="fa fa-check" ng-show="color=='color_yellow'"></i> 
</button>
<button type="button" class="btn btn-success" ng-click="setColor('color_green1')">

<i class="fa fa-check" ng-show="color=='color_green1'"></i> 
</button>
<button type="button" class="btn btn-primary" ng-click="setColor('color_blue1')">

<i class="fa fa-check" ng-show="color=='color_blue1'"></i> 
</button>
</div>
</div>
```

Here is the code for the `newLabelController` that we created.

label.js

JavaScript

```
module.controller('newLabelController',
   ['$scope','$state','$stateParams','dataService',
   function($scope,$state,$stateParams,dataService){

      $scope.name="";
      $scope.color=""

      $scope.setColor=function(value){
         $scope.color=value;
      };

      $scope.saveLabel=function(){

         var label={
            "name":$scope.name,
            "color":$scope.color
         };

         dataService.addNewLabel(label);

         dataService.setSelectedLabel($scope.name);
         $state.transitionTo('showTasks',$stateParams,{reload:true});


      };

   }]);
```

In `newLabelController` we are creating label object using the `$scope.color`and `$scope.name` then calling a method in the `dataService` named`addNewLabel()` to add the New Label to the label array. Then we are setting the selectedLabel as the created label and changing the state to `showTasks`, which will display the list of tasks available for the new label (which will be empty).

This is the code to add the label object to the label array present in the`dataService` service.

common-service.js

JavaScript

```
this.addNewLabel=function(label){

arrLabels.push(label);
};
```

Now open the index.html and click on New Label then add the name of the New Label and click on a color. Click on Save. You should have the label created and the taskList should be loaded for the label.

I will have you work on the “Add Task” button, which is very similar to the New Label Function. Here is the list of steps and code for your reference.

1. Create a new partial named `newTask.template.html` inside `/app/task/partial` and paste the dynamic content from the `createTask.html`.
2. Delete the `createTask.html` file.
3. Now create a `createNewTask()` function inside `taskListController` to change the state to `newTask`.
4. Add a new state named `newTask` and create a new controller named `newTaskController` inside `task.js`.

taskList.template.html

XHTML

```
<div class="col-sm-1">
<button type="button" class="btn btn-info pull-right"

ng-click="createNewTask()">Add Task</button>
</div>
```

task.js

JavaScript

```
module.config(['$stateProvider', '$urlRouterProvider',function($stateProvider){
...
// Add the new state named newTask
.state('newTask',{
            url:'/newTask',
            templateUrl:'app/task/partial/newTask.template.html',
            controller:'newTaskController'
        });

}]);

// add the createNewTask() function to change the state.
//The other way of doing this is using ui-sref
module.controller('taskListController',['$scope','dataService','$state','$stateParams'

    ,function($scope,dataService,$state,$stateParams){
...
$scope.createNewTask=function(){
        $state.transitionTo('newTask',$stateParams,{reload:true});
    };

}]);

// Add the newTaskController to save the Task.
module.controller('newTaskController',['$scope','$state','$stateParams','dataService',

        function($scope,$state,$stateParams,dataService) {

        $scope.name="";
        $scope.due_date="";
        $scope.labelName=dataService.getSelectedLabel();
```

```
32
33
34          $scope.saveTasks=function(){
35
36              if($scope.name.trim()!="") {
37
38                  $scope.task = {
39                      "id": (new Date()).getTime(),
40                      "name": $scope.name,
41                      "due_date": $scope.due_date,
42                      "completed": false,
43                      "labelName": $scope.labelName
44
45                  };
46                  dataService.addNewTask($scope.task);
47
48                  $state.transitionTo('showTasks', $stateParams, {reload: true});
49              }
            };
        }]);
```

common-service.js

JavaScript

```
1           this.addNewTask=function(tasks){
2
3               arrAllTasks.push(tasks);
            };
```

5. In the `newTask.template.html` file, replace onclick of the Save button to `ng-click="saveTasks()"`.
6. Add `ng-model="name"` in the textbox for Task Name and `ng-model="due_date"` for the Due Date text box.

Note : I have not implemented the date or note field in the New Task view. This is something you may want to work on and comeplete.

Save all the files and test the code now. You can always find the full codebase in github, link given at the end of the article.

In the `taskList.template.html` we have a text box on top of the Task Lists to create task just by entering the task name and then pressing enter/return. We would need to create a method named `createQuickTask()` inside `taskListController`.

task.js

JavaScript

```
1   // Add this inside taskListController.
2
3   $scope.tasks=[];
4   $scope.txtTaskName="";
5   $scope.createQuickTask=function(){
6
7   if($scope.txtTaskName.trim()!=""){
8   $scope.task={
9   "id":(new Date()).getTime(),
10  "name":$scope.txtTaskName,
11  "due_date":null,
12  "completed":false,
13  "labelName":$scope.label
14  };
15  dataService.addNewTask($scope.task);
16  $scope.tasks=dataService.getTasksForLabel($scope.label);
```

```
17
18  $scope.txtTaskName="";
19  }
20  };
```

In the `taskList.template.html`, bind the textbox with `$scope.txtTaskName`and call the `createQuickTask()` method on pressing enter. Here is the code :

taskList.template.html

XHTML

```
1  <div class="col-sm-10" >
2  <input type="text" class="form-control input-create-task"
3     placeholder="Create New Task" ng-model="txtTaskName"
4           ng-keyup="$event.keyCode==13 ? createQuickTask() : null"/>
5
   </div>
```

We are using a new directive here named `ng-keyup`. The `$event.keyCode`provides the ACSII code for the key pressed. In this instance we will validate whether it's 13, which is the ACSII code for enter key, then call the`createQuickTask()` method.You can see expression can be very useful.

Now test the code by entering a task name and press enter. The new task should be added automatically.

in case you haven't noticed, there is a filter textbox in the header of our Task Tracker application. The list of tasks displayed should be filtered by the text that has been entered into the filter textbox.

In order to implement this, we need to first capture any text entered in the filter textbox then pass the details to the `taskListController`, then apply a "filter" to display only teh tasks name matches with the entered text in Filter.

In index.html lets bind a model named `txtFilter` with the filter textbox and call a method named `updateFilter()` on `ng-change`

index.html

XHTML

```
1  ...
2  <input type="text" class="form-control" ng-model="txtFilter" ng-change="updateFilter()"/>
3
   ...
```

Since the header already has a controller named `headerController` we are going to create our updateFilter() there.

header.js

JavaScript

```
1  $scope.txtFilter="";
2
3  $scope.updateFilter=function(){
4  $rootScope.filter = $scope.txtFilter;
5
   };
```

Here we are assigning the `$scope.txtFilter` to `$rootScope.filter`.

$rootScope :

We know that the `$scope` is used between controller and view to share the objects (model,

function). `$rootScope` is very similar to a variable with global scope, means anything you put in $rootScope can be accessed from any controller or service.

Here we are using `$rootScope` to share the value of the filter text field with`taskListController`.

Note : You need to be very careful while using `$rootScope` since it may impact the performace of the application.

Next, in the `taskListController` we need something like a listener to get the values of the `$rootScope.filter` only when it has changed.

$watch :

`$watch` is a function in angular to listen to any change in either `$scope` or`$rootScope`. If you attach `$watch` to any model then angular will have periodically look for any change in the model.You need to be careful while using $watch since using too many of them will impact the performace very badly. There is something called `Digest Cycle` in Angular which I am not going to talk about in this series, however look for another artical on this topic in future.

task.js

JavaScript

```
module.controller('taskListController',['$scope','$state','$stateParams','dataService','$rootScope',

        function($scope,$state,$stateParams,dataService,$rootScope){

$scope.filterText={
name:''
};

$rootScope.$watch('filter',function(data){
if((data!=undefined || data!=null)){
$scope.filterText.name=data;
}
});

...

}])
```

The first argument of the `$watch()` is the name of the model, `filter` in this case. The 2nd argument is a function, which will be called whenever there is a change in the model.

We are setting the updated value of the filter text to `$scope.filterText.name`model in the `taskListController`.

Angular Filter:

Angular provides many native functions to filter the data and reformat them. We can also create our own custom filters. We will be using one of the predefined Angular filter in our Task Tracker application to filter the list of tasks. You can access the filter in the View or from the Controller and Service. In order to access a filter from View (HTML) you need to use the a pipe (|), e.g. – `{{name | uppercase}}`.

Here is a demo of some of the available filters.

We can also create custom filters, however that will be a seperate discussion. In real application you don’t have to write too many filters though.

Check our the last example, where you can filter the names so easily. We will implement the similar

solution in our application.

taskList.template.html

XHTML

```
...
<li class="list-group-item" ng-repeat="task in tasks | filter:filterText:strict">

...
```

Save and test the app, it should now work fine.

I am using the `strict` option here to filter only the name field in the task object. Since the task object has many fields, if we simply use a filter like in the demo above, it will apply filter on each field. However the `filterText` only has one field named `name` which is the same as the `name` present in a task object.In order to match `filtertext.name` to `task.name` we are setting the filter text to `filterText.name` and not directly to the `filterText`.

You can find the full source code for Part 6 here.

So we have successfully completed learning the basiscs of Angular !! I hope this The Best Way to Master Angular JS series will help you to have a strong foundation in Angular JS.

As a next step, you can complete the TaskTracker application. There are many action icons for each tasks, you can enable them. Also to help you, I have already implemented the delete task functionality. You can find the code here. I also have few more minor changes.

P.S. I will publish more articles on Routing, Directive, Filter etc.

About Abhisek Jana

Abhisek's journey as a developer started with programming in BASIC using a BBC Micro Computer at his middle school. He is now an Architect, essentially focusing on web technologies. He has been always very passionate about programming and learning new technology.

De <http://www.adeveloperdiary.com/angular-js/angular-1-x/the-best-way-to-master-angular-js-part-6/>