A Developer Diary
{about:"code learn and share"}

October 14, 2015 By Abhisek Jana — Leave a Comment (Edit)

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`.

```
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.

```
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:

```
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()`

```
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);
});
```

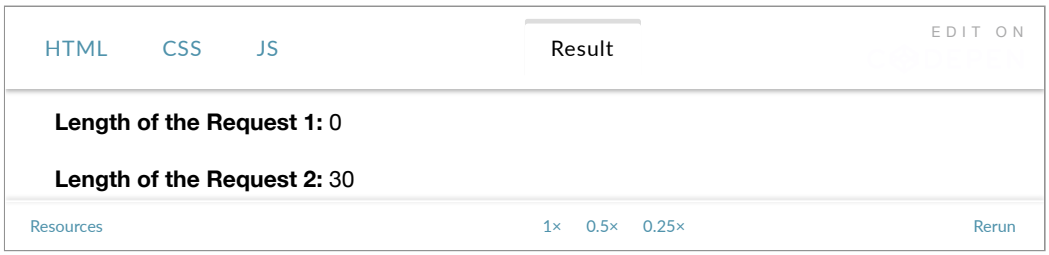
Here is a full working example of a `$http` call with and with-out the Promises.

```
Length of the Request 1:

{{data1.length}}
```

```
Length of the Request 2:

{{data2.length}}
```

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.

```
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.

```
this.getAllLabels=function(){
        return arrLabels;
};
```

Now lets complete the `labelController`.

```
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.

```
this.getAllTasks=function(){
        return arrAllTasks;
};
```

Here is the updated `taskListController`.

```
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.

```
        ...


 ...
```

```
        ...
```

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.

```
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.

```
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.

```
        {{label.name}}

   {{getTasksLengthForLabel(label.name)}}
```

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

```
$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.

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

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

Now call them from the `index.html`.

```
Inbox
   {{getTasksLengthForLabel('Inbox')}}

All Tasks
   {{getAllTasksLength()}}

All Pending
```

```
{{getPendingTasksLength()}}
```

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.

Related

How to Learn and Master Angular easily – Part3

In "Angular 1.x"

How to Learn and Master Angular easily – Part2

In "Angular 1.x"

How to Learn and Master Angular easily – Part6

In "Angular 1.x"

Filed Under: Angular 1.x, JavaScript | Tagged With: Angular, Angular JS, Code, example, Expression, JavaScript, Learn, master, Programming, Promises, step by step, tutorial

Subscribe

Leave a Reply

Logged in as Abhisek Jana. Edit your profile. Log out? Required fields are marked *

Comment *

Post Comment

This site uses Akismet to reduce spam. Learn how your comment data is processed.