A Developer Diary (https://www.adeveloperdiary.com/)

{about:"code learn and share"}

Home (https://www.adeveloperdiary.com/)

Data Science (https://www.adeveloperdiary.com/category/data-science/)

Java (https://www.adeveloperdiary.com/category/java/)

JavaScript (https://www.adeveloperdiary.com/category/javascript/)

jBPM (https://www.adeveloperdiary.com/category/bpm/jbpm/)

Tools (https://www.adeveloperdiary.com/category/tools/)

Tips (https://www.adeveloperdiary.com/category/tips/)

About (https://www.adeveloperdiary.com/about-me-1/)

> October 14, 2015 By Abhisek Jana (https://www.adeveloperdiary.com/author/adeveloperdiarygmail-com/) — 2 Comments (https://www.adeveloperdiary.com/angular-js/angular-1-x/the-best-way-to-master-

> angular-js-part-5/#comments) (Edit) (https://www.adeveloperdiary.com/wp-admin/post.php?post=367&action=edit)

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 (https://www.adeveloperdiary.com/angular-js/angular-1-x/the-best-way-to-master-angular-js-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 (https://github.com/angular-ui/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
```

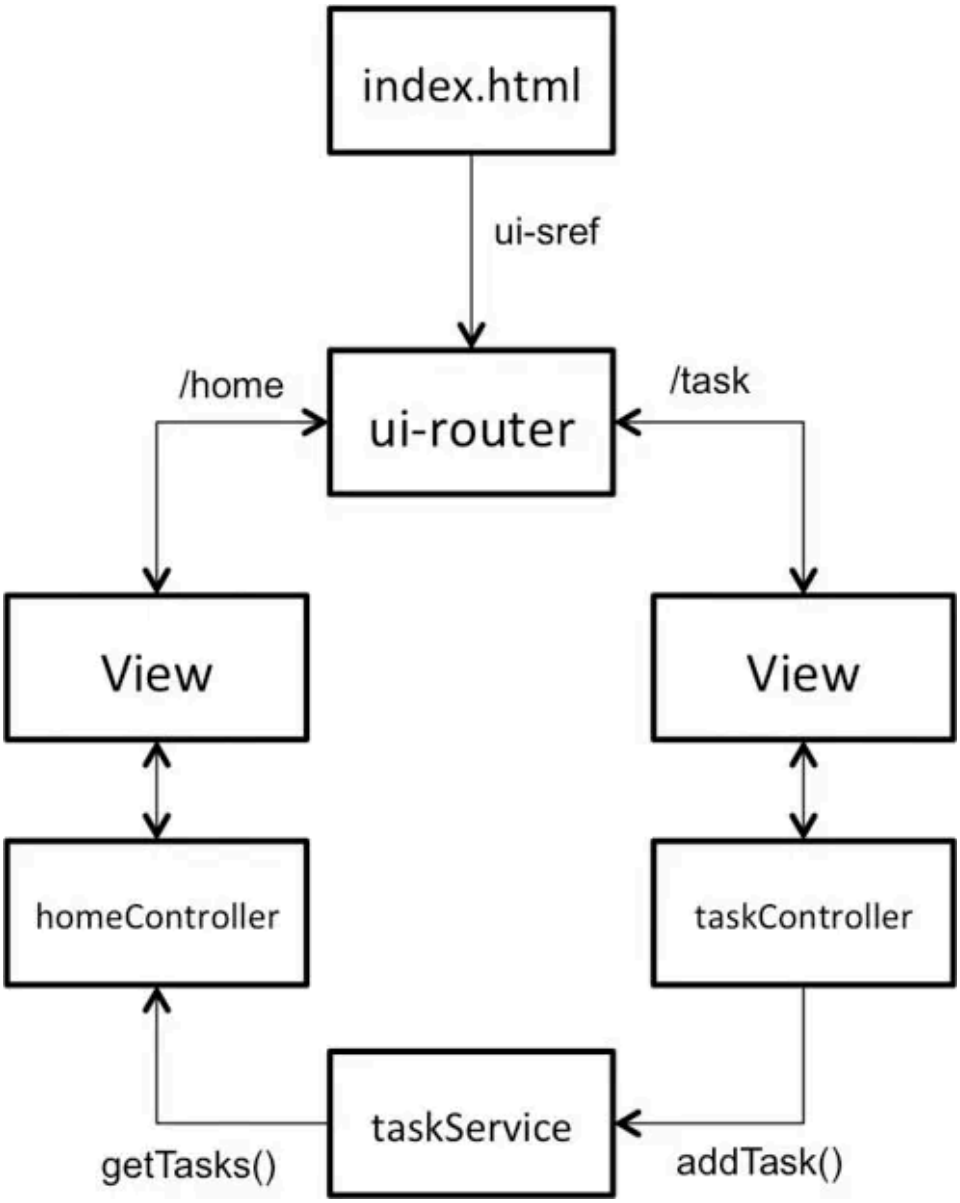
Lets complete the code and make it working, sometime hands-on experience is much better then words/explanations.

List of Tasks :

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.



(https://www.adeveloperdiary.com/angular-js/angular-1-x/the-best-way-to-master-angular-js-part-5/)

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.

```
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:

```
$stateProvider
        .state('home', {
                url: '/home',   // URL for the state.
                template: "
Home
", //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.

```
$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.

```
Home

New Task
```

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.

List of Tasks :

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 won't 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.

```
$state.transitionTo('home',$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.

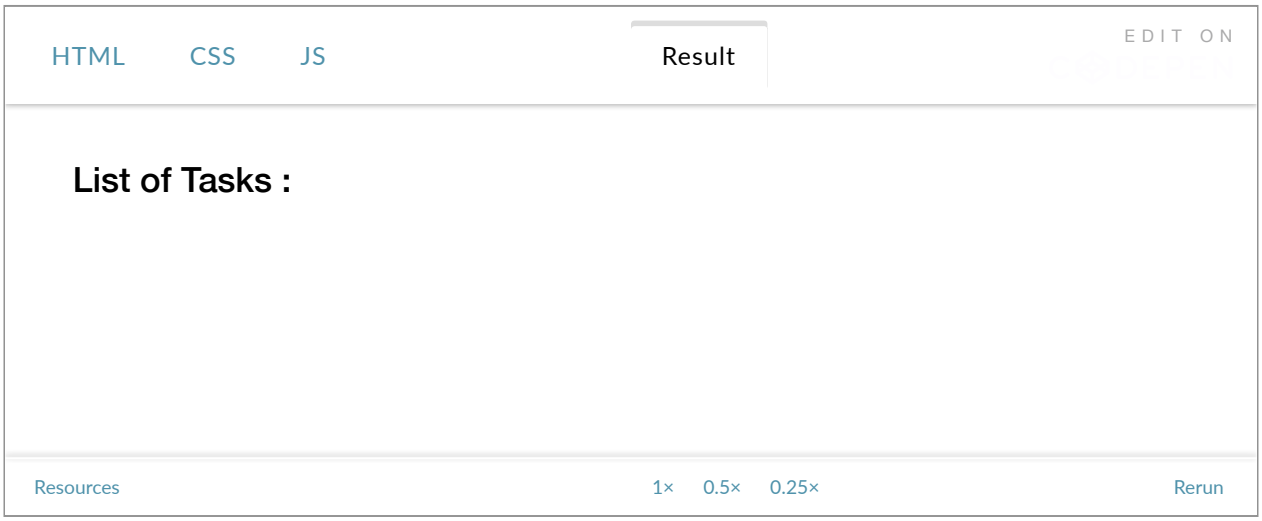
Let's take our previous example and implement using `$stateParams`. There won't be any change to the html, find the only change required in the js below.

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

```
.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 (https://www.adeveloperdiary.com/angular-js/angular-1-x/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.

```
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.

```
List of Tasks :
```

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

```
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.

```
//change this to
angular.controller('taskListController', function($scope,dataService){});

//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:

```
angular.controller('taskListController',
        ['$scope','dataService',function($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.

```
//before minification
angular.controller('taskListController',
        ['$scope','dataService',function($scope,dataService){}]);

//after minification - This will also work since a is still referring $scope
angular.controller('taskListController',
        ['$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- $scopeshould$ 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.

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`

```
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.

```
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:

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

    $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`.

```
...
```

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.

```
        Inbox
        ...


        All Tasks
        ...


        All Pending
        ...
```

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

```
$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`.

```
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.

```
 ...
☐
 ...
```

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.

```
 ...
☐
 ...
```

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.

```
$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.

```
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.

In the `taskList.template.html` file replace the hardcoded “Inbox” inside the `panel-top-header` with 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 (https://github.com/adeveloperdiary/angular-for-web-developers/tree/master/chapter5) in the github repository.

Related

(https://www.adeveloperdiary.com/angular-js/angular-1-x/the-best-way-to-master-angular-js-part-2/)

How to Learn and Master Angular easily – Part2 (https://www.adeveloperdiary.com/angular-js/angular-1-x/the-best-way-to-master-angular-js-part-2/)

In "Angular 1.x"

(https://www.adeveloperdiary.com/angular-js/angular-1-x/the-best-way-to-master-angular-js-part-1/)

How to Learn and Master Angular easily – Part1 (https://www.adeveloperdiary.com/angular-js/angular-1-x/the-best-way-to-master-angular-js-part-1/)

In "Angular 1.x"

(https://www.adeveloperdiary.com/angular-js/angular-1-x/the-best-way-to-master-angular-js-part-3/)

How to Learn and Master Angular easily – Part3 (https://www.adeveloperdiary.com/angular-js/angular-1-x/the-best-way-to-master-angular-js-part-3/)

In "Angular 1.x"

Filed Under: Angular 1.x (https://www.adeveloperdiary.com/category/angular-js/angular-1-x/), JavaScript (https://www.adeveloperdiary.com/category/javascript/) Tagged With: Angular (https://www.adeveloperdiary.com/tag/angular/), Angular JS (https://www.adeveloperdiary.com/tag/angular-js/), Code (https://www.adeveloperdiary.com/tag/code/), example (https://www.adeveloperdiary.com/tag/example/), JavaScript (https://www.adeveloperdiary.com/tag/javascript/), Learn (https://www.adeveloperdiary.com/tag/learn/), master (https://www.adeveloperdiary.com/tag/master/), ngRoute (https://www.adeveloperdiary.com/tag/ngroute/), Programming (https://www.adeveloperdiary.com/tag/programming/), Routing (https://www.adeveloperdiary.com/tag/routing/), step by step (https://www.adeveloperdiary.com/tag/step-by-step/), tutorial (https://www.adeveloperdiary.com/tag/tutorial/)

Comments

Shahzan says
January 21, 2016 at 9:57 am (https://www.adeveloperdiary.com/angular-js/angular-1-x/the-best-way-to-master-angular-js-part-5/#comment-170)

(Edit) (https://www.adeveloperdiary.com/wp-admin/comment.php?action=editcomment&c=170)

HI A Developer Diary,

Thanks a lot for this amazing tutorials.

However I'm facing one problem here right after creating PArtials view when I run index.html, I see this Error: $injector : modulerrModuleErrorFailedtoinstantiatemoduletaskTrackerdueto : Error :[injector:unpr] Unknown provider: $stateProvider.

Could you help me with this.

Thanks

Reply

A Developer Diary (https://www.adeveloperdiary.com/) says
January 27, 2016 at 3:52 pm (https://www.adeveloperdiary.com/angular-js/angular-1-x/the-best-way-to-master-angular-js-part-5/#comment-178)

(Edit) (https://www.adeveloperdiary.com/wp-admin/comment.php?action=editcomment&c=178)

Hi Shahzan,
Sorry for the late response. Please make sure you have angular-ui-router.js added in index.html. Also add the same in module dependency in app.js.

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

Let me know whether this resolves the issue. Glad to hear that this tutorial has helped you !

Reply

Leave a Reply

Logged in as Abhisek Jana. Edit your profile (https://www.adeveloperdiary.com/wp-admin/profile.php). Log out? (https://www.adeveloperdiary.com/wp-login.php?action=logout&redirect_to=https%3A%2F%2Fwww.adeveloperdiary.com%2Fangular-js%2Fangular-1-x%2Fthe-best-way-to-master-angular-js-part-5%2F&_wpnonce=c3573f4cf6) Required fields are marked *

Comment *

Post Comment