AngularJS--第五天 课程

ng-init：初始化方法。作用同于（$scope.name='hi';）

```
<!DOCTYPE html>
<html ng-app="myApp">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <script src="js/angular.min.js"></script>
    <script>
    var m1 = angular.module('myApp',[]);
    m1.controller('Aaa',["$scope",function($scope){
        // $scope.name='hi';
        $scope.arr = [['aaa','bbb'],['ccc','ddd']]

    }])
    </script>
</head>
<body>
    <!-- <div ng-controller='Aaa' ng-init="name='hello'">
    ng-init="name='hello'"相当于$scope.name='hi';
        <p>{{ name }}</p>
    </div> -->
    <!-- <div ng-controller='Aaa' >
        <div ng-repeat="arr1 in arr">
            <div ng-repeat="arr2 in arr1">
            遍历出数组的每个值
                <p>{{arr2}}</p>
            </div>
        </div>
    </div> -->
    <div ng-controller='Aaa' >
        <div ng-repeat="arr1 in arr" ng-init="arr1Index=$index">
            <div ng-repeat="arr2 in arr1" ng-init="arr2Index=$index">
            <!-- 通过$index方法得到相应的下标 -->
                <p>{{arr2}}:{{arr1Index}}{{arr2Index}}</p>
            </div>
        </div>
    </div>
</body>
</html>
```

在嵌套循环中可以应用ng-init方法定义一些变量。方便我们操作。

ng-model 中的ng-model-options和updateOn

ng-model-options：可以根据自己需求配置效果

updateOn：是属性

案例

```
<!DOCTYPE html>
<html ng-app="myApp">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <script src="js/angular.min.js"></script>
    <script>
    var m1 = angular.module('myApp',[]);
    m1.controller('Aaa',["$scope",function($scope){
        $scope.name='请输入内容！';

    }])
    </script>
</head>
<body>
    <div ng-controller='Aaa' >
        <input type="text" ng-model="name" ng-model-options="{updateOn:'blur'}">
        <p>{{name}}</p>
    </div>
</body>
</html>
```

ng-controller中的as方法

案例

```
<!DOCTYPE HTML>
<html ng-app="myApp">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>无标题文档</title>
<script src="js/angular.min.js"></script>
<script>
var m1 = angular.module('myApp',[]);
m1.controller('Aaa',['$scope',FnAaa]);

function FnAaa($scope){
}
FnAaa.prototype.num = '123';
FnAaa.prototype.text = 'hello';
FnAaa.prototype.show = function(){
    return 'angularJS';
};

</script>
</head>
<body>
<div ng-controller="FnAaa as a1">
            <!-- 用面向对象方式 -->
    <div>{{a1.text}}:{{a1.show()}}</div>
</div>
</body>
</html>
```

标签指令：在原有的html标签上进行升级。

<a>：在angular中把刷新页面问题给屏蔽了。

案例

```
<!DOCTYPE html>
<html >
<head>
    <meta charset="UTF-8">
    <title>Document</title>
```

```
    <script src="js/angular.min.js"></script>
    <script>
    var m1 = angular.module('myApp',[]);
    m1.controller('Aaa',["$scope",function($scope){
        $scope.name='hi';

    }])
    </script>
</head>
<body>
    <div ng-app="myApp" ng-controller='Aaa' >
        <a href="">a标签angular</a>
    </div>
    <a href="">a标签</a>
</body>
</html>
```

<select>：下拉列表的指令ng-options用for in方法生成样式

案例

```
<!DOCTYPE html>
<html >
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <script src="js/angular.min.js"></script>
    <style>
        .red{
            width:100px;
            height: 100px;
            background: red;
        }
        .gray{
            width:100px;
            height: 100px;
            background: gray;
        }
        .green{
            width:100px;
            height: 100px;
            background: green;
        }
    </style>
    <script>
    var m1 = angular.module('myApp',[]);
    m1.controller('Aaa',["$scope",function($scope){
        $scope.colors=[
            {name:'red'},
            {name:'green'},
            {name:'gray'}
        ];

    }])
    </script>
</head>
<body>
    <!-- <div ng-app="myApp" ng-controller='Aaa' >
        <a href="">a标签angular</a>
    </div>
    <a href="">a标签</a> -->
    <div ng-app="myApp" ng-controller='Aaa' >
        <select ng-options="color.name for color in colors" ng-model="Bbb">

        </select>
```

```
        <!-- <p>{{Bbb.name}}</p> -->
        <!-- 获取相应的数据 -->
        <div class="{{Bbb.name}}"></div>
        <!-- 获取到数据并改变div的样式 -->
    </div>
</body>
</html>
```

<textarea><input><form>：有一些html的自带样式用novalidate属性可以阻止自带样式的触发。

H5标签 在火狐可以实现type：email

```
<!DOCTYPE html>
<html ng-app="myApp">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <script src="js/angular.min.js"></script>
    <script>
    var m1 = angular.module('myApp',[]);
    m1.controller('Aaa',["$scope",function($scope){
        $scope.name='hello';
    }]);

    </script>
</head>
<body>
    <div ng-controller='Aaa'>
        <form novalidate action="">
            <input type="email" />
        </form>
    </div>

</body>
</html>
```

表单验证（type类型为email、number、url。。。）
$valid：表单验证通过时是true、验证失败了就是false

$invalid：表单验证通过时是false、验证失败了就是true。

$pristine：当我们的值是一个初始值（未修改的）时候是true、当这个值被修改了就是false。

$dirty：当我们的值是一个初始值（未修改的）时候是false、当这个值被修改了就是true。

$error：当我验证失败或者成功时、都会得到相应的信息。可以根据相应的数据，进行操作。

这些方法都是通过name的方式进行查找、要写ng-model

```
<!DOCTYPE html>
<html ng-app="myApp">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <script src="js/angular.min.js"></script>
    <script>
    var m1 = angular.module('myApp',[]);
    m1.controller('Aaa',["$scope",function($scope){
        $scope.text='hi';

    }])
    </script>
</head>
<body>
    <div ng-controller='Aaa' >
        <form name="oForm">
            <!-- <input type="text" name="oInput" ng-model="text"> -->
            <input type="email" name="oInput" ng-model="text">
            <p>{{ oForm.oInput.$valid }}</p>
            <p>{{ oForm.oInput.$invalid }}</p>
            <p>{{ oForm.oInput.$pristine }}</p>
            <p>{{ oForm.oInput.$dirty }}</p>
            <p>{{ oForm.oInput.$error }}</p>
        </form>
    </div>
</body>
</html>
```

required：验证是否为空。

ng-minlength：数据的最小长度是。

ng-maxlength：数据的最大长度是。

ng-pattern：正则表达式。

```
<!DOCTYPE html>
<html ng-app="myApp">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <script src="js/angular.min.js"></script>
    <script>
    var m1 = angular.module('myApp',[]);
    m1.controller('Aaa',["$scope",function($scope){
        $scope.text='hi';

    }])
    </script>
</head>
<body>
    <div ng-controller='Aaa' >
        <form name="oForm">
            <input type="text" name="oInput" ng-model="text" required ng-minlength="5" ng-maxlength="10" ng-
pattern="/^[a-zA-Z]+$/">
            <!-- required：判断是否为空。 -->
            <!-- ng-minlength="5"：数据的最小长度是5。 -->
            <!-- ng-maxlength="10"：数据的最大长度是10。 -->
            <!-- ng-pattern="/^[a-zA-Z]+$/"：用正则表达式验证。 -->
            <p>{{ oForm.oInput.$valid }}</p>
            <p>{{ oForm.oInput.$invalid }}</p>
            <p>{{ oForm.oInput.$pristine }}</p>
            <p>{{ oForm.oInput.$dirty }}</p>
            <p>{{ oForm.oInput.$error }}</p>
        </form>
```

```
    </div>
</body>
</html>
```

通过定义class实现项目需求

.ng-valid{}：验证通过

.ng-invalid{}：验证未通过

.ng-pristine{}：初始值

.ng-dirty{}：不是初始值

```
<style>
        input.ng-valid{
            background: blue;
        }
        input.ng-invalid{
            background: red;
        }
 </style>
```

一个简单的表单验证

案例一

```
<!DOCTYPE html>
<html ng-app="myApp">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <script src="js/angular.min.js"></script>
    <style>
        /*input.ng-valid{
            background: blue;
        }
        input.ng-invalid{
            background: red;
        }*/
        .red{
            color: red;
        }
    </style>
    <script>

    var m1 = angular.module('myApp',[]);
    m1.controller('Aaa',["$scope",function($scope){
        $scope.oFormText ={
                oFormVal:'default',
                oFormList:[
                { name : 'default' , tips : '请输入账号！' } ,
                { name : 'required' , tips : '账号不能为空！' } ,
                { name : 'pattern' , tips : '请输入英文！' } ,
                { name : 'ok' , tips : '√' } ,
                ],
                change : function(err){
                        // console.log(err);
                        // 值为false时是通过验证，为true时是验证失败
                        for( var attr in err){
                            if(err[attr] == true){
                                // if判断attr是否为true
                                this.oFormVal=attr;
                                // 如果是true就给oFormVal进行从新赋值
                                return;
```

```
                    // 结束
                  }
                }
                 this.oFormVal='ok';
                 // 如果if中的条件都不符合，那就说明通过验证。
                 // 就给oFormVal进行从新赋值ok。
            }
        };
        $scope.oFormPassword ={
            oFormVal:'default',
            oFormList:[
            { name : 'default' , tips : '请输入密码！' } ,
            { name : 'required' , tips : '密码不能为空！' } ,
            { name : 'minlength' , tips : '请输入至少6位密码！' } ,
            { name : 'ok' , tips : '√' } ,
            ],
            change : function(err){
                // console.log(err);
                // 值为false时是通过验证，为true时是验证失败
                for( var attr in err){
                  if(err[attr] == true){
                    // if判断attr是否为true
                    this.oFormVal=attr;
                    // 如果是true就给oFormVal进行从新赋值
                    return;
                    // 结束
                  }
                }
                 this.oFormVal='ok';
                 // 如果if中的条件都不符合，那就说明通过验证。
                 // 就给oFormVal进行从新赋值ok。
            }
        };
    }])
    </script>
</head>
<body>
    <div ng-controller='Aaa' >
        <form name="oForm">
            <div>
                <label>账号：</label>
                <input type="text" name="unText" ng-model="oFormText.un" required ng-pattern="/^[a-zA-Z]+$/" ng-
blur="oFormText.change(oForm.unText.$error)">
                <!-- <span ng-repeat="Fm in oFormText.oFormList | filter:'default' ">{{Fm.tips}}</span> -->
                <!-- 步骤一：静态方法改变 -->
                <!-- 用筛选方法filter，只让默认文字显示default的对应文字 -->

                <span ng-repeat="Fm in oFormText.oFormList | filter:oFormText.oFormVal " class="red">{{Fm.tips}}</span>

                <!-- 步骤二：动态方法获取值 -->
                <!-- 用筛选方法filter，在数据中添加一条oFormVal：'default'。在筛选中添加这个条件oFormText.oFormVal的对应文字 -->

                <!-- 步骤三：添加条件和触发事件 -->
                <!-- 添加required为空判断、ng-pattern="/^[a-zA-Z]+$/"正则判断、ng-
blur="oFormText.change(oForm.unText.$error)当文本框失去焦点时候触发chenge事件 -->
            </div>
            <div>
                <label>密码：</label>
                <input type="password" name="upText" ng-model="oFormPassword.up" required ng-minlength="6" ng-
blur="oFormPassword.change(oForm.upText.$error)">
                <span ng-repeat="Fm in oFormPassword.oFormList | filter:oFormPassword.oFormVal " class="red">{{Fm.tips}}
</span>
            </div>
        </form>
```

```
    </div>
</body>
</html>
```

案例（封装方法）

```
<!DOCTYPE html>
<html ng-app="myApp">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <script src="js/angular.min.js"></script>
    <style>
        /*input.ng-valid{
            background: blue;
        }
        input.ng-invalid{
            background: red;
        }*/
        .red{
            color: red;
        }
    </style>
    <script>

    var m1 = angular.module('myApp',[]);
    m1.controller('Aaa',["$scope",function($scope){
        $scope.oFormText ={
                oFormVal:'default',
                oFormList:[
                { name : 'default' , tips : '请输入账号！' } ,
                { name : 'required' , tips : '账号不能为空！' } ,
                { name : 'pattern' , tips : '请输入英文！' } ,
                { name : 'ok' , tips : '√' } ,
                ]
        };
        $scope.oFormPassword ={
                oFormVal:'default',
                oFormList:[
                { name : 'default' , tips : '请输入密码！' } ,
                { name : 'required' , tips : '密码不能为空！' } ,
                { name : 'minlength' , tips : '请输入至少6位密码！' } ,
                { name : 'ok' , tips : '√' } ,
                ]

        };

        $scope.change = function(oF,err){
            for( var attr in err){
                        if(err[attr] == true){
                            $scope[oF].oFormVal=attr;
                            return;
                        }
                    }
                     $scope[oF].oFormVal='ok';
        }
    }])
    </script>
</head>
<body>
    <div ng-controller='Aaa' >
        <form name="oForm">
            <div>
                <label>账号：</label>
                <input type="text" name="unText" ng-model="oFormText.un" required ng-pattern="/^[a-zA-Z]+$/" ng-
```

```
blur="change('oFormText',oForm.unText.$error)">
            <span ng-repeat="Fm in oFormText.oFormList | filter:oFormText.oFormVal " class="red">{{Fm.tips}}</span>
        </div>
        <div>
            <label>密码：</label>
            <input type="password" name="upText" ng-model="oFormPassword.up" required ng-minlength="6" ng-
blur="change('oFormPassword',oForm.upText.$error)">
            <span ng-repeat="Fm in oFormPassword.oFormList | filter:oFormPassword.oFormVal " class="red">{{Fm.tips}}
</span>
        </div>
    </form>
  </div>
</body>
</html>
```

自定义指令directive

```
//m1.directive('hi',function(){
    return{

    };
  });
  // 创建一个自定义指令，接受两个参数一个是指令的名字，另一个是回调函数、通过return一个对象方式。
```

下面是四个配置选项。

restrict：指定类型

四种定义方式

1. "E" ：element元素（标签）

```
<hi></hi>
```

如果restrict： "E" ；那么上面的指令名就相当于一个标签

2.“A” ：属性

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

如果 restrict:"A",那么上面的指令名就相当于一个属性

3.“C” ：class的类名

```
<div class="hi"></div>
```

如果 restrict:"C",那么上面的指令名就相当于一个类名

replace：替换作用，替换外层包裹的标签功能

template：模板

templateUrl：引入外部文件

案例一

```
<!DOCTYPE html>
<html ng-app="myApp">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
  <script src="js/angular.min.js"></script>
  <script>
  var m1 = angular.module('myApp',[]);
  // m1.directive( '名字' , function(){});
  // 创建一个自定义指令，接受两个参数一个是指令的名字，另一个是回调函数。
  m1.directive('hi',function(){
```

```
    return{
          // restrict:"E",
          // hi的类型定义是E-element标签
          // 一定要大写、小写不识别。
          //
          // restrict:"A",
          //  hi的类型定义是A属性
          //  可以组合使用restrict:"EA",就是标签和属性都可以支持
          //
          // restrict:"C",
          //  hi的类型定义是C  class类名
          //
            restrict:"M",
            //  hi的类型定义是M  注释
            //
             restrict:"EACM",
            replace:true,
            // 替换作用，替换外层包裹的标签和替换注释。
          template:"<p>angularJs</p>"
          // 给hi这个标签添加模板，内容为"<p>angularJs</p>"
    };
  });

  m1.controller('Aaa',["$scope",function($scope){


  }]);
  </script>
</head>
<body>
  <hi></hi>
  <!-- 标签方式一般用于模板替换或重置HTML标签功能 -->
  <!-- 属性方式一般用于功能实现 -->
  <div hi></div>
  <!-- 前两种是比较常用的、后面两种不是比较直观 -->
  <div class="hi"></div>
  <!-- directive:hi -->
  <!-- 如果没有replace:true替换这个配置的话，是不能实现替换注释的 -->
  <!-- directive:hi(这里需要有一个空格) -->
  <div ng-controller='Aaa'>
  </div>
</body>
</html>
```

案例ng写法

```
<!DOCTYPE html>
<html ng-app="myApp">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
  <script src="js/angular.min.js"></script>
  <script>
  var m1 = angular.module('myApp',[]);
  m1.directive('myHi',function(){
        // 要用驼峰是写my-hi为myHi
    return{
          restrict:"M",
           restrict:"EA",
          replace:true,
          template:"<p>angularJs</p>"
    };
  });

  m1.controller('Aaa',["$scope",function($scope){
```

```
    }]);
    </script>
</head>
<body>
    <my-hi></my-hi>
    <!-- ng-为angular系统指令  自定义指令不要用ng- -->
    <div my-hi></div>
    <div ng-controller='Aaa'>
    </div>
</body>
</html>
```

案例二