AngularJS
常用指令
1. ng-app
整个 AngularJS 项目的开始指令
在使用 RequireJS+AngularJS 中，不需要显示的添加添加 ng-app 指令，而是通过代码动态添加
```
<body ng-app="App">
</body>
```
2. ng-controller
负责与 JavaScript 中的控制器进行关联
```
<div ng-controller="Ctrl">
</div>
<script type="text/javascript">
    app.controller('Ctrl', ['$scope', function ($scope) {
        // ...
    }]);
</script>
```
3. ng-model
给表单元素进行使用，负责数据的双向绑定，代替了表单元素中的 value 属性 ng-model 是一种语法糖，不使用此指令，也可以实现同样的效果
```
<input type="input" ng-model="username" />
<h2>{{ username }}</h2>
```
4. ng-style
给元素添加 CSS 样式
```
<h2 ng-style="{ color: 'red', fontSize: '20px' }">ng-style 的使用</h2>
```
5. ng-class
给元素添加 class 值
```
<h2 ng-class="{ 'active': index === currentIndex }">ng-class 的使用</h2>
<h2 ng-class="{ 'active': true, 'normal': false }[selectedFlag]">ng-class 的使用</h2>
```
6. ng-click
给元素绑定单击事件
```
<button ng-click="showInfo()">ng-click 的使用</button>
```
7. ng-mouseover
8. ng-focus
9. ng-blur
10. ng-change
11. ng-repeat
遍历数组或对象
注意：如果数组中存在重复元素，则使用 track by $index
```
<ul>
    <li ng-repeat="(index, item) in people track by $index">{{ index + ' ' + item.name }}</li>
</ul>
```

12. ng-src

图片的 src 属性替换成 ng-src 属性进行使用

```
<img ng-src="http://ww1.sinaimg.cn/large/006tNc79gy1ffh2borp4uj30rs0rs3yv.jpg" alt="header image"/>
```

13. ng-href

a 标签的 href 属性替换成 ng-href 属性进行使用

14. ng-bind

表达式的指令形式，不会出现页面上渲染出大括号的问题

```
<h2 ng-bind="username"></h2>
```

15. ng-cloak

解决页面渲染出大括号的问题

```
<body ng-cloak></body>
```

16. ng-init

初始化数据使用

```
<div ng-init="flag = true">
    <button ng-click="flag = !flag">开/关</button>
    <h2>{{ flag }}</h2>
</div>
```

17. ng-show

18. ng-hide

19. ng-if

20. ng-include

```
<div ng-include="'header.html'"></div>
```

控制器

```
// 打包压缩会修改形参名称，造成无法使用
.controller('Ctrl', function ($scope) {
    // ...
})
// 常规写法
.controller('Ctrl', ['$scope', function ($scope) {
    // ...
}])
```

父子控制器作用域问题

父控制器中的数据，子控制器可以直接使用

如果子控制器和父控制器中变量冲突，优先使用自己的，使用$parent 来访问父控制器中的数据

是根作用域，只要在 rootScope 上面的数据，任何控制器都可以直接使用

传值方式（父传子、子传父）

```
$scope.$on('funcName', function (event, data) {
    // ...
})
// 子传父
```

```
$scope.$emit('funcName', { a : 10 });
// 父传子
$scope.$broadcast('funcName', { b : 20 });
```

内置服务

1. $scope

添加属性和方法，渲染到页面上

```
$scope.data = ['首页', '闪送超市', '购物车', '我的'];
$scope.showInfo = function () {
    console.log(this.$index);
};
```

2. $filter

在控制器中使用过滤器

```
$scope.money = $filter('currency')(10000, '￥', 2)；
```

3. $rootScope

根部的 scope，添加到此处的值，任何控制器都可以访问

```
$rootScope.tabBarDisplayFlag = true;
```

4. $http

网络请求服务

```
$http.get('http://...').when(function (res) {
    console.log(res.data);
});
```

5. $interval

angularJS 提供的定时器

```
var timer = $interval(function () {
    $scope.count++;
}, 1000)
$interval.cancel(timer);
```

6. $timeout

angularJS 提供的定时器

```
var timer = $timeout(function () {
    $scope.count++;
}, 1000)

$timeout.cancel(timer);
```

7. $location

获取有关地址的信息

```
$scope.url = $location.absUrl();
$scope.hash = $location.hash();
console.log($location);
```

过滤器

currency

价钱的显示

```
<h3>{{ 100000 | currency: ' ￥ ': 0 }}</h3>
```

2. number

数字千位分隔符

```
<h3>{{ 123456789 | number: 2 }}</h3>
```

3. orderBy

按照某种方式排序

```
<li ng-repeat="person in people | orderBy: 'age'"></li>
<li ng-repeat="person in people | orderBy: '-age'"></li>
<li ng-repeat="person in people | orderBy: 'age': flag"></li>
```

4. date

时间格式化输出

```
<h3>{{ time | date: 'yyyy-MM-dd HH:mm:ss' }}</h3>
```

5. limitTo

```
<h3>{{ [1, 2, 3, 4, 5] | limitTo: 3: 2 }}</h3>
<h3>{{ "李大泽啊" | limitTo: 1 }}</h3>
```

6. filter

根据关键字，在数据中进行筛选

```
<li ng-repeat="item in friends | filter: serarchText"></li>
```

7. uppercase lowercase

大小写转换

```
<h3>{{ 'lidaze' | uppercase }}</h3>
<h3>{{ 'LIDAZE' | lowercase }}</h3>
```

8. json

格式化 json 显示

```
<pre id="default-spacing">{{ {'name': 'laowang', 'age': 43} | json }}</pre>
<pre id="custom-spacing">{{ {'name': 'laozhao', age: 46} | json: 4 }}</pre>
```

9. 自定义过滤器

使用 filter()方法，进行过滤器的自定义

高阶函数，函数内部返回一个函数，内部函数返回结果

```
//  自定义过滤器，将小写字母大写
.filter('bigFilter', function () {
        return function (str) {
                return str.toUpperCase();
        };
})
//  自定义过滤器，筛选年龄大于 20 岁的人
.filter('myFilter', function () {
        // filter 的返回值函数函数
        return function (val, key, value) {
                //  内层函数返回的是结果
```

```
            // key = 'age'
            // value = 20
            var res = [];
            for (var i = 0; i < val.length; i++) {
                //  筛选出年龄大于 20 岁的人，放入新数组进行返回
                if (val[i][key] >= value) {
                    res.push(val[i])
                }
            }
            return res;
        };
})
```

自定义过滤器的使用\

```
<h3>{{ 'wo shi lidaze' | bigFilter }}</h3>
<h3 ng-repeat="p in people | myFilter: 'age': 20"></h3>
```

自定义指令

使用 directive()方法，进行自定义指令

```
.directive('myDirective', function () {
        return {
            restrict: 'ECMA', //  何种使用方式
            replace: true, //  是否替换掉原有的标签
            template: '', //  对应的模板
            templateUrl: '', //  对应的模板内容
            controller: function () { //  对应的控制器。可以直接写控制器，也可以写控制器的名称
            }
        };
})
```

自定义指令的使用

```
<header-directive></header-directive>
<div header-directive></div>
<div class="header-directive"></div>
<!-- directive: header-directive -->
```

服务相关

service

直接向 this 上面添加属性和方法即可

```
.service('DataService', function () {
        var arr = [];
        this.addItem = function (item) {
            arr.push(item);
        };
        this.reduceItem = function (index) {
```

```
            arr.splice(index, 1);
        };
});
factory
返回对象
.factory('UtilFactory', function () {
        return {
            min: function (a, b) {
                return    a > b ? b : a;
            },
            max: function (a, b) {
                return a > b ? a : b;
            }
        }
})
value
变量的共享
.value('count', 20);
.value('commonFunc', function () {
        // ...
})
constant
常量的共享
可以注入到 config 中
.constant('PI', 3.14);
.constant('TITLE_COLOR', 'hotpink');
provider
provider 是 service factory value 的内部实现
provider 提供的服务可以注入到 config 中
.provider('Data', function () {
        this.func = function () {
            // ...
        }l
        this.$get = function () {
            return {
                a: 10,
                b: 20
            };
        }
});
.config(['DataProvider', function (DataProvider) {
```

```
}]);
.controller('Ctrl', ['$scope', 'Data', function ($scope, Data) {
    // ...
}])
```

AngularJS 模块

Angular-ui-router

模块使用时，首先在初始化 app 的时候添加模块名称

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

2. 编写 HTML 结构代码

```
<a ui-sref="home">首页</a>
<a ui-sref="cart">购物车</a>
<a ui-sref="mine">我的</a>

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

3. 编写路由规则

```
.config(['$stateProvider', '$urlRouterProvider', function ($stateProvider, $urlRouterProvider) {
    // 默认路由
    $urlRouterProvider
    .when('', '/home')
    .when('home', '/home/one');
    // 路由规则
    $stateProvder
    .state('home', {
        url: '/home',
        templateUrl: '',
        controller: 'HomeCtrl',
        controller: function () {
        }
    })
    .state('home.one', {
        url: '/one',
        templateUrl: '',
        controller: 'HomeCtrl'
    })
    .state('goodDetail', {
        url: '/goodDetail:id',
        templateUrl: '',
        controller: ''
    });
}])
```

4. 路由跳转以及参数的传递

```
<li ui-sref="goodDetail({id: 'abc'})">路由传参</li>
$scope.sendValue = function () {
$state.go('goodDetal', {
id: 'abc'
});
};
```

5. 参数的接收

```
.controller('GoodDetailCtrl', ['$scope', '$stateParams', function ($scope, $stateParams) {
    $scope.id = $stateParams.id;
}])
```

6. 内置服务

$stateProvider 状态提供者，负责各种路由规则的设定
$urlRouterProvider 设置默认路由
$state 路由跳转使用
$stateParams 用于接收路由传递的参数

AngularCSS

引入模块

```
var app = angular.module('App', ['angularCSS']);
```

2. 在控制器中使用$css 使用

```
.controller('Ctrl', ['$scope', '$css', function ($scope, $css) {
    // 添加和移除 CSS 样式
    $css.bind({
        href: ''
    }, $scope);
    // 添加 CSS 样式
    $css.add('url');
    // 移除 CSS 样式
    $css.remove('url');
    // 移除全部添加的 CSS 样式
    $css.removeAll();
}])
```

内置服务

$css

AngularAMD

angular-ui-router 配置路由规则的时候，state 中需要修改

```
.config(['$stateProvider', '$urlRouterProvider', function ($stateProvider, $urlRouterProvider) {
    // 默认路由
    $urlRouterProvider
    .when('', '/home')
    .when('home', '/home/one');
```

```
        // 路由规则
        $stateProvder
        .state('home', angularAMD.route({
            url: '/home',
            templateUrl: 'html url',
            controllerUrl: 'controller url'
        }))
}])
```

2. 对应的控制器文件 home.js，返回的只是数组而已

```
define(['app'], function (app) {
        return ['$scope', function ($scope) {
        }];
});
```

RequireJS

引入文件，并设置主模块

```
<script type="text/javascript" src="libs/require.js" data-main="app" async="true" defer></script>
```

2. 配置操作

```
requirejs.config({
        baseUrl: '', // 基础路径
        paths: { // 名称和路径的映射
            jquery: 'libs/jquery',
            angularUIRouter: 'libs/angular-ui-router'
            baiduTemplate: 'libs/baiduTemplate',
            domReady: 'libs/domReady'
        },
        shim: {
            baiduTemplate: { // 解决不支持 requirejs 模块的问题
                exports: 'baidu.template'
            },
            angularUIRouter: { // 解决前置依赖的问题
                deps: ['angular']
            }
        }
});
```

3. 模块的引入操作

```
// 参数 1：所依赖的模块
// 参数 2：回调函数，形参需要和第一个参数中的数组一一对应。
// 注意：有些模块不需要对外提供接口，建议写在数组的后面部分
require(['angular', 'domReady'], function (angular, domReady) {
        domReady(function () {
            angular.bootstrap(document, ['app']);
```

```
    });
});
```

4. 定义模块

```
// 参数 1：模块的名称，如果存在，则不可修改，可选
// 参数 2：当前模块所依赖的模块，可选
// 参数 3：模块实现体，函数，需要有返回值
define('jquery', [], function () {
    return angular.module('axfAPP', ['ui.router', 'angularCSS']);
});
```

Gulp

1. 电脑上全局安装

```
$ npm install -g gulp
```

2. 初始化 package.json 文件（进入项目根目录）

```
$ npm init
```

3. 在项目中安装 gulp

```
$ npm install --save-dev gulp
```

4. 在项目中安装 gulp 插件

```
$ npm install --save-dev gulp-htmlmin
$ npm install --save-dev gulp-minify-css
$ npm install --save-dev gulp-uglify
```

5. 创建 gulpefile.js 文件，并添加代码

```
// 引入对应的对象
var gulp = require('gulp'),
    htmlmin = require('gulp-htmlmin'),
    minifycss = require('gulp-minify-css'),
    uglify = require('gulp-uglify');
// 创建任务
gulp.task('html', function () {
});
gulp.task('css', function () {
});
gulp.task('js', function () {
     gulp.src('')
         .pipe(uglify())
         .pipe(gulp.dest(''));
});
gulp.task('copy', function () {
    gulp.src('imgs/*')
         .pipe(gulp.dest('app/imgs'));
});
// 默认任务，执行默认任务时，自动执行 html/css/js/copy 等任务
```

```
gulp.task('default', ['html', 'css', 'js', 'copy']);
```

6. 执行命令，进行打包压缩

```
$ gulp
```

JSSDK 的开发

1. 配置安全域名

weixindemo.applinzi.com

2. 代码中的配置

```
wx.config({
    debug: true, // 开启调试模式,调用的所有 api 的返回值会在客户端 alert 出来，若要查看传入的参数，可以在 pc 端打开，参数信息会通过 log 打出，仅在 pc 端时才会打印。
    appId: '', // 必填，公众号的唯一标识
    timestamp: , // 必填，生成签名的时间戳
    nonceStr: '', // 必填，生成签名的随机串
    signature: '',// 必填，签名，见附录 1
    jsApiList: [] // 必填，需要使用的 JS 接口列表，所有 JS 接口列表见附录 2
});
```

3. ready 方法的使用

```
wx.ready(function () {
    // 页面加载进来之后，直接运行的代码放在这里。
    // 例如：获取定位信息
});
```

4. 其他方法的使用

拍照或从手机相册选取照片

```
wx.chooseImage({
    count: 1, // 默认 9
    sizeType: ['original', 'compressed'], // 可以指定是原图还是压缩图，默认二者都有
    sourceType: ['album', 'camera'], // 可以指定来源是相册还是相机，默认二者都有
    success: function (res) {
        var localIds = res.localIds; // 返回选定照片的本地 ID 列表，localId 可以作为 img 标签的 src 属性显示图片
    }
});
```

打开地图，定位到指定的位置

```
wx.openLocation({
    latitude: 0, // 纬度，浮点数，范围为 90 ~ -90
    longitude: 0, // 经度，浮点数，范围为 180 ~ -180。
    name: '', // 位置名
    address: '', // 地址详情说明
    scale: 1, // 地图缩放级别,整形值,范围从 1~28。默认为最大
    infoUrl: '' // 在查看位置界面底部显示的超链接,可点击跳转
});
```