**Angular开发规范**

目录

[1.应用程序结构 4](#_Toc425438045)

[总指南 4](#_Toc425438046)

[应用程序结构LIFT原则 4](#_Toc425438047)

[按功能划分文件夹结构 4](#_Toc425438048)

[单一职责 4](#_Toc425438049)

[IIFE（避免变量冲突） 4](#_Toc425438050)

[模块 6](#_Toc425438051)

[配置 6](#_Toc425438052)

[路由 6](#_Toc425438053)

[Layout 7](#_Toc425438054)

[2.命名规范 7](#_Toc425438055)

[原则 7](#_Toc425438056)

[文件目录命名 7](#_Toc425438057)

[测试文件命名 7](#_Toc425438058)

[代码级命名 7](#_Toc425438059)

[3.异常处理 9](#_Toc425438060)

[修饰符 9](#_Toc425438061)

[异常捕获器 10](#_Toc425438062)

[路由错误 11](#_Toc425438063)

[4.Data Services 12](#_Toc425438064)

[独立的数据调用 12](#_Toc425438065)

[数据调用返回一个Promise 14](#_Toc425438066)

[5.常量 15](#_Toc425438067)

[供应全局变量 15](#_Toc425438068)

[常量文件的必要性 15](#_Toc425438069)

[实例—目录结构（重要） 16](#_Toc425438070)

[实例—代码规范（重要） 17](#_Toc425438071)

//参看详细文档，目录如下

[17.启动逻辑 21](#_Toc425438072)

[配置 21](#_Toc425438073)

[运行代码块 21](#_Toc425438074)

[10.手动依赖注入 22](#_Toc425438075)

[16.模块化 24](#_Toc425438076)

[许多小的、独立的模块 24](#_Toc425438077)

[创建一个App Module 24](#_Toc425438078)

[保持App Module的精简 24](#_Toc425438079)

[功能区域就是模块 24](#_Toc425438080)

[可重用的块就是模块 25](#_Toc425438081)

[模块依赖 25](#_Toc425438082)

[3.Modules 26](#_Toc425438083)

[避免命名冲突（命名唯一） 26](#_Toc425438084)

[定义(aka Setters)（直接定义，不用变量） 26](#_Toc425438085)

[Getters（使用getter的链式语法） 27](#_Toc425438086)

[Setting vs Getting（只能设置一次） 27](#_Toc425438087)

[命名vs 匿名（回调函数使用命名函数） 27](#_Toc425438088)

[4.Controllers 28](#_Toc425438089)

[controllerAs在View中的语法 28](#_Toc425438090)

[controllerAs在controller中的语法 29](#_Toc425438091)

[controllerAs with vm 29](#_Toc425438092)

[可绑定成员放到顶部 30](#_Toc425438093)

[函数声明隐藏实现细节 33](#_Toc425438094)

[把Controller中的逻辑延迟到Service中 35](#_Toc425438095)

[保持Controller的专一性 40](#_Toc425438096)

[分配Controller 40](#_Toc425438097)

[9.解决Controller的Promises 41](#_Toc425438098)

[Controller Activation Promises 41](#_Toc425438099)

[Route Resolve Promises 42](#_Toc425438100)

[8.Directives 45](#_Toc425438101)

[一个directive一个文件 45](#_Toc425438102)

[在directive中操作DOM 47](#_Toc425438103)

[提供一个唯一的Directive前缀 47](#_Toc425438104)

[限制元素和属性 47](#_Toc425438105)

[Directives和ControllerAs 49](#_Toc425438106)

[bindToController 50](#_Toc425438107)

[5.Services 51](#_Toc425438108)

[单例 51](#_Toc425438109)

[6.Factories 52](#_Toc425438110)

[单一职责 52](#_Toc425438111)

[单例 52](#_Toc425438112)

[可访问的成员放到顶部 53](#_Toc425438113)

[函数声明隐藏实现细节 55](#_Toc425438114)

[27.路由 58](#_Toc425438115)

[用AngularUI Router来做路由控制 58](#_Toc425438116)

[Define routes for views in the module 58](#_Toc425438117)

[29.Filters 59](#_Toc425438118)

[11.压缩和注释（Gulp或Grunt中使用ng-annotate） 59](#_Toc425438119)

[ng-annotate（智能添加注入对象，使压缩更安全） 59](#_Toc425438120)

[使用Gulp或Grunt结合ng-annotate 61](#_Toc425438121)

[18.Angular $包装服务（方便测试） 61](#_Toc425438122)

[$document和$window 61](#_Toc425438123)

[$timeout和$interval 62](#_Toc425438124)

[20.动画 62](#_Toc425438125)

[用法 62](#_Toc425438126)

[Sub Second 62](#_Toc425438127)

[animate.css 62](#_Toc425438128)

[19.测试（单元测试基础） 63](#_Toc425438129)

[用故事来编写测试 63](#_Toc425438130)

[测试库（Jasmine/Mocha） 64](#_Toc425438131)

[测试运行器（Karma） 64](#_Toc425438132)

[Stubbing和Spying（用Sinon） 64](#_Toc425438133)

[Headless Browser（不可视浏览器） 65](#_Toc425438134)

[代码分析 65](#_Toc425438135)

[对测试降低全局JSHint规则 65](#_Toc425438136)

[组织测试 66](#_Toc425438137)

[25.文件模板和片段（规范工具等） 67](#_Toc425438138)

[Sublime Text 67](#_Toc425438139)

[Visual Studio 67](#_Toc425438140)

[WebStorm 68](#_Toc425438141)

[Atom 68](#_Toc425438142)

[Brackets 69](#_Toc425438143)

[26.Yeoman Generator 69](#_Toc425438144)

[生成Angular起步应用 69](#_Toc425438145)

[21.注释 70](#_Toc425438146)

[jsDoc（文档） 70](#_Toc425438147)

[22.JSHint（分析JS代码） 71](#_Toc425438148)

[使用一个Options文件 71](#_Toc425438149)

[23.JSCS（检查代码风格） 73](#_Toc425438150)

[用一个Options文件 73](#_Toc425438151)

[28.任务自动化 75](#_Toc425438152)

[用Gulp或者Grunt来创建自动化任务 75](#_Toc425438153)

[30.Angular文档 76](#_Toc425438154)

[31.贡献 76](#_Toc425438155)

[过程 76](#_Toc425438156)

[32.许可 76](#_Toc425438157)

[Copyright 76](#_Toc425438158)

[(The MIT License) 76](#_Toc425438159)

# 1.[应用程序结构](https://github.com/johnpapa/angular-styleguide/blob/master/i18n/zh-CN.md#%E5%BA%94%E7%94%A8%E7%A8%8B%E5%BA%8F%E7%BB%93%E6%9E%84)

### 总指南

app的所有代码都在一个叫做app的根目录下，所有的内容都遵循一个功能一个文件，每一个controller、service、module、view都是独立的文件。第三方脚本存放在另外的根文件夹中（bower\_components、scripts、lib）。

### 应用程序结构LIFT原则

构建一个可以快速定位（Locate）代码、一目了然地识别（Identify）代码、拥有一个平直（Flattest）的结构、尽量（Try）坚持DRY（Don’t Repeat Yourself）的应用程序，其结构应该遵循这4项基本准则。

* 1. 轻松定位代码（L）
  2. 一眼识别代码（I）
  3. 平直的代码结构（层级不要太多）（F）
  4. 尽量保持不要写重复代码（T）

### 按功能划分文件夹结构

* 按照它们代表的功能来给创建的文件夹命名
* 当文件夹包含的文件超过7个，就考虑新建文件夹。
* 不要使用按类型划分文件夹结构（防止来回切换文件）

### [单一职责](https://github.com/johnpapa/angular-styleguide/blob/master/i18n/zh-CN.md#%E5%8D%95%E4%B8%80%E8%81%8C%E8%B4%A3)

* 一个文件只定义一个组件，便于后期扩展，协作开发

### [IIFE](https://github.com/johnpapa/angular-styleguide/blob/master/i18n/zh-CN.md#iife)（避免变量冲突）

* JavaScript闭包

把Angular组件包装到一个立即调用函数表达式中（IIFE）。

为什么？：把变量从全局作用域中删除了，这有助于防止变量和函数声明比预期在全局作用域中有更长的生命周期，也有助于避免变量冲突。

为什么？：当你的代码为了发布而压缩了并且被合并到同一个文件中时，可能会有很多变量发生冲突，使用了IIFE（给每个文件提供了一个独立的作用域），你就不用担心这个了。

/\* avoid \*/

// logger.js

angular

.module('app')

.factory('logger', logger);

// logger function会被当作一个全局变量

function logger() { }

// storage.js

angular

.module('app')

.factory('storage', storage);

// storage function会被当作一个全局变量

function storage() { }

/\*\*

\* recommended

\*

\* 再也不存在全局变量了

\*/

// logger.js

(function() {

'use strict';

angular

.module('app')

.factory('logger', logger);

function logger() { }

})();

// storage.js

(function() {

'use strict';

angular

.module('app')

.factory('storage', storage);

function storage() { }

})();

* 注：为了简洁起见，本指南余下的示例中将会省略IIFE语法。
* 注：IIFE阻止了测试代码访问私有成员（正则表达式、helper函数等），这对于自身测试是非常友好的。然而你可以把这些私有成员暴露到可访问成员中进行测试，例如把私有成员（正则表达式、helper函数等）放到factory或是constant中。（？？？？）

### 模块

* 当有很多的模块时，主模块文件命名成app.module.js，其它依赖模块以它们代表的内容来命名。例如，一个管理员模块命名成admin.module.js，它们各自的注册模块名字就是app和admin。

为什么？：给多模块的应用提供统一的方式，这也是为了扩展大型应用。

为什么？：对使用任务来自动化加载所有模块的定义（先）和其它所有的angular文件（后）提供了一种简单的方式。

### 配置

* 把一个模块的配置独立到它自己的文件中，以这个模块为基础命名。app模块的配置文件命名成app.config.js（或是config.js），admin.module.js的配置文件命名成admin.config.js。

为什么？：把配置从模块定义、组件和活跃代码中分离出来。

为什么？：为设置模块的配置提供了一个可识别的地方。

### 路由

* 把路由的配置独立到单独的文件。主模块的路由可能是app.route.js，admin模块的路由可能是admin.route.js。即使是在很小的应用中，我也喜欢把路由的配置从其余的配置中分离出来。

### Layout

把定义应用程序总体布局的组件放到layout文件夹中，如导航、内容区等等，方便复用。

# 2.[命名](https://github.com/johnpapa/angular-styleguide/blob/master/i18n/zh-CN.md#%E5%91%BD%E5%90%8D)规范

### 原则

* 遵循以描述组件功能，然后是类型（可选）的方式来给所有的组件提供统一的命名，我推荐的做法是feature.type.js。

### 文件目录命名

* 按照业务划分文件，并以业务描述命名（稍后以SMG为例给出文件目录结构）。

### 测试文件命名

* 和文件命名差不多，带上一个spec后缀。

为什么？：为快速识别组件提供统一的方式。

为什么？：为[karma](http://karma-runner.github.io/)或是其它测试运行器提供模式匹配。

/\*\*

\* recommended

\*/

avengers.controller.spec.js

logger.service.spec.js

avengers.routes.spec.js

avenger-profile.directive.spec.js

### 代码级命名

* Controller命名（采用UpperCamelCase（每个单词首字母大写）的方式）
* 除controller外，其余代码命名遵循camel-casing（驼峰式，第一个单词首字母小写，后面单词首字母大写）方式。

/\* recommended \*/

// app.module.js

angular

.module('app', ['ngRoute']);

/\* recommended \*/

// someController.js

angular

.module('app')

.controller('SomeController', SomeController);

function SomeController() { }

/\* recommended \*/

// someFactory.js

angular

.module('app')

.factory('someFactory', someFactory);

function someFactory() { }

// avengers.controller.js

angular

.module

.controller('HeroAvengersController', HeroAvengersController);

function HeroAvengers(){ }

// logger.service.js

angular

.module

.factory('logger', logger);

function logger(){ }

// avenger-profile.directive.js

angular

.module

.directive('xxAvengerProfile', xxAvengerProfile);

// usage is <xx-avenger-profile> </xx-avenger-profile>

function xxAvengerProfile(){ }

# 3.[异常处理](https://github.com/johnpapa/angular-styleguide/blob/master/i18n/zh-CN.md#%E5%BC%82%E5%B8%B8%E5%A4%84%E7%90%86)

### 修饰符

###### [Style [Y110](https://github.com/johnpapa/angular-styleguide/blob/master/i18n/zh-CN.md#style-y110)]

* 使用一个[decorator](https://docs.angularjs.org/api/auto/service/$provide#decorator)，在配置的时候用[$provide](https://docs.angularjs.org/api/auto/service/$provide)服务，当发生异常时，在[$exceptionHandler](https://docs.angularjs.org/api/ng/service/$exceptionHandler)服务中执行自定义的处理方法。

为什么？：在开发时和运行时提供了一种统一的方式来处理未被捕获的Angular异常。

注：另一个选项是用来覆盖service的，这个可以代替decorator，这是一个非常nice的选项，但是如果你想保持默认行为，那么推荐你扩展一个decorator。

/\* recommended \*/

angular

.module('blocks.exception')

.config(exceptionConfig);

exceptionConfig.$inject = ['$provide'];

function exceptionConfig($provide) {

$provide.decorator('$exceptionHandler', extendExceptionHandler);

}

extendExceptionHandler.$inject = ['$delegate', 'toastr'];

function extendExceptionHandler($delegate, toastr) {

return function(exception, cause) {

$delegate(exception, cause);

var errorData = {

exception: exception,

cause: cause

};

/\*\*

\* Could add the error to a service's collection,

\* add errors to $rootScope, log errors to remote web server,

\* or log locally. Or throw hard. It is entirely up to you.

\* throw exception;

\*/

toastr.error(exception.msg, errorData);

};

}

### 异常捕获器

###### [Style [Y111](https://github.com/johnpapa/angular-styleguide/blob/master/i18n/zh-CN.md#style-y111)]

* 创建一个暴露了一个接口的factory来捕获异常并以合适方式处理异常。

为什么？：提供了一个统一的方法来捕获代码中抛出的异常。

注：异常捕获器对特殊异常的捕获和反应是非常友好的，例如，使用XHR从远程服务获取数据时，你想要捕获所有异常并做出不同的反应。

/\* recommended \*/

angular

.module('blocks.exception')

.factory('exception', exception);

exception.$inject = ['logger'];

function exception(logger) {

var service = {

catcher: catcher

};

return service;

function catcher(message) {

return function(reason) {

logger.error(message, reason);

};

}

}

### 路由错误

###### [Style [Y112](https://github.com/johnpapa/angular-styleguide/blob/master/i18n/zh-CN.md#style-y112)]

* 用[$routeChangeError](https://docs.angularjs.org/api/ngRoute/service/$route#$routeChangeError)来处理并打印出所有的路由错误信息。

为什么？：提供一个统一的方式来处理所有的路由错误。

为什么？：当一个路由发生错误的时候，可以给展示一个提示信息，提高用户体验。

/\* recommended \*/

var handlingRouteChangeError = false;

function handleRoutingErrors() {

/\*\*

\* Route cancellation:

\* On routing error, go to the dashboard.

\* Provide an exit clause if it tries to do it twice.

\*/

$rootScope.$on('$routeChangeError',

function(event, current, previous, rejection) {

if (handlingRouteChangeError) { return; }

handlingRouteChangeError = true;

var destination = (current && (current.title ||

current.name || current.loadedTemplateUrl)) ||

'unknown target';

var msg = 'Error routing to ' + destination + '. ' +

(rejection.msg || '');

/\*\*

\* Optionally log using a custom service or $log.

\* (Don't forget to inject custom service)

\*/

logger.warning(msg, [current]);

/\*\*

\* On routing error, go to another route/state.

\*/

$location.path('/');

}

);

}

# 4.[Data Services](https://github.com/johnpapa/angular-styleguide/blob/master/i18n/zh-CN.md" \l "data-services)

### 独立的数据调用

###### [Style [Y060](https://github.com/johnpapa/angular-styleguide/blob/master/i18n/zh-CN.md#style-y060)]

* 把进行数据操作和数据交互的逻辑放到factory中，数据服务负责XHR请求、本地存储、内存存储和其它任何数据操作。

为什么？：controller的作用是查看视图和收集视图的信息，它不应该关心如何取得数据，只需要知道哪里需要用到数据。把取数据的逻辑放到数据服务中能够让controller更简单、更专注于对view的控制。

为什么？：方便测试。

为什么？：数据服务的实现可能有非常明确的代码来处理数据仓库，这可能包含headers、如何与数据交互或是其它service，例如$http。把逻辑封装到单独的数据服务中，这隐藏了外部调用者（例如controller）对数据的直接操作，这样更加容易执行变更。

/\* recommended \*/

// dataservice factory

angular

.module('app.core')

.factory('dataservice', dataservice);

dataservice.$inject = ['$http', 'logger'];

function dataservice($http, logger) {

return {

getAvengers: getAvengers

};

function getAvengers() {

return $http.get('/api/maa')

.then(getAvengersComplete)

.catch(getAvengersFailed);

function getAvengersComplete(response) {

return response.data.results;

}

function getAvengersFailed(error) {

logger.error('XHR Failed for getAvengers.' + error.data);

}

}

}

注意：数据服务被调用时（例如controller），隐藏调用的直接行为，如下所示。

/\* recommended \*/

// controller calling the dataservice factory

angular

.module('app.avengers')

.controller('Avengers', Avengers);

Avengers.$inject = ['dataservice', 'logger'];

function Avengers(dataservice, logger) {

var vm = this;

vm.avengers = [];

activate();

function activate() {

return getAvengers().then(function() {

logger.info('Activated Avengers View');

});

}

function getAvengers() {

return dataservice.getAvengers()

.then(function(data) {

vm.avengers = data;

return vm.avengers;

});

}

}

### 数据调用返回一个Promise

###### [Style [Y061](https://github.com/johnpapa/angular-styleguide/blob/master/i18n/zh-CN.md#style-y061)]

* 就像$http一样，调用数据时返回一个promise，在你的调用函数中也返回一个promise。

为什么？：你可以把promise链接到一起，在数据调用完成并且resolve或是reject这个promise后采取进一步的行为。

/\* recommended \*/

activate();

function activate() {

/\*\*

\* Step 1

\* Ask the getAvengers function for the

\* avenger data and wait for the promise

\*/

return getAvengers().then(function() {

/\*\*

\* Step 4

\* Perform an action on resolve of final promise

\*/

logger.info('Activated Avengers View');

});

}

function getAvengers() {

/\*\*

\* Step 2

\* Ask the data service for the data and wait

\* for the promise

\*/

return dataservice.getAvengers()

.then(function(data) {

/\*\*

\* Step 3

\* set the data and resolve the promise

\*/

vm.avengers = data;

return vm.avengers;

});

}

# 5.[常量](https://github.com/johnpapa/angular-styleguide/blob/master/i18n/zh-CN.md#%E5%B8%B8%E9%87%8F)

### 供应全局变量

###### [Style [Y240](https://github.com/johnpapa/angular-styleguide/blob/master/i18n/zh-CN.md#style-y240)]

* 为供应库中的全局变量创建一个Angular常量。

为什么？：提供一种注入到供应库的方法，否则就是全局变量。通过让你更容易地了解你的组件之间的依赖关系来提高代码的可测试性。这还允许你模拟这些依赖关系，这是很有意义的。

// constants.js

/\* global toastr:false, moment:false \*/

(function() {

'use strict';

angular

.module('app.core')

.constant('toastr', toastr)

.constant('moment', moment);

})();

### 常量文件的必要性

###### [Style [Y241](https://github.com/johnpapa/angular-styleguide/blob/master/i18n/zh-CN.md#style-y241)]

* 对于一些不需要变动，也不需要从其它service中获取的值，使用常量定义，当一些常量只是在一个模块中使用但是有可能会在其它应用中使用的话，把它们写到一个以当前的模块命名的文件中。把常量集合到一起是非常有必要的，你可以把它们写到constants.js的文件中。

为什么？：一个可能变化的值，即使变动的很少，也会从service中重新被检索，因此你不需要修改源代码。例如，一个数据服务的url可以被放到一个常量中，但是更好的做法是把它放到一个web service中。

为什么？：常量可以被注入到任何angular组件中，包括providers。

为什么？：当一个应用程序被分割成很多可以在其它应用程序中复用的小模块时，每个独立的模块都应该可以操作它自己包含的相关常量。

// Constants used by the entire app

angular

.module('app.core')

.constant('moment', moment);

// Constants used only by the sales module

angular

.module('app.sales')

.constant('events', {

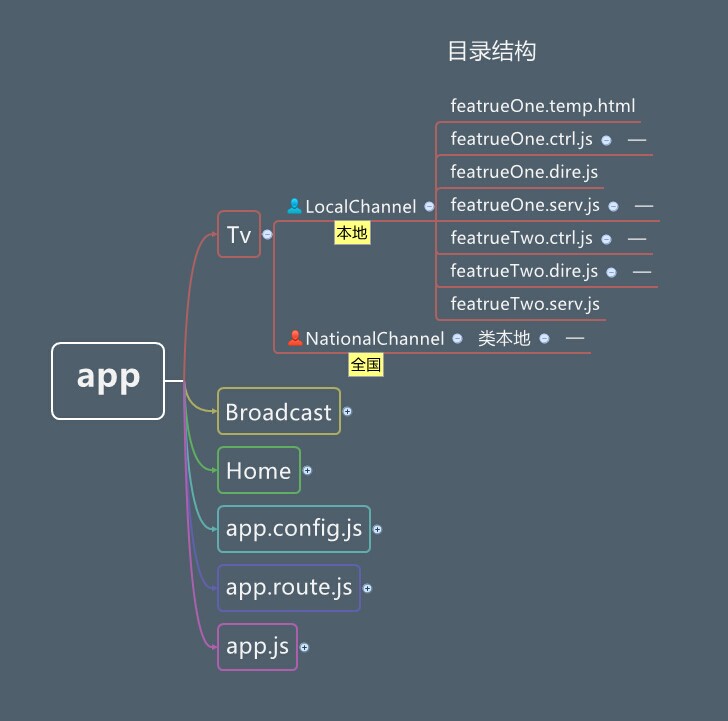
ORDER\_CREATED: 'event\_order\_created',

INVENTORY\_DEPLETED: 'event\_inventory\_depleted'

});

# 实例—目录结构（重要）

基于以上两点，以SMG项目为例进行规范说明。如下图所示，文件夹首字母大写，文件以“功能.类型”格式命名。



# 实例—代码规范（重要）

总原则：

* + 单一职责
  + IIFE（采用js闭包函数）
  + 包含成员放到顶部（建议方法、变量按照字母排序放置）
  + 回调函数使用命名函数，函数声明隐藏实现细节
  + 手动依赖注入
  + 模块细分，单一职责；功能区及复用块均写成模块

具体实例：

* Controller

/\* recommended \*/

(function() {

'use strict';

angular

.module('app')

.controller('Sessions', Sessions);

function Sessions(dataservice) {

var vm = this;

vm.gotoSession = gotoSession;

vm.refresh = refresh;

vm.search = search;

vm.sessions = [];

vm.title = 'Sessions';

// Controller中promise

activate();

////////////

function activate() {

return dataservice.getAvengers().then(function(data) {

vm.avengers = data;

return vm.avengers;

});

}

////////////

function gotoSession() {

/\* \*/

}

function refresh() {

/\* \*/

}

function search() {

/\* \*/

}

}

})();

* Directive

/\* recommended \*/

(function() {

'use strict';

Angular

.module('app')

.directive('myExample', myExample);

function myExample() {

var directive = {

restrict: 'EA',

templateUrl: 'app/feature/example.directive.html',

scope: {

max: '='

},

link: linkFunc,

controller : ExampleController,

controllerAs: 'vm',

bindToController: true // because the scope is isolated

};

return directive;

function linkFunc(scope, el, attr, ctrl) {

console.log('LINK: scope.min = %s \*\*\* should be undefined', scope.min);

console.log('LINK: scope.max = %s \*\*\* should be undefined', scope.max);

console.log('LINK: scope.vm.min = %s', scope.vm.min);

console.log('LINK: scope.vm.max = %s', scope.vm.max);

}

}

ExampleController.$inject = ['$scope'];

function ExampleController($scope) {

// Injecting $scope just for comparison

var vm = this;

vm.min = 3;

console.log('CTRL: $scope.vm.min = %s', $scope.vm.min);

console.log('CTRL: $scope.vm.max = %s', $scope.vm.max);

console.log('CTRL: vm.min = %s', vm.min);

console.log('CTRL: vm.max = %s', vm.max);

}

})();

* route-config.js

/\* recommended \*/

// route-config.js

(function() {

'use strict';

angular

.module('app')

.config(config);

function config ($routeProvider) {

$routeProvider

.when('/avengers', {

templateUrl: 'avengers.html',

controller: 'AvengersController',

controllerAs: 'vm',

resolve: {

moviesPrepService: moviePrepService

}

});

}

moviePrepService.$inject = ['movieService'];

function moviePrepService(movieService) {

return movieService.getMovies();

}

})();

* Factory(服务类以此为标准)

// logger.js

(function() {

'use strict';

angular

.module('app')

.factory('logger', logger);

function logger() { }

})();