無料電子ブック

学習

AngularJS

Free unaffiliated eBook created from
Stack Overflow contributors.

#angularjs

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: angularjs

It is an unofficial and free AngularJS ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official AngularJS.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

1: AngularJSをいめる

AngularJSは、リッチなクライアントサイドのアプリケーションをするためにされたWebアプリケーションフレームワークです。このドキュメントは、よりなAngular 2のであるAngular 1.xまたはAngular 2のStack Overflowドキュメントをしてください。

バージョン

バージョン	
1.6.5	2017-07-03
1.6.4	2017-03-31
1.6.3	2017-03-08
1.6.2	2017-02-07
1.5.11	2017-01-13
1.6.1	2016-12-23
1.5.10	20161215
1.6.0	2016-12-08
1.6.0-rc.2	2016-11-24
1.5.9	2016-11-24
1.6.0-rc.1	2016-11-21
1.6.0-rc.0	2016-10-26
1.2.32	2016-10-11
1.4.13	2016-10-10
1.2.31	2016-10-10
1.5.8	2016-07-22
1.2.30	2016-07-21
1.5.7	2016-06-15
1.4.12	2016-06-15

バージョン	
1.5.6	2016527
1.4.11	2016527
1.5.5	2016-04-18
1.5.4	2016-04-14
1.5.3	2016-03-25
1.5.2	2016-03-19
1.4.10	2016-03-16
1.5.1	2016-03-16
1.5.0	2016-02-05
1.5.0-rc.2	2016-01-28
1.4.9	2016-01-21
1.5.0-rc.1	2016-01-16
1.5.0-rc.0	2015-12-09
1.4.8	2015-11-20
1.5.0-beta.2	2015-11-18
1.4.7	2015-09-30
1.3.20	2015-09-30
1.2.29	2015-09-30
1.5.0-β.1	2015-09-30
1.5.0-beta.0	2015-09-17
1.4.6	2015-09-17
1.3.19	2015-09-17
1.4.5	2015-08-28
1.3.18	2015-08-19
1.4.4	2015-08-13

バージョン	
1.4.3	2015-07-15
1.3.17	2015-07-07
1.4.2	2015-07-07
1.4.1	2015-06-16
1.3.16	2015-06-06
1.4.0	2015-05-27
1.4.0-rc.2	2015-05-12
1.4.0-rc.1	2015-04-24
1.4.0-rc.0	2015-04-10
1.3.15	2015-03-17
1.4.0-beta.6	2015-03-17
1.4.0-beta.5	2015-02-24
1.3.14	2015-02-24
1.4.0-beta.4	2015-02-09
1.3.13	2015-02-09
1.3.12	2015-02-03
1.4.0-β.3	2015-02-03
1.3.11	2015-01-27
1.4.0-beta.2	2015-01-27
1.4.0-beta.1	2015-01-20
1.3.10	2015-01-20
1.3.9	2015-01-15
1.4.0-beta.0	2015-01-14
1.3.8	20141219
1.2.28	20141216

バージョン	
1.3.7	2014-12-15
1.3.6	2014-12-09
1.3.5	2014-12-02
1.3.4	2014-11-25
1.2.27	2014-11-21
1.3.3	2014-11-18
1.3.2	2014-11-07
1.3.1	2014-10-31
1.3.0	2014-10-14
1.3.0-rc.5	2014-10-09
1.2.26	2014-10-03
1.3.0-rc.4	2014-10-02
1.3.0-rc.3	2014-09-24
1.2.25	2014-09-17
1.3.0-rc.2	2014-09-17
1.2.24	2014-09-10
1.3.0-rc.1	2014-09-10
1.3.0-rc.0	2014-08-30
1.2.23	2014-08-23
1.3.0-β.19	2014-08-23
1.2.22	2014-08-12
1.3.0-β.18	2014-08-12
1.2.21	2014-07-25
1.3.0-β.17	2014-07-25
1.3.0-β.16	2014-07-18

バージョン	
1.2.20	2014-07-11
1.3.0-β.15	2014-07-11
1.2.19	2014-07-01
1.3.0-β.14	2014-07-01
1.3.0-β.13	2014616
1.3.0-β.12	2014-06-14
1.2.18	2014-06-14
1.3.0-beta.11	201466
1.2.17	201466
1.3.0-beta.10	2014-05-24
1.3.0-β.9	2014-05-17
1.3.0-beta.8	2014-05-09
1.3.0-beta.7	2014-04-26
1.3.0-beta.6	2014-04-22
1.2.16	2014-04-04
1.3.0-beta.5	2014-04-04
1.3.0-beta.4	2014-03-28
1.2.15	2014-03-22
1.3.0-β.3	2014-03-21
1.3.0-beta.2	2014-03-15
1.3.0-beta.1	2014-03-08
1.2.14	2014-03-01
1.2.13	2014-02-15
1.2.12	2014-02-08
1.2.11	2014-02-03

バージョン	
1.2.10	2014-01-25
1.2.9	2014-01-15
1.2.8	2014-01-10
1.2.7	2014-01-03
1.2.6	20131220
1.2.5	2013-12-13
1.2.4	2013-12-06
1.2.3	2013-11-27
1.2.2	2013-11-22
1.2.1	2013-11-15
1.2.0	2013-11-08
1.2.0-rc.3	20131014
1.2.0-rc.2	2013-09-04
1.0.8	2013-08-22
1.2.0rc1	2013-08-13
1.0.7	2013-05-22
1.1.5	2013-05-22
1.0.6	2013-04-04
1.1.4	2013-04-04
1.0.5	2013-02-20
1.1.3	2013-02-20
1.0.4	2013-01-23
1.1.2	2013-01-23
1.1.1	2012-11-27
1.0.3	2012-11-27

バージョン	
1.1.0	2012-09-04
1.0.2	2012-09-04
1.0.1	2012-06-25
1.0.0	2012-06-14
v1.0.0rc12	2012-06-12
v1.0.0rc11	2012-06-11
v1.0.0rc10	2012-05-24
v1.0.0rc9	2012-05-15
v1.0.0rc8	2012-05-07
v1.0.0rc7	2012-05-01
v1.0.0rc6	2012-04-21
v1.0.0rc5	2012-04-12
v1.0.0rc4	2012-04-05
v1.0.0rc3	2012-03-30
v1.0.0rc2	2012-03-21
g3-v1.0.0rc1	2012-03-14
g3-v1.0.0-rc2	2012-03-16
1.0.0rc1	2012-03-14
0.10.6	2012-01-17
0.10.5	2011118
0.10.4	2011-10-23
0.10.3	2011-10-14
0.10.2	2011108
0.10.1	2011-09-09
0.10.0	2011-09-02

バージョン	
0.9.19	2011-08-21
0.9.18	2011-07-30
0.9.17	2011-06-30
0.9.16	20110608
0.9.15	2011-04-12
0.9.14	2011-04-01
0.9.13	2011-03-14
0.9.12	2011-03-04
0.9.11	2011-02-09
0.9.10	2011-01-27
0.9.9	2011-01-14
0.9.7	2010-12-11
0.9.6	2010-12-07
0.9.5	2010-11-25
0.9.4	2010-11-19
0.9.3	2010-11-11
0.9.2	2010-11-03
0.9.1	2010-10-27
0.9.0	2010-10-21

Examples

しいHTMLファイルをし、のコンテンツをりけます。

```
<!DOCTYPE html>
<html ng-app>
<head>
  <title>Hello, Angular</title>
  <script src="https://code.angularjs.org/1.5.8/angular.min.js"></script>
</head>
<body ng-init="name='World'">
```

```
  <label>Name</label>
  <input ng-model="name" />
  <span>Hello, {{ name }}!</span>
  <p ng-bind="name"></p>
</body>
</html>
```

ライブデモ

ブラウザでファイルをくと、フィールドのに`Hello, World!`というテキストがされます`Hello, World!`。のをすると、ページをすることなく、テキストがリアルタイムでされます。

---

1. コンテンツネットワークからAngularフレームワークをロードします。

   ```
   <script src="https://code.angularjs.org/1.5.8/angular.min.js"></script>
   ```

2. `ng-app`ディレクティブをしてHTMLドキュメントをAngularアプリケーションとしてする

   ```
   <html ng-app>
   ```

3. `ng-init`をして`name`をする

   ```
   <body ng-init=" name = 'World' ">
   ```

   *ng-init*は、デモンストレーションとテストのでのみされるべきであることにしてください。のアプリケーションをする、コントローラはデータをするがあります。

4. モデルからHTMLコントロールのビューにデータをバインドします。 `ng-model name`プロパティに`<input>`をバインドする

   ```
   <input ng-model="name" />
   ```

5. `{{ }}`をしてモデルからコンテンツをする

   ```
   <span>Hello, {{ name }}</span>
   ```

6. `name`プロパティをバインドするのは、ハンドルバー"`{{ }}`"わりに`ng-bind`を`ng-bind`ことです

   ```
    <span ng-bind="name"></span>
   ```

の3つのステップは、 データバインディングをします。にえられたはモデルをし、ビューにされます 。

ハンドルバーと`ng-bind`にはいがあります。ハンドルバーをすると、がされるデータがロードされるにページがロードされるので、の`Hello, {{name}}`れることがあります。、 `ng-bind`を`ng-bind`、されました。わりに、ハンドルバーがコンパイルされるにされるのをぐために、 `ng-cloak`ディレ

クティブをすることもできます。

すべてのなAngularをする

のは、1つのファイルのなAngularJSをしています。

```
<!DOCTYPE html>
<html ng-app="myDemoApp">
  <head>
    <style>.started { background: gold; }</style>
    <script src="https://code.angularjs.org/1.5.8/angular.min.js"></script>
    <script>
      function MyDataService() {
        return {
          getWorlds: function getWorlds() {
            return ["this world", "another world"];
          }
        };
      }

      function DemoController(worldsService) {
        var vm = this;
        vm.messages = worldsService.getWorlds().map(function(w) {
          return "Hello, " + w + "!";
        });
      }

      function startup($rootScope, $window) {
        $window.alert("Hello, user! Loading worlds...");
        $rootScope.hasStarted = true;
      }

      angular.module("myDemoApp", [/* module dependencies go here */])
        .service("worldsService", [MyDataService])
        .controller("demoController", ["worldsService", DemoController])
        .config(function() {
          console.log('configuring application');
        })
        .run(["$rootScope", "$window", startup]);
    </script>
  </head>
  <body ng-class="{ 'started': hasStarted }" ng-cloak>
    <div ng-controller="demoController as vm">
      <ul>
        <li ng-repeat="msg in vm.messages">{{ msg }}</li>
      </ul>
    </div>
  </body>
</html>
```

ファイルのすべてのについてにします。

ライブデモ

1. `ng-app="myDemoApp"` 、アプリケーションをブートストラップし、DOMが`"myDemoApp"`というのの`angular.module`によってされるようにするngAppディレクティブ 。
2. `<script src="angular.min.js">`はAngularJSライブラリをブートストラップするのステップで

す。

にす3つの `MyDataService` 、 `DemoController` 、 および`startup` がされています。

3. で2のとしてされた`angular.module(...)`は、しいモジュールをします。このは、モジュールののリストをするためにされます。このでは、 `module(...)`のをびしてします。

4. `.service(...)`はサービスをし、のためのモジュールをします。

5. `.controller(...)`はコントローラをし、のためのモジュールをします。

6. `.config(...)`このメソッドをして、モジュールのロードになをします。

7. `.run(...)`は、にコードがされ 、のをパラメータとして`.run(...)`ようにします。このメソッドをして、インジェクタがすべてのモジュールのロードをしたときにされるをします。

    - のは、 `startup`がみみの`$rootScope`サービスをとしてするがあることをAngularにらせるものです。
    - 2のは、 `startup`がみみの`$window`サービスをとしてするがあることをAngularにらせるものです。
    - ののである`startup`は`startup`にされるのです。

8. `ng-class`あるngClassディレクティブのする`class` 、このでは`hasStarted$rootScope`

9. `ng-cloak`は、 Angularがアプリケーションをにロードするに、レンダリングされていないAngular htmlテンプレートえば、 "`{{ msg }}`"をにしないようにするです。

10. `ng-controller`は、ののしいコントローラをインスタンスしてDOMのそのをするようAngularにするです。

11. `ng-repeat`は、コレクションにしてAngularをさせ、アイテムのDOMテンプレートをするです。

12. `{{ msg }}`ショーケース オンスポットスコープはののレンダリングを、

スコープの

Angularは、HTMLをしてWebページとのJavascriptをしてロジックをするので、 **ng- app** 、 **ng-controller** 、 **ng-if** 、 **ng-repeat**などのビルトインディレクティブをしてWebページをにできます。 。しい**controllerAs**をすると、Angularユーザーのユーザーは、 `$scope`をするわりに、とデータをコントローラにでき`$scope`。

しかし、かれかれ、この`$scope`ものがであるかをすることがです。それはいくつかのをつことがであるので、でされけるでしょう。

いニュースは、シンプルでなコンセプトだということです。

をするとき

```
<div ng-app="myApp">
 <h1>Hello {{ name }}</h1>
</div>
```

はどこにんでいますか

えは、Angularが$rootScopeオブジェクトをするということです。これはにのJavascriptオブジェクトなので、 **name**は$rootScopeオブジェクトのプロパティです。

```
angular.module("myApp", [])
  .run(function($rootScope) {
    $rootScope.name = "World!";
  });
```

Javascriptのグロ―バルスコ―プとじように、グロ―バルスコ―プや$rootScopeアイテムをすることは、はあまりいことではありません。

もちろん、ほとんどの、コントロ―ラをし、なをそのコントロ―ラにみみます。しかし、コントロ―ラをすると、Angularはそれをにして、そのコントロ―ラの$scopeオブジェクトをします。これはロ―カルスコ―プとばれることもあります 。

したがって、のコントロ―ラをします。

```
<div ng-app="myApp">
  <div ng-controller="MyController">
    <h1>Hello {{ name }}</h1>
  </div>
</div>
```

$scopeパラメ―タでロ―カルスコ―プにアクセスできるようになります。

```
angular.module("myApp", [])
  .controller("MyController", function($scope) {
    $scope.name = "Mr Local!";
  });
```

$scopeパラメ―タをたないコントロ―ラはらかのでそれをとしないかもしれません。ただし、controllerAsをしても、ロ―カルスコ―プがすることをすることがです。

$scopeはJavaScriptオブジェクトなので、Angularは$rootScopeからプロトタイプにするようにのようにします。あなたがすることができるように、のスコ―プがするがあります。たとえば、コントロ―ラにモデルをし、コントロ―ラのスコ―プに$scope.modelとしてアタッチすることができます。

その、プロトタイプチェ―ンをして、コントロ―ラは$scope.modelしてそのじモデルにロ―カルにアクセスできます。

はAngularがバックグラウンドでそのをやっているので、これははではありません。しかし、`$scope`することはAngularのみをるでなステップです。

もなのあるのある**Hello World**。

Angular 1はDOMコンパイラのにあります。テンプレートとしても、のWebペーージとしてもHTMLをして、アプリケーションをコンパイルすることができます。

`{{ }}`ハンドルバーのスタイルをして、ペーージのをとしてうようにAngularにできます。ののものは、のようにコンパイルされます

```
{{ 'Hello' + 'World' }}
```

これはされます

```
HelloWorld
```

ng-app

`ng-app` ディレクティブをしてDOMのどのをマスターテンプレートとしてうかをAngularにえます。ディレクティブは、アングルテンプレートコンパイラがするをっているカスタムまたはです。すぐng-appディレクティブをしましょう

```
<html>
  <head>
    <script src="/angular.js"></script>
  </head>
  <body ng-app>
    {{ 'Hello' + 'World' }}
  </body>
</html>
```

は、bodyにルートテンプレートとげました。そののかがコンパイルされます。

ディレクティブはコンパイラディレクティブです。 Angular DOMコンパイラのをします。 **Misko**、のは、りのをし、なぜこれがあります

> 「Webアプリケーションのためにされたwebブラウザとはか

り、しいHTMLのとをし、それらをアプリケーションにコンパイルします。 `ng-app`はにコンパイラをにするです。そののには、

- `ng-click`クリックハンドラをし、
- きでを`ng-hide`にする`ng-hide`
- `<form>` のHTMLフォームにのをします。

Angularには、もなタスクをするための100のみみがあります。たちもくことができ、これらはみ

みのとじようにわれます。

のディレクティブのからAngularアプリケーションをし、HTMLとにします。

の

ミニネーションとはですか

これは、をすることなく、ソースコードからなをすべてするです。

コントローラのにのをすると、ファイルをしたで、それがをすことになります。

コントローラ

```
var app = angular.module('mainApp', []);
app.controller('FirstController', function($scope) {
    $scope.name= 'Hello World !';
});
```

minification toolをした、のようにされます。

```
var app=angular.module("mainApp",[]);app.controller("FirstController",function(e){e.name=
'Hello World !'})
```

ここで、minificationはなスペースと$ scopeをコードからしました。だからたちはこのミニコードをすると、もされません。 $ scopeはコントローラとビューのでなであるため、さな 'e'できえられます。したがって、アプリケーションをすると、なプロバイダの 'e'エラーがします。

ミニネイションセーフであるサービスをしてコードにをけるには、2つのがあります。

インライン

```
var app = angular.module('mainApp', []);
app.controller('FirstController', ['$scope', function($scope) {
    $scope.message = 'Hello World !';
}]);
```

$ injectプロパティ

```
FirstController.$inject = ['$scope'];
var FirstController = function($scope) {
    $scope.message = 'Hello World !';
}

var app = angular.module('mainApp', []);
app.controller('FirstController', FirstController);
```

、このコードは

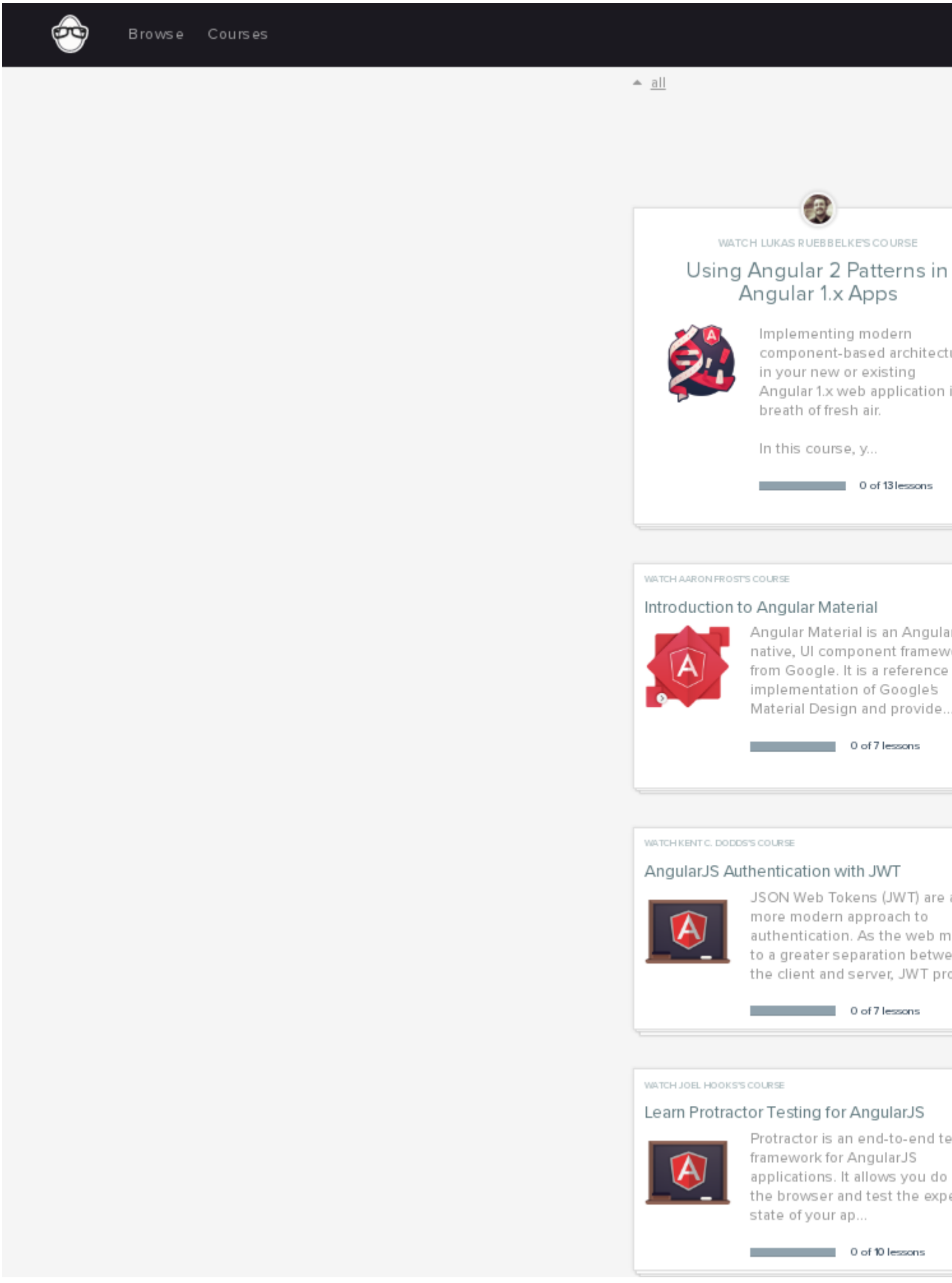
```
var
```

```
app=angular.module("mainApp",[]);app.controller("FirstController",["$scope",function(a){a.message="Hel
World !"}]);
```

ここでは、angleは 'a'を$ scopeとみなし、を 'Hello World'とします。

AngularJSビデオチュートリアル

egghead.ioには、AngularJSフレームワークにするくのれたビデオチュートリアルがあります



- https://egghead.io/courses/angularjs-app-from-scratch-getting-started

- https://egghead.io/courses/angularjs-application-architecture
- https://egghead.io/courses/angular-material-introduction
- https://egghead.io/courses/building-an-angular-1-x-ionic-application
- https://egghead.io/courses/angular-and-webpack-for-modular-applications
- https://egghead.io/courses/angularjs-authentication-with-jwt
- https://egghead.io/courses/angularjs-data-modeling
- https://egghead.io/courses/angular-automation-with-gulp
- https://egghead.io/courses/learn-protractor-testing-for-angularjs
- https://egghead.io/courses/ionic-quickstart-for-windows
- https://egghead.io/courses/build-angular-1-x-apps-with-redux
- https://egghead.io/courses/using-angular-2-patterns-in-angular-1-x-apps

オンラインでAngularJSをいめるをむ https://riptutorial.com/ja/angularjs/topic/295/angularjsをいめる

2: $ httpリクエスト

Examples

コントローラので$ httpをう

$httpサービスは、HTTPリクエストをしてをすです。

な

```
// Simple GET request example:
$http({
  method: 'GET',
  url: '/someUrl'
}).then(function successCallback(response) {
    // this callback will be called asynchronously
    // when the response is available
  }, function errorCallback(response) {
    // called asynchronously if an error occurs
    // or server returns response with an error status.
  });
```

コントローラーでの

```
appName.controller('controllerName',
    ['$http', function($http){

    // Simple GET request example:
    $http({
        method: 'GET',
        url: '/someUrl'
    }).then(function successCallback(response) {
        // this callback will be called asynchronously
        // when the response is available
    }, function errorCallback(response) {
        // called asynchronously if an error occurs
        // or server returns response with an error status.
    });
}])
```

ショートカットメソッド

$httpサービスにもショートカットがあります。 ここでhttpメソッドについてむ

```
$http.get('/someUrl', config).then(successCallback, errorCallback);
$http.post('/someUrl', data, config).then(successCallback, errorCallback);
```

ショートカットメソッド

- $ http.get
- $ http.head

- $ http.post
- $ http.put
- $ http.delete
- $ http.jsonp
- $ http.patch

サービスで$ httpリクエストをする

HTTPリクエストは、すべてのWebアプリでりしされるため、のリクエストごとにメソッドをし、アプリののでることをおめします。

`httpRequestsService.js`する

httpRequestsService.js

```
appName.service('httpRequestsService', function($q, $http){

    return {
        // function that performs a basic get request
        getName: function(){
            // make sure $http is injected
            return $http.get("/someAPI/names")
                .then(function(response) {
                    // return the result as a promise
                    return response;
                }, function(response) {
                    // defer the promise
                    return $q.reject(response.data);
                });
        },

        // add functions for other requests made by your app
        addName: function(){
            // some code...
        }
    }
})
```

のサービスは、サービスのgetをします。これは、サービスがされているすべてのコントローラでできます。

```
appName.controller('controllerName',
    ['httpRequestsService', function(httpRequestsService){

        // we injected httpRequestsService service on this controller
        // that made the getName() function available to use.
        httpRequestsService.getName()
            .then(function(response){
                // success
            }, function(error){
                // do something with the error
            })
    }])
```

このアプローチをして、いつでも**httpRequestsService.js**をのコントローラでできます。

$ http リクエストのタイミング

$ httpにはサーバーによってなるがですが、いくつかはミリかかりますが、いくつかはかかるもあります。くの、からデータをするのになが です。レスポンスがのであるとして、のをえてみましょう

った

```
$scope.names = [];

$http({
    method: 'GET',
    url: '/someURL'
}).then(function successCallback(response) {
            $scope.names = response.data;
        },
        function errorCallback(response) {
            alert(response.status);
        });

alert("The first name is: " + $scope.names[0]);
```

$ httpのすぐにある`$scope.names[0]`アクセスすると、しばしばエラーがします。このコードは、サーバーからをけるにされます。

しい

```
$scope.names = [];

$scope.$watch('names', function(newVal, oldVal) {
    if(!(newVal.length == 0)) {
        alert("The first name is: " + $scope.names[0]);
    }
});

$http({
    method: 'GET',
    url: '/someURL'
}).then(function successCallback(response) {
            $scope.names = response.data;
        },
        function errorCallback(response) {
            alert(response.status);
        });
```

$ watchサービスをすると、をけったときにのみ`$scope.names`にアクセスします。は`$scope.names`がされていてもがびされるため、 `newVal.length`が0のであるかどうかをするがあります。してください。 `$scope.names`えられたは、watchをします。

オンラインで$ httpリクエストをむ https://riptutorial.com/ja/angularjs/topic/3620/--httpリクエスト

3: $ qサービスによるの

Examples

$ q.allをしてのをする

`$q.all`をして、のがにされ、されたデータをフェッチしたに`.then`メソッドをびすことができます。

JS

```
$scope.data = []

$q.all([
   $http.get("data.json"),
   $http.get("more-data.json"),
   ]).then(function(responses) {
     $scope.data = responses.map((resp) => resp.data);
   });
```

のコードがされる`$http.get`をするとき、ローカルのJSONファイルのデータの2を`get`なを、らはそれにするをし、アレイのすべてのはされたときに、 `.then`は、ののデータでまる`responses`。

データはテンプレートにされるようにマップされます

HTML

```
<ul>
   <li ng-repeat="d in data">
      <ul>
         <li ng-repeat="item in d">{{item.name}}: {{item.occupation}}</li>
      </ul>
   </li>
</ul>
```

JSON

```
[{
  "name": "alice",
  "occupation": "manager"
}, {
  "name": "bob",
  "occupation": "developer"
}]
```

$ qコンストラクタをしてをする

`$q`コンストラクタは、コールバックをしてをすAPIからをするためにされます。

$ qfunctionresolve、reject{...}

コンストラクタは、をまたはするためにされるである`resolve`と`reject` 2つのでびされるをけります。

1

```
function $timeout(fn, delay) {
    return = $q(function(resolve, reject) {
         setTimeout(function() {
              try {
                 let r = fn();
                 resolve(r);
              }
              catch (e) {
                 reject(e);
              }
         }, delay);
     };
}
```

のは、 WindowTimers.setTimeout APIからをします 。 AngularJSフレームワークは、こののよりなバージョンをします。については、 AngularJS $ timeoutサービスAPIリファレンスをしてください 。

2

```
$scope.divide = function(a, b) {
    return $q(function(resolve, reject) {
      if (b===0) {
        return reject("Cannot devide by 0")
      } else {
        return resolve(a/b);
      }
    });
}
```

のコードは、されたをしています。がなは、とにをし、があればします。

その、びしてすることができます`.then`

```
$scope.divide(7, 2).then(function(result) {
    // will return 3.5
}, function(err) {
    // will not run
})

$scope.divide(2, 0).then(function(result) {
    // will not run as the calculation will fail on a divide by 0
}, function(err) {
    // will return the error string.
})
```

$ q.deferをしたの

$q.deferをすることにより、のオブジェクトをっているに$qをしてのをすることができます。、またはするをします。

$q.deferをしてをすかどうかまたはさなかったのルーチンをするため、このメソッドは$qコンストラクタのとじではありません。

```
var runAnimation = function(animation, duration) {
    var deferred = $q.defer();
    try {
        ...
        // run some animation for a given duration
        deferred.resolve("done");
    } catch (err) {
        // in case of error we would want to run the error hander of .then
        deferred.reject(err);
    }
    return deferred.promise;
}

// and then
runAnimation.then(function(status) {}, function(error) {})
```

1. に、 deferred.promiseオブジェクトをすか、エラーを.thenがあります。
2. あなたはにあなたのオブジェクトまたはするか、していることをしてください.thenできないがあり、あなたはメモリリークのをします

$ qサービスでのをう

$qは、をし、がしたときにりまたはをするのにつビルトインサービスです。

$qは$rootScope.Scopeモデルのメカニズムとされています。これは、モデルへのやがよりくし、なブラウザのをけ、UIがちらつくことをします。

このでは、プロミスオブジェクトをすファクトリgetMyDataをびします。オブジェクトがresolvedと、がされます。 rejectedは、2にエラーメッセージとともにをします。

のあるで

```
function getMyData($timeout, $q) {
  return function() {
    // simulated async function
    var promise = $timeout(function() {
      if(Math.round(Math.random())) {
        return 'data received!'
      } else {
        return $q.reject('oh no an error! try again')
      }
    }, 2000);
```

```
    return promise;
  }
}
```

でをう

```
angular.module('app', [])
.factory('getMyData', getMyData)
.run(function(getData) {
  var promise = getData()
    .then(function(string) {
      console.log(string)
    }, function(error) {
      console.error(error)
    })
    .finally(function() {
      console.log('Finished at:', new Date())
    })
})
```

をするには、 `$q`をとしてします。ここで`getMyData`ファクトリに`$q`を`getMyData`ました。

```
var defer = $q.defer();
```

`$q.defer()`びすことによって、のしいインスタンスがされます。

オブジェクトは、にをらかにするオブジェクトであり、をするためのするです。これは`$q.deferred()`をってされ、 `resolve()` 、 `reject()` 、 `notify()` 3つのなメソッドをします。

- `resolve(value)` - したをでします。
- `reject(reason)` - したをそのでします。
- `notify(value)` - のにするをします。これは、がまたはされるにびされることがあります。

プロパティ

けられたオブジェクトは、のプロパティをしてアクセスされます。 `promise` - {Promise} - このにけられたオブジェクトをします。

インスタンスがされ、びすことによってすることができたときにしいインスタンスがされ`deferred.promise`。

`promise`オブジェクトのは、したときにがタスクのにアクセスできるようにすることです。

-

- `then(successCallback, [errorCallback], [notifyCallback])` - がされたかどうかにかかわらず、がになるとすぐに、またはエラ―のコ―ルバックをにびします。コ―ルバックは、の、つまりまたはのでびされます。さらに、コ―ルバックは、がまたはされるに、をするために0

びされることがあります。

- `catch(errorCallback)` - promise.thennull、errorCallbackのです。
- `finally(callback, notifyCallback)` - のまたはのいずれかをすることができますが、なをすることはありません。

のもなの1つは、それらをさせるです。これにより、データはをってれ、ステップでされ、される。これは、のでされます。

1

```
// Creates a promise that when resolved, returns 4.
function getNumbers() {

  var promise = $timeout(function() {
    return 4;
  }, 1000);

  return promise;
}

// Resolve getNumbers() and chain subsequent then() calls to decrement
// initial number from 4 to 0 and then output a string.
getNumbers()
  .then(function(num) {
      // 4
      console.log(num);
      return --num;
  })
  .then(function (num) {
      // 3
      console.log(num);
      return --num;
  })
   .then(function (num) {
      // 2
      console.log(num);
      return --num;
  })
  .then(function (num) {
      // 1
      console.log(num);
      return --num;
  })
  .then(function (num) {
      // 0
      console.log(num);
      return 'And we are done!';
  })
  .then(function (text) {
      // "And we are done!"
      console.log(text);
  });
```

$ q.whenをってなをする

もしあなたがとするのは、そのをするものであるならば、ここのようないをうはありません

```
//OVERLY VERBOSE
var defer;
defer = $q.defer();
defer.resolve(['one', 'two']);
return defer.promise;
```

この、のようにくことができます。

```
//BETTER
return $q.when(['one', 'two']);
```

$ q.whenとその$ q.resolve

またはなであるのあるオブジェクトを$ qにラップします。これは、であろうとなかろうとしているオブジェクトをっている、またはができないソースからたにです。

- AngularJS $ qサービスAPIリファレンス - $ q.when

AngularJS v1.4.1のリリースで

また、ES6ののある`resolve`することもでき`resolve`

```
//ABSOLUTELY THE SAME AS when
return $q.resolve(['one', 'two'])
```

$ qパターンをける

このアンチパターンをける

```
var myDeferred = $q.defer();

$http(config).then(function(res) {
   myDeferred.resolve(res);
}, function(error) {
   myDeferred.reject(error);
});

return myDeferred.promise;
```

$ httpサービスがすでにをすので、$ `$q.defer`を`$q.defer`はありません。

```
//INSTEAD
return $http(config);
```

$ httpサービスによってされたをすだけです。

オンラインで$ qサービスによるのをむ https://riptutorial.com/ja/angularjs/topic/4379/--qサービスによるの

4: AngularJSのとしとトラップ

Examples

データバインディングがしなくなる

のことをえておいてください。

1. Angularのデータバインディングは、JavaScriptのプロトタイプのにしているため、 シャドウイングのです 。
2. スコープは、、プロトタイプにスコープからします。 このルールののの1つは、プロトタイプにしないため、したスコープをつディレクティブです。
3. `ng-repeat` 、 `ng-switch` 、 `ng-view` 、 `ng-if` 、 `ng-controller` 、 `ng-include`など、しいスコープをするディレクティブがいくつかあります。

つまり、スコープのにあるプリミティブにデータをでバインドしようとすると、そのようなことがどおりにしないことがあります。 に 、AngularJS を「する」のをします。

このは、のでにできます。

1. ありがとうございます。データをバインドするたびにHTMLテンプレート
2. 「ドットき」オブジェクトへのバインディングのをするため、 `controllerAs`を`controllerAs`
3. $ parentは、スコープではなく、`scope`にアクセスするためにできます。のような`ng-if` 、 々がすることができ`ng-model="$parent.foo"` ..

のわりに`ngModel`をでびされたときにキャッシュされたモデルのバージョンをするgetter / setterにバインドするか、なしでびされたときにモデルをすことができます。 getter / setterをするには、`ngModal`をつに`ng-model-options="{ getterSetter: true }"`をし、そのをにするはgetterをびすがあります 。

```
<div ng-app="myApp" ng-controller="MainCtrl">
    <input type="text" ng-model="foo" ng-model-options="{ getterSetter: true }">
    <div ng-if="truthyValue">
        <!-- I'm a child scope (inside ng-if), but i'm synced with changes from the outside
scope -->
        <input type="text" ng-model="foo">
    </div>
    <div>$scope.foo: {{ foo() }}</div>
</div>
```

コントローラ

```
angular.module('myApp', []).controller('MainCtrl', ['$scope', function($scope) {
    $scope.truthyValue = true;

    var _foo = 'hello'; // this will be used to cache/represent the value of the 'foo' model
```

```
    $scope.foo = function(val) {
        // the function return the the internal '_foo' varibale when called with zero
arguments,
        // and update the internal `_foo` when called with an argument
        return arguments.length ? (_foo = val) : _foo;
    };
}]);
```

ベストプラクティス Angularがコードのの よりもにびすがあるため、ゲッターをにつのがです。

html5Modeをするの

`html5Mode([mode])`をするは、のことがです。

1. `index.html`に`<base href="">`をけて、アプリケーションのベースURLをします。
2. `base`タグは、urlリクエストをつタグのにることがです。それのは、このエラーがするがあります。 `"Resource interpreted as stylesheet but transferred with MIME type text/html"`。えば

   ```
   <head>
       <meta charset="utf-8">
       <title>Job Seeker</title>

       <base href="/">

       <link rel="stylesheet" href="bower_components/bootstrap/dist/css/bootstrap.css" />
       <link rel="stylesheet" href="/styles/main.css">
   </head>
   ```

3. `base`タグをしたくないは、 `$locationProvider`を`requireBase:false`オブジェクトを`$locationProvider.html5Mode()`すことで、 `base`タグをとしないようにします

   ```
   $locationProvider.html5Mode({
       enabled: true,
       requireBase: false
   });
   ```

4. HTML5 URLのみみをサポートするには、サーバーのURLきえをにするがあります。 AngularJS / Developer Guideから/ $ locationをする

   > このモードをするには、サーバーでURLをきえるがあります。には、すべてのリンクをアプリケーションのエントリポイント `index.html` にきすがあります。`<base>`タグをとすることは、アプリケーションベースであるURLのとアプリケーションによってされるべきパスとをすることができるので、このにもです。

   さまざまなHTTPサーバーのにするリクエストきえのれたリソースは、 ui-router FAQ - How tohtml5Modeでするようにサーバーをするをしてください 。たとえば、Apache

```
RewriteEngine on

# Don't rewrite files or directories
RewriteCond %{REQUEST_FILENAME} -f [OR]
RewriteCond %{REQUEST_FILENAME} -d
RewriteRule ^ - [L]

# Rewrite everything else to index.html to allow html5 state links
RewriteRule ^ index.html [L]
```

nginx

```
server {
    server_name my-app;

    root /path/to/app;

    location / {
        try_files $uri $uri/ /index.html;
    }
}
```

エクスプレス

```
var express = require('express');
var app = express();

app.use('/js', express.static(__dirname + '/js'));
app.use('/dist', express.static(__dirname + '/../dist'));
app.use('/css', express.static(__dirname + '/css'));
app.use('/partials', express.static(__dirname + '/partials'));

app.all('/*', function(req, res, next) {
    // Just send the index.html for other files to support HTML5Mode
    res.sendFile('index.html', { root: __dirname });
});

app.listen(3006); //the port you want to use
```

AngularJSの7つのな

は、AngularJSののにがしばしばういくつかのいのリスト、いくつかのされたレッスンおよびそれらにするです。

1.コントローラをって**DOM**をする

それはですが、けなければなりません。コントローラは、をし、データをビューにバインドし、さらにビジネスロジックをするです。あなたはにコントローラのDOMをすることができますが、アプリケーションののでじまたはのがなときはいつでも、のコントローラがになります。このアプローチのベストプラクティスは、すべてのをむディレクティブをし、そのディレクティブをアプリケーションですることです。したがって、コントローラはビューをそのまして、それはです。ディレクティブでは、リンキングはDOMをするもいです。スコープとににアクセスできるため、ディレクティブをすると、をすることもできます。

```
link: function($scope, element, attrs) {
    //The best place to manipulate DOM
}
```

のDOMには、 `element`パラメータ、 `angular.element()`メソッド、またはなJavascriptなど、いくつかのでアクセスできます。

2.のデータバインディング

AngularJSは、のデータバインディングでです。しかし、あなたのデータがディレクティブでにされていることが々あります。そこにまって、AngularJSはっているのではないが、おそらくあなた。ディレクティブは、スコープとスコープがしているため、しです。 1つのトランジションでのがあるとします

```
<my-dir>
    <my-transclusion>
    </my-transclusion>
</my-dir>
```

のには、のにあるデータにバインドされたがいくつかあります。

```
<my-dir>
    <my-transclusion>
        <input ng-model="name">
    </my-transclusion>
</my-dir>
```

のコードはしくしません。ここでは、transclusionによってスコープがされ、nameをできますが、このにえたはそのまります。ですから、このに**$ parent.name**としてにアクセスできます。しかし、このはベストプラクティスではないかもしれません。よりいアプローチは、オブジェクトのをりすことです。たとえば、コントローラでのようにできます。

```
$scope.data = {
    name: 'someName'
}
```

のところ、このに 'data'オブジェクトをしてアクセスし、バインディングがにすることをすることができます。

```
<input ng-model="data.name">
```

だけでなく、アプリをして、ドットきをすることをおめします。

3.のをまとめて

じに2つのディレクティブをにするのはにはです。ルールにうり、2つのしたスコープはじにすることはできません。に、しいカスタムディレクティブをするときは、なパラメータしのためにしたスコープをりてます。ディレクティブmyDirAとmyDirBがisoletedスコープをち、myDirCがそう

でないとすると、のはです。

```
<input my-dir-a my-dirc>
```

のはコンソ―ルエラ―をきこします

```
<input my-dir-a my-dir-b>
```

したがって、スコ―プをして、をにするがあります。

4. $ emの

$ emit、$ broadcast、$ onの、これらは - のでします。いえれば、それらはコントロ―ラののです。えば、のは、コントロ―ラ―Aからの「someEvent」をし、するコントロ―ラ―Bによってされます。

```
$scope.$emit('someEvent', args);
```

そして、のは 'someEvent'

```
$scope.$on('someEvent', function(){});
```

これまでのところすべてがとわれます。しかし、コントロ―ラBがまだされていなければ、イベントはされないことをえておいてください。つまり、エミッタとレシ―バののコントロ―ラをびすがあります。また、$ emitをずするがあるかどうかからないは、サ―ビスをするがいとわれます。

**$**スコ―プの。$ watch

$ scope。$ watchはのをするためにされます。がされるたびに、このメソッドがびされます。しかし、よくあるいの1つは、$ scopeのをすることです。これにより、あるでと$のダイジェストル―プがします。

```
$scope.$watch('myCtrl.myVariable', function(newVal) {
    this.myVariable++;
});
```

したがって、のでは、myVariableとnewValにがないことをしてください。

6.ビュ―へのメソッドのバインド

これはなのつです。 AngularJSにはバインディングがあり、かがされたときはいつでも、ビュ―はもされます。そのため、ビュ―のにメソッドをバインドすると、そのメソッドが100びされるがあり、デバッグににになります。ただし、ng-click、ng-blur、ng-on-changeなど、メソッドをバインドするためにされたはしかありません。たとえば、マ―クアップにのビュ―があるとします。

```
<input ng-disabled="myCtrl.isDisabled()" ng-model="myCtrl.name">
```

ここでは、isDisabledメソッドをしてビュ―のをチェックします。コントロ―ラmyCtrlには、があります。

```
vm.isDisabled = function(){
    if(someCondition)
        return true;
    else
        return false;
}
```

には、しいとわれるかもしれませんが、には、メソッドがにされるため、オ―バ―ロ―ドがします。これをするには、をバインドするがあります。あなたのコントロ―ラには、のがするがあります

```
vm.isDisabled
```

コントロ―ラのにこのをすることができます

```
if(someCondition)
    vm.isDisabled = true
else
    vm.isDisabled = false
```

がしていない、これをのイベントにバインドすることができます。に、このをビュ―にバインドするがあります。

```
<input ng-disabled="myCtrl.isDisabled" ng-model="myCtrl.name">
```

さて、ビュ―のすべてのはどおりで、メソッドはなときにのみされます。

7. Angularのをしない

AngularJSは、コ―ドをするだけでなく、コ―ドをよりにするなど、いくつかのをににします。これらののをにします。

1. **angular.for**ル―プのために、あなたは "すことはできません"それはあなたがにることをぐことができますので、ここでパフォ―マンスをしてください。
2. DOMセレクタの**angular.element**
3. **angular.copy** メインオブジェクトをしないにします
4. フォ―ムのバリデ―ションはすでにらしいです。い、な、れた、な、なものなどをしてください。
5. Chromeデバッガ―のに、モバイルのリモ―トデバッグもします。
6. **Batarang**をしていることをしてください。これは、スコ―プをにできるのChromeです。

オンラインでAngularJSのとしとトラップをむ
https://riptutorial.com/ja/angularjs/topic/3208/angularjsのとしとトラップ

5: AngularJSバインディングオプション `=`、`@`、``など

をってプレイするには、 このプランカーをします。

Examples

@バインディング、バインディング

やなどのリテラルオブジェクトではないをします。

スコープはのをちます。をすると、スコープはのいをちますスコープはスコープのをできません。スコープのをすると、スコープのもされます。すべてのは、ダイジェストだけでなく、ダイジェストびしにされます。

```
<one-way text="Simple text." <!-- 'Simple text.' -->
         simple-value="123" <!-- '123' Note, is actually a string object. -->
         interpolated-value="{{parentScopeValue}}" <!-- Some value from parent scope. You
can't change parent scope value, only child scope value. Note, is actually a string object. --
>
         interpolated-function-value="{{parentScopeFunction()}}" <!-- Executes parent scope
function and takes a value. -->

         <!-- Unexpected usage. -->
         object-item="{{objectItem}}" <!-- Converts object|date to string. Result might be:
'{"a":5,"b":"text"}'. -->
         function-item="{{parentScopeFunction}}"> <!-- Will be an empty string. -->
</one-way>
```

=バインディング。

でをすと、のスコープでをし、のスコープからをするがあります。には{{...}}をすべきではありません。

```
<two-way text="'Simple text.'" <!-- 'Simple text.' -->
         simple-value="123" <!-- 123 Note, is actually a number now. -->
         interpolated-value="parentScopeValue" <!-- Some value from parent scope. You may
change it in one scope and have updated value in another. -->
         object-item="objectItem" <!-- Some object from parent scope. You may change object
properties in one scope and have updated properties in another. -->

         <!-- Unexpected usage. -->
         interpolated-function-value="parentScopeFunction()" <!-- Will raise an error. -->
         function-item="incrementInterpolated"> <!-- Pass the function by reference and you
may use it in child scope. -->
</two-way>
```

ファンクションのをするスコープに2つのウォッチャがされるようにするには、ウォッチャをにえ

るがあります。

バインディング、バインディング

メソッドをディレクティブにします。スコープのコンテキストでをするをします。メソッドはのスコープでされますが、そこにスコープのパラメータをすことができます。には{{...}}をすべきではありません。 をディレクティブですると、スコープにしてされたのをすをしますをすとじではありません。

```
<expression-binding interpolated-function-value="incrementInterpolated(param)" <!--
interpolatedFunctionValue({param: 'Hey'}) will call passed function with an argument. -->
                    function-item="incrementInterpolated" <!-- functionItem({param: 'Hey'})()
will call passed function, but with no possibility set up a parameter. -->
                    text="'Simple text.'"  <!-- text() == 'Simple text.'-->
                    simple-value="123" <!-- simpleValue() == 123 -->
                    interpolated-value="parentScopeValue" <!-- interpolatedValue() == Some
value from parent scope. -->
                    object-item="objectItem"> <!-- objectItem() == Object item from parent
scope. -->
</expression-binding>
```

すべてのパラメータはにラップされます。

なサンプルをしたバインディング

```
angular.component("SampleComponent", {
  bindings: {
     title: '@',
     movies: '<',
     reservation: "=",
     processReservation: "&"
  }
});
```

ここではすべてのがあります。

@は、ウォッチャーなしで、スコープからスコープまで、になバインディングがであることをします。スコープのすべてのはスコープにとどまり、スコープのはスコープにされません。

< バインディングをします。スコープのはスコープにされますが、スコープのはスコープにされません。

=はすでにバインディングとしてられています。スコープのはすべてスコープにされ、すべてのはスコープにされます。

はバインディングにされます。コンポーネントのドキュメントによると、スコープメソッドをするためにするがあります。スコープをするわりに、されたデータでメソッドをびすだけです

オプションのをバインドする

```
bindings: {
   mandatory: '='
   optional: '=?',
   foo: '=?bar'
}
```

オプションのは、でマークするがあります。 `=?`または`=?bar ($compile:nonassign)`にするです。

オンラインでAngularJSバインディングオプション`=`、`@`、``などをむ
https://riptutorial.com/ja/angularjs/topic/6149/angularjsバインディングオプション----------など-

6: AngularJsをしたSignalR

き

このでは、「AngularJsとSignalRをったなプロジェクトの」にをて、このトレーニングでは「anglejsでアプリケーションをする」、「でサービスを/する」、これは https://www.codeproject.com/Tips/590660/Introduction-to-SignalRをおめします 。

Examples

SignalRとAngularJs [チャットプロジェクト]

1プロジェクトをする

```
- Application
    - app.js
    - Controllers
        - appController.js
    - Factories
        - SignalR-factory.js
- index.html
- Scripts
    - angular.js
    - jquery.js
    - jquery.signalR.min.js
- Hubs
```

SignalRバージョンのsignalR-2.2.1

ステップ**2Startup.cs**と**ChatHub.cs**

*"/ Hubs"*ディレクトリにし、 *[Startup.cs*、ChatHub.cs]の 2つのファイルをします。

Startup.cs

```
using Microsoft.Owin;
using Owin;
[assembly: OwinStartup(typeof(SignalR.Hubs.Startup))]

namespace SignalR.Hubs
{
    public class Startup
    {
        public void Configuration(IAppBuilder app)
        {
            app.MapSignalR();
        }
    }
}
```

ChatHub.cs

```
using Microsoft.AspNet.SignalR;

namespace SignalR.Hubs
{
    public class ChatHub : Hub
    {
        public void Send(string name, string message, string time)
        {
            Clients.All.broadcastMessage(name, message, time);
        }
    }
}
```

ステップ**3**アプリをする

*"/ Application"*ディレクトリにし、 *[app.js]*ファイルをしてください

app.js

```
var app = angular.module("app", []);
```

ステップ**4SignalR Factory**をする

*"/ Application / Factories"*ディレクトリにし、 *[SignalR-factory.js]*ファイルをしてください

SignalR-factory.js

```
app.factory("signalR", function () {
    var factory = {};

    factory.url = function (url) {
        $.connection.hub.url = url;
    }

    factory.setHubName = function (hubName) {
        factory.hub = hubName;
    }

    factory.connectToHub = function () {
        return $.connection[factory.hub];
    }

    factory.client = function () {
        var hub = factory.connectToHub();
        return hub.client;
    }

    factory.server = function () {
        var hub = factory.connectToHub();
        return hub.server;
    }

    factory.start = function (fn) {
        return $.connection.hub.start().done(fn);
```

```
    }

    return factory;
});
```

ステップ**5app.js**をする

```
var app = angular.module("app", []);

app.run(function(signalR) {
    signalR.url("http://localhost:21991/signalr");
});
```

localhost21991 / signalr | これはあなたの**SignalR Hubs Urls**です

ステップ**6**コントロ―ラをする

*"/ Application / Controllers"*ディレクトリにし、 *[appController.js]*ファイルをしてください

```
app.controller("ctrl", function ($scope, signalR) {
    $scope.messages = [];
    $scope.user = {};

    signalR.setHubName("chatHub");

    signalR.client().broadcastMessage = function (name, message, time) {
        var newChat = { name: name, message: message, time: time };

        $scope.$apply(function() {
            $scope.messages.push(newChat);
        });
    };

    signalR.start(function () {
        $scope.send = function () {
            var dt = new Date();
            var time = dt.getHours() + ":" + dt.getMinutes() + ":" + dt.getSeconds();

            signalR.server().send($scope.user.name, $scope.user.message, time);
        }
    });
});
```

signalR.setHubName "chatHub"| **[ChatHub]**パブリッククラス**>** *ChatHub.cs*

*HubName*をでしないでください。 のはです。

signalR.client |このメソッドはハブにし、ハブのすべてのをしようとします。このサンプルでは、「chatHub」があり、「broadcastMessage」をします。

ステップ**7**ディレクトリのル―トに**index.html**をする

index.html

```
<!DOCTYPE html>
<html ng-app="app" ng-controller="ctrl">
<head>
    <meta charset="utf-8" />
    <title>SignalR Simple Chat</title>
</head>
<body>
    <form>
        <input type="text" placeholder="name" ng-model="user.name" />
        <input type="text" placeholder="message" ng-model="user.message" />
        <button ng-click="send()">send</button>

        <ul>
            <li ng-repeat="item in messages">
                <b ng-bind="item.name"></b> <small ng-bind="item.time"></small> :
{{item.message}}
            </li>
        </ul>
    </form>

    <script src="Scripts/angular.min.js"></script>
    <script src="Scripts/jquery-1.6.4.min.js"></script>
    <script src="Scripts/jquery.signalR-2.2.1.min.js"></script>
    <script src="signalr/hubs"></script>
    <script src="app.js"></script>
    <script src="SignalR-factory.js"></script>
</body>
</html
```

による

ユーザー1

ユーザー2

オンラインでAngularJsをしたSignalRをむ https://riptutorial.com/ja/angularjs/topic/9964/angularjsをしたsignalr

7: ES6をしたカスタムフィルタ

Examples

ES6をしたFileSizeフィルタ

のモジュールにcostumフィルタをするをするファイルサイズフィルタがあります。

```
let fileSize=function (size,unit,fixedDigit) {
return size.toFixed(fixedDigit) + ' '+unit;
};

  let fileSizeFilter=function () {
        return function (size) {
            if (isNaN(size))
                size = 0;

            if (size < 1024)
                return size + ' octets';

            size /= 1024;

            if (size < 1024)
               return fileSize(size,'Ko',2);

            size /= 1024;

            if (size < 1024)
               return  fileSize(size,'Mo',2);

            size /= 1024;

            if (size < 1024)
               return  fileSize(size,'Go',2);

            size /= 1024;
            return  fileSize(size,'To',2);
        };
    };
export default fileSizeFilter;
```

モジュールへのフィルターびし

```
import fileSizeFilter from 'path...';
let myMainModule =
    angular.module('mainApp', [])
   .filter('fileSize', fileSizeFilter);
```

フィルタとばれるHTMLコード

```
<div ng-app="mainApp">

    <div>
```

```
        <input type="text" ng-model="size" />
    </div>
    <div>
        <h3>Output:</h3>
        <p>{{size| Filesize}}</p>
    </div>
</div>
```

オンラインでES6をしたカスタムフィルタをむ https://riptutorial.com/ja/angularjs/topic/9421/es6をしたカスタムフィルタ

8: ES6をしたコントローラ

Examples

コントローラ

オブジェクトプログラミングにれていれば、ES6でangularJSコントローラをくのはとてもです

```
class exampleContoller{

       constructor(service1,service2,...serviceN){
          let ctrl=this;
              ctrl.service1=service1;
              ctrl.service2=service2;
               .
               .
               .
              ctrl.service1=service1;
              ctrl.controllerName = 'Example Controller';
              ctrl.method1(controllerName)

       }

       method1(param){
          let ctrl=this;
               ctrl.service1.serviceFunction();
              .
              .
              ctrl.scopeName=param;
       }
       .
       .
       .
       methodN(param){
          let ctrl=this;
               ctrl.service1.serviceFunction();
              .
              .
       }


   }
   exampleContoller.$inject = ['service1','service2',...,'serviceN'];
   export default exampleContoller;
```

オンラインでES6をしたコントローラをむ https://riptutorial.com/ja/angularjs/topic/9419/es6をしたコントローラ

9: HTTPインターセプタ

き

AngularJSの$ httpサービスは、バックエンドとしてHTTPリクエストをうことができます。サーバーにするに、すべてのをしてするがあります。それのは、コールをするにレスポンスをしてしたいがあります。グローバルなHTTPエラーもそのようなのいになります。インターセプタはそのようなににされます。

Examples

Angularのみみ$httpサービスをすると、HTTPリクエストをできます。しばしば、リクエストのに、えばリクエストにトークンをする、またはなエラーロジックをするなどのがになります。

なhttpInterceptorをに

ののHTMLファイルをします。

```
<!DOCTYPE html>
<html>
<head>
  <title>Angular Interceptor Sample</title>
  <script src="https://code.angularjs.org/1.5.8/angular.min.js"></script>
  <script src="app.js"></script>
  <script src="appController.js"></script>
  <script src="genericInterceptor.js"></script>
</head>
<body ng-app="interceptorApp">
    <div ng-controller="appController as vm">
        <button ng-click="vm.sendRequest()">Send a request</button>
    </div>
</body>
</html>
```

「app.js」というJavaScriptファイルをします

```
var interceptorApp = angular.module('interceptorApp', []);

interceptorApp.config(function($httpProvider) {
        $httpProvider.interceptors.push('genericInterceptor');
});
```

'appController.js'というのものをしてください

```
(function() {
    'use strict';

    function appController($http) {
```

```
        var vm = this;

        vm.sendRequest = function(){
            $http.get('http://google.com').then(function(response){
                console.log(response);
            });
        };
    }

    angular.module('interceptorApp').controller('appController',['$http', appController]);
})();
```

そして、に、インターセプタ 'genericInterceptor.js'をむファイル

```
(function() {
    "use strict";

    function genericInterceptor($q) {
        this.responseError = function (response) {
            return $q.reject(response);
        };

        this.requestError = function(request){
            if (canRecover(rejection)) {
                return responseOrNewPromise
            }
            return $q.reject(rejection);
        };

        this.response = function(response){
            return response;
        };

        this.request = function(config){
            return config;
        }
    }

    angular.module('interceptorApp').service('genericInterceptor', genericInterceptor);
})();
```

'genericInterceptor'はなをカバーし、アプリケーションになるいをすることができます。

HTTP インターセプタをしたのフラッシュメッセージ

ビューファイル

のスクリプトやアプリケーションでされるHTMLをむHTMLindex.htmlでは、のdivのままにしておくと、このdivのにフラッシュメッセージがされます

```
<div class="flashmessage" ng-if="isVisible">
    {{flashMessage}}
</div>
```

スクリプトファイル

モジュールのconfigメソッドでは、httpProviderをし、httpProviderにはインターセプタプロパティがあり、カスタムインターセプタをプッシュします。のでは、カスタムインターセプタはのみをインターセプトし、rootScopeにされたメソッドをびします。

```
var interceptorTest = angular.module('interceptorTest', []);

    interceptorTest.config(['$httpProvider',function ($httpProvider) {

        $httpProvider.interceptors.push(["$rootScope",function ($rootScope) {
            return {     //intercept only the response
                        'response': function (response)
                                    {

$rootScope.showFeedBack(response.status,response.data.message);

                                      return response;
                                    }
                    };
        }]);

    }])
```

プロバイダのみがモジュールhttpProviderであり、rootスコープではないのconfigメソッドにできるので、モジュールのrunメソッドのでrootscopeにされたメソッドをします。

また、$ timeoutのにメッセージをすると、メッセージにはフラッシュプロパティがあります。これは、しきいにえています。このでは3000 msです。

```
interceptorTest.run(["$rootScope","$timeout",function($rootScope,$timeout){
     $rootScope.showFeedBack = function(status,message){

        $rootScope.isVisible = true;
        $rootScope.flashMessage = message;
        $timeout(function(){$rootScope.isVisible = false },3000)
    }
}]);
```

のとし

$ rootScopeやそののサービスをモジュールのコンフィグレーションメソッドのにしようとすると、アプリケーションのライフサイクルはそのことをさず、のプロバイダエラーがします。モジュールのコンフィグレーションメソッドでプロバイダのみをインジェクトすることができます

オンラインでHTTPインターセプタをむ https://riptutorial.com/ja/angularjs/topic/6484/httpインターセプタ

10: ng-classディレクティブ

Examples

3のng-class

Angularは、 `ng-class`ディレクティブの3のをサポートしています。

1.

```
<span ng-class="MyClass">Sample Text</span>
```

にされるをすると、$ scopeとしてうようにAngularにします。 Angularは$スコープをチェックし、 "MyClass"というをします。 "MyClass"にまれるテキストは、この`<span>`されるのクラスになり`<span>`。 クラスをスペースでってのクラスをすることができます。

あなたのコントローラには、のようながあります

```
$scope.MyClass = "bold-red deleted error";
```

Angularは`MyClass`というをし、$ scopeをつけます。 `<span>`に "bold-red"、 "deleted"、 "error"の3つのクラスをします。

このようにクラスをすると、コントローラのクラスをにできます。たとえば、のユーザーやサーバーからロードされたしいデータにづいてクラスをするがあります。また、するがたくさんあるは、 `$scope`のクラスのリストをするでできます。これは、HTMLテンプレートの`ng-class`にくのをりむよりもです。

2.オブジェクト

これは、どのクラスをするかをするをにできるため、 `ng-class`をしてクラスをするもなです。

キーとのペアをむオブジェクトをします。キーは、きがtrueとされたにされるクラスです。

```
<style>
    .red { color: red; font-weight: bold; }
    .blue { color: blue; }
    .green { color: green; }
    .highlighted { background-color: yellow; color: black; }
</style>

<span ng-class="{ red: ShowRed, blue: ShowBlue, green: ShowGreen, highlighted: IsHighlighted
```

```
}">Sample Text</span>

<div>Red: <input type="checkbox" ng-model="ShowRed"></div>
<div>Green: <input type="checkbox" ng-model="ShowGreen"></div>
<div>Blue: <input type="checkbox" ng-model="ShowBlue"></div>
<div>Highlight: <input type="checkbox" ng-model="IsHighlighted"></div>
```

3.アレイ

にされるでは、 の1をときオブジェクト の2のみわせをできます。

```
<style>
    .bold { font-weight: bold; }
    .strike { text-decoration: line-through; }
    .orange { color: orange; }
</style>

<p ng-class="[ UserStyle, {orange: warning} ]">Array of Both Expression Types</p>
<input ng-model="UserStyle" placeholder="Type 'bold' and/or 'strike'"><br>
<label><input type="checkbox" ng-model="warning"> warning (apply "orange" class)</label>
```

これにより、ユーザーがのクラスをできるスコープ`UserStyle`バインドされたテキストフィールドがされます。これらは、ユーザータイプとして`<p>`ににされます。また、 `warning`スコープにデータバインドされているチェックボックスをクリックすることもできます。これは`<p>`にもにされます。

オンラインでng-classディレクティブをむ https://riptutorial.com/ja/angularjs/topic/2395/ng-classディレクティブ

11: ngModelControllerをするディレクティブ

Examples

なコントロール

のようにするためのシンプルなコントロール、ウィジェットをってみましょう

```
<rating min="0" max="5" nullifier="true" ng-model="data.rating"></rating>
```

はファンシーなCSSはありません。これはのようにされます

```
0 1 2 3 4 5 x
```

をクリックすると、そのがされます。「x」をクリックするとがnullにされます。

```
app.directive('rating', function() {

    function RatingController() {
        this._ngModel = null;
        this.rating = null;
        this.options = null;
        this.min = typeof this.min === 'number' ? this.min : 1;
        this.max = typeof this.max === 'number' ? this.max : 5;
    }

    RatingController.prototype.setNgModel = function(ngModel) {
        this._ngModel = ngModel;

        if( ngModel ) {
            // KEY POINT 1
            ngModel.$render = this._render.bind(this);
        }
    };

    RatingController.prototype._render = function() {
        this.rating = this._ngModel.$viewValue != null ? this._ngModel.$viewValue : -
Number.MAX_VALUE;
    };

    RatingController.prototype._calculateOptions = function() {
        if( this.min == null || this.max == null ) {
            this.options = [];
        }
        else {
            this.options = new Array(this.max - this.min + 1);
            for( var i=0; i < this.options.length; i++ ) {
                this.options[i] = this.min + i;
            }
        }
    };

    RatingController.prototype.setValue = function(val) {
```

```
        this.rating = val;
        // KEY POINT 2
        this._ngModel.$setViewValue(val);
    };

    // KEY POINT 3
    Object.defineProperty(RatingController.prototype, 'min', {
        get: function() {
            return this._min;
        },
        set: function(val) {
            this._min = val;
            this._calculateOptions();
        }
    });

    Object.defineProperty(RatingController.prototype, 'max', {
        get: function() {
            return this._max;
        },
        set: function(val) {
            this._max = val;
            this._calculateOptions();
        }
    });

    return {
        restrict: 'E',
        scope: {
            // KEY POINT 3
            min: '<?',
            max: '<?',
            nullifier: '<?'
        },
        bindToController: true,
        controllerAs: 'ctrl',
        controller: RatingController,
        require: ['rating', 'ngModel'],
        link: function(scope, elem, attrs, ctrls) {
            ctrls[0].setNgModel(ctrls[1]);
        },
        template:
            '<span ng-repeat="o in ctrl.options" href="#" class="rating-option" ng-
class="{\'rating-option-active\': o <= ctrl.rating}" ng-click="ctrl.setValue(o)">{{ o
}}</span>' +
            '<span ng-if="ctrl.nullifier" ng-click="ctrl.setValue(null)" class="rating-
nullifier">&#10006;</span>'
    };
});
```

キーポイント

1. `ngModel.$render`をして、モデルのビューをビューにします。
2. ビューのをするがあるとじるたびに、 `ngModel.$setViewValue()`びします。
3. もちろん、コントロールをパラメータすることもできます。 - バインディングをにすために、> = 1.5のは、パラメータ'<'スコープバインディングを'<'してください。パラメータがされたときにアクションをするがあるは、JavaScriptプロパティ `Object.defineProperty()`をしていくつかのをできます。

1をにさせないために、はにされます `ctrl.options` 。これはではありません。よりですが、よりなでは、 `min` / `max` にDOMをしてを/できます。

2 '<'スコープバインディングをいて、このはAngular <1.5でできます。 Angular> = 1.5のは、これをコンポーネントにし、コントローラのコンストラクタではなく、 `$onInit()` ライフサイクルフックをして `min` および `max` をすることをおめします。

そしてない https : //jsfiddle.net/h81mgxma/

いくつかのなコントロールなオブジェクトをする

カスタムコントロールは、プリミティブのようななものにするはありません。よりいものをすることができます。ここでは、との2のカスタムコントロールをします。アドレスコントロールは、のをするためにされます。はのとおりです。

```
<input-person ng-model="data.thePerson"></input-person>
<input-address ng-model="data.thePerson.address"></input-address>
```

こののモデルはにされています。

```
function Person(data) {
  data = data || {};
  this.name = data.name;
  this.address = data.address ? new Address(data.address) : null;
}

function Address(data) {
  data = data || {};
  this.street = data.street;
  this.number = data.number;
}
```

アドレスエディタ

```
app.directive('inputAddress', function() {

    InputAddressController.$inject = ['$scope'];
    function InputAddressController($scope) {
        this.$scope = $scope;
        this._ngModel = null;
        this.value = null;
        this._unwatch = angular.noop;
    }

    InputAddressController.prototype.setNgModel = function(ngModel) {
        this._ngModel = ngModel;

        if( ngModel ) {
            // KEY POINT 3
            ngModel.$render = this._render.bind(this);
        }
    };
```

```
    InputAddressController.prototype._makeWatch = function() {
        // KEY POINT 1
        this._unwatch = this.$scope.$watchCollection(
            (function() {
                return this.value;
            }).bind(this),
            (function(newval, oldval) {
                if( newval !== oldval ) { // skip the initial trigger
                    this._ngModel.$setViewValue(newval !== null ? new Address(newval) : null);
                }
            }).bind(this)
        );
    };

    InputAddressController.prototype._render = function() {
        // KEY POINT 2
        this._unwatch();
        this.value = this._ngModel.$viewValue ? new Address(this._ngModel.$viewValue) : null;
        this._makeWatch();
    };

    return {
        restrict: 'E',
        scope: {},
        bindToController: true,
        controllerAs: 'ctrl',
        controller: InputAddressController,
        require: ['inputAddress', 'ngModel'],
        link: function(scope, elem, attrs, ctrls) {
            ctrls[0].setNgModel(ctrls[1]);
        },
        template:
            '<div>' +
                '<label><span>Street:</span><input type="text" ng-model="ctrl.value.street"
/></label>' +
                '<label><span>Number:</span><input type="text" ng-model="ctrl.value.number"
/></label>' +
            '</div>'
    };
});
```

キーポイント

1. たちはオブジェクトをしています。たちはからたちにえられたオブジェクトをしたくはありませんたちのモデルはのにするようにしたい。そこで、のオブジェクトにしていをし、プロパティがされるたびに`$setViewValue()`モデルをします。たちはにコピーをします 。
2. モデルがからされるときはいつでも、それをコピーしてコピーをたちのスコープにします。のは、のコピーはではありませんが、はにうまくいくがあります。さらに、ウォッチャーがモデルによってプッシュされたをトリガするのをけるため、ウォッチ`this_unwatch();this._makeWatch();` をします。 UIでわれたにしてのみがトリガーされるようにします。
3. のは、 `ngModel.$render()`をし、 `ngModel.$setViewValue()`をなコントロールと`ngModel.$setViewValue()`ようにびし`ngModel.$render()`を。

のカスタムコントロールのコードはほぼじです。テンプレートは`<input-address>`をしてい`<input-`

address> 。よりなでは、なモジュールでコントローラをすることができました。

```
app.directive('inputPerson', function() {

    InputPersonController.$inject = ['$scope'];
    function InputPersonController($scope) {
        this.$scope = $scope;
        this._ngModel = null;
        this.value = null;
        this._unwatch = angular.noop;
    }

    InputPersonController.prototype.setNgModel = function(ngModel) {
        this._ngModel = ngModel;

        if( ngModel ) {
            ngModel.$render = this._render.bind(this);
        }
    };

    InputPersonController.prototype._makeWatch = function() {
        this._unwatch = this.$scope.$watchCollection(
            (function() {
                return this.value;
            }).bind(this),
            (function(newval, oldval) {
                if( newval !== oldval ) { // skip the initial trigger
                    this._ngModel.$setViewValue(newval !== null ? new Person(newval) : null);
                }
            }).bind(this)
        );
    };

    InputPersonController.prototype._render = function() {
        this._unwatch();
        this.value = this._ngModel.$viewValue ? new Person(this._ngModel.$viewValue) : null;
        this._makeWatch();
    };

    return {
        restrict: 'E',
        scope: {},
        bindToController: true,
        controllerAs: 'ctrl',
        controller: InputPersonController,
        require: ['inputPerson', 'ngModel'],
        link: function(scope, elem, attrs, ctrls) {
            ctrls[0].setNgModel(ctrls[1]);
        },
        template:
            '<div>' +
                '<label><span>Name:</span><input type="text" ng-model="ctrl.value.name"
/></label>' +
                '<input-address ng-model="ctrl.value.address"></input-address>' +
            '</div>'
    };
});
```

ここでオブジェクトはけされています。つまり、なコンストラクタがあります。これはではあり

ません。モデルはなJSONオブジェクトにすることができます。このは、コンストラクタのわりに`angular.copy()`して`angular.copy()`。のは、コントローラが2つのコントロールでになり、のモジュールににできることです。

フィドル https : //jsfiddle.net/3tzyqfko/2/

コントローラーのコードをした2つのバージョンのフィドル https : //jsfiddle.net/agj4cp0e/と https://jsfiddle.net/ugb6Lw8b/

オンラインでngModelControllerをするディレクティブをむ https://riptutorial.com/ja/angularjs/topic/2438/ngmodelcontrollerをするディレクティブ

12: ng-repeat

き

`ngRepeat`ディレクティブは、コレクションからアイテムごとに1テンプレートをインスタンスします。コレクションはまたはオブジェクトでなければなりません。テンプレートインスタンスはのスコープをします。ここでは、されたループはのコレクションアイテムにされ、 `$index`はアイテムインデックスまたはキーにされます。

- `<element ng-repeat="expression"></element>`
- `<div ng-repeat="(key, value) in myObj">...</div>`
- `<div ng-repeat="variable in expression">...</div>`

パラメーター

`$index`	りし0..length-1のオフセットイテレータ
`$first`	*boolean*りしがイテレータののであるはtrueです。
`$middle`	*boolean*りしがイテレータのとのにあるはtrueです。
`$last`	*boolean*りしがイテレータのであるはtrueです。
`$even`	ブール iterator position `$index`がのはtrueそうでないはfalse。
`$odd`	*boolean* iterator position `$index`がのはtrueそうでないはfalse。

AngularJSは、これらのパラメータを、ng-repeatおよびng-repeatがするHTMLタグののでできるなとしてします。

Examples

オブジェクトのプロパティをする

```
<div ng-repeat="(key, value) in myObj"> ... </div>
```

えば

```
<div ng-repeat="n in [42, 42, 43, 43]">
  {{n}}
</div>
```

トラッキングと

`ngRepeat`は、 $ watchCollectionをしてコレクションのをします。がすると、 `ngRepeat`はするDOMへのをいます。

- アイテムがされると、テンプレートのしいインスタンスがDOMにされます。
- アイテムがされると、テンプレートインスタンスがDOMからされます。
- アイテムのがされると、それぞれのテンプレートがDOMでべえられます。

- `track by` 、するをんでいてもよいのリストのため。
- `track by`もリストがきくスピードアップします。
- このに`track by`をしないと、エラーがします `[ngRepeat:dupes]`

```
$scope.numbers = ['1','1','2','3','4'];

<ul>
  <li ng-repeat="n in numbers track by $index">
    {{n}}
  </li>
</ul>
```

ng-repeat-start + ng-repeat-end

AngularJS 1.2 `ng-repeat` 、 `ng-repeat-start`および`ng-repeat-end` `ng-repeat`してのをします。

```
// table items
$scope.tableItems = [
    {
        row1: 'Item 1: Row 1',
        row2: 'Item 1: Row 2'
    },
    {
        row1: 'Item 2: Row 1',
        row2: 'Item 2: Row 2'
    }
];

// template
<table>
    <th>
        <td>Items</td>
    </th>
    <tr ng-repeat-start="item in tableItems">
        <td ng-bind="item.row1"></td>
    </tr>
    <tr ng-repeat-end>
        <td ng-bind="item.row2"></td>
    </tr>
</table>
```

アイテム
11

アイテム
12
21
22

オンラインでng-repeatをむ https://riptutorial.com/ja/angularjs/topic/8118/ng-repeat

13: ngRouteをしたルーティング

ngRouteはngRouteモジュールで、アプリのルーティングとディープリンクサービスとディレクティブをします。

ngRouteにするなドキュメントはhttps://docs.angularjs.org/api/ngRouteにあります。

Examples

な

このでは、3つのルートをつさなアプリケーションをし、それぞれにのビューとコントローラがあり、 controllerAsをしてcontrollerAsます。

ルータはangular .configでします

1. $routeProviderを.config $routeProviderし.config
2. .whenメソッドでルートをルートオブジェクトでします。
3. .whenメソッドに、 templateまたはtemplateUrl 、 controllerおよびcontrollerAsするオブジェクトをします

app.js

```
angular.module('myApp', ['ngRoute'])
  .controller('controllerOne', function() {
    this.message = 'Hello world from Controller One!';
  })
  .controller('controllerTwo', function() {
    this.message = 'Hello world from Controller Two!';
  })
  .controller('controllerThree', function() {
    this.message = 'Hello world from Controller Three!';
  })
  .config(function($routeProvider) {
    $routeProvider
    .when('/one', {
      templateUrl: 'view-one.html',
      controller: 'controllerOne',
      controllerAs: 'ctrlOne'
    })
    .when('/two', {
      templateUrl: 'view-two.html',
      controller: 'controllerTwo',
      controllerAs: 'ctrlTwo'
    })
    .when('/three', {
      templateUrl: 'view-three.html',
      controller: 'controllerThree',
      controllerAs: 'ctrlThree'
    })
    // redirect to here if no other routes match
```

```
    .otherwise({
      redirectTo: '/one'
    });
  });
```

に、たちのHTMLでは、 `<a>`を`href`でしてナビゲーションをし、 `helloRoute`というルートにしては、 `<a href="#/helloRoute">My route</a>`として`<a href="#/helloRoute">My route</a>`

々はまた、コンテナやディレクティブとたちのビューを`ng-view`のルートをするが。

index.html

```
<div ng-app="myApp">
  <nav>
    <!-- links to switch routes -->
    <a href="#/one">View One</a>
    <a href="#/two">View Two</a>
    <a href="#/three">View Three</a>
  </nav>
  <!-- views will be injected here -->
  <div ng-view></div>
  <!-- templates can live in normal html files -->
  <script type="text/ng-template" id="view-one.html">
    <h1>{{ctrlOne.message}}</h1>
  </script>

  <script type="text/ng-template" id="view-two.html">
    <h1>{{ctrlTwo.message}}</h1>
  </script>

  <script type="text/ng-template" id="view-three.html">
    <h1>{{ctrlThree.message}}</h1>
  </script>
</div>
```

パラメータの

このでは、ルートでパラメータをすなをコントローラするためにしています

これをうには、のことがです。

1. ルートにパラメータのとをする
2. コントローラに`$routeParams`サービスを`$routeParams`

app.js

```
angular.module('myApp', ['ngRoute'])
  .controller('controllerOne', function() {
    this.message = 'Hello world from Controller One!';
  })
  .controller('controllerTwo', function() {
    this.message = 'Hello world from Controller Two!';
  })
  .controller('controllerThree', ['$routeParams', function($routeParams) {
    var routeParam = $routeParams.paramName
```

```
    if ($routeParams.message) {
        // If a param called 'message' exists, we show it's value as the message
        this.message = $routeParams.message;
    } else {
        // If it doesn't exist, we show a default message
        this.message = 'Hello world from Controller Three!';
    }
  }])
  .config(function($routeProvider) {
    $routeProvider
    .when('/one', {
      templateUrl: 'view-one.html',
      controller: 'controllerOne',
      controllerAs: 'ctrlOne'
    })
    .when('/two', {
      templateUrl: 'view-two.html',
      controller: 'controllerTwo',
      controllerAs: 'ctrlTwo'
    })
    .when('/three', {
      templateUrl: 'view-three.html',
      controller: 'controllerThree',
      controllerAs: 'ctrlThree'
    })
    .when('/three/:message', { // We will pass a param called 'message' with this route
      templateUrl: 'view-three.html',
      controller: 'controllerThree',
      controllerAs: 'ctrlThree'
    })
    // redirect to here if no other routes match
    .otherwise({
      redirectTo: '/one'
    });
  });
```

に、カスタムメッセージでしいリンクをするだけで、テンプレートのをいながら、しいカスタムメッセージがされます。

index.html

```
<div ng-app="myApp">
  <nav>
    <!-- links to switch routes -->
    <a href="#/one">View One</a>
    <a href="#/two">View Two</a>
    <a href="#/three">View Three</a>
    <!-- New link with custom message -->
    <a href="#/three/This-is-a-message">View Three with "This-is-a-message" custom message</a>
  </nav>
  <!-- views will be injected here -->
  <div ng-view></div>
  <!-- templates can live in normal html files -->
  <script type="text/ng-template" id="view-one.html">
    <h1>{{ctrlOne.message}}</h1>
  </script>

  <script type="text/ng-template" id="view-two.html">
```

```
    <h1>{{ctrlTwo.message}}</h1>
  </script>

  <script type="text/ng-template" id="view-three.html">
    <h1>{{ctrlThree.message}}</h1>
  </script>
</div>
```

##々のルートのカスタムの

々のルートのカスタムをするもなはかなりです。

このでは、ユーザをするためにこのをします。

1 routes.js ののルートにしてしいプロパティ **requireAuth** をする

```
angular.module('yourApp').config(['$routeProvider', function($routeProvider) {
    $routeProvider
        .when('/home', {
            templateUrl: 'templates/home.html',
            requireAuth: true
        })
        .when('/login', {
            templateUrl: 'templates/login.html',
        })
        .otherwise({
            redirectTo: '/home'
        });
}])
```

2にしていないトップのコントローラで**ng-view** とのをするために**$routeProvider** かどうかを**newUrl** する**requireAuth**プロパティをし、それにじてします

```
angular.module('YourApp').controller('YourController', ['$scope', 'session', '$location',
    function($scope, session, $location) {

        $scope.$on('$routeChangeStart', function(angularEvent, newUrl) {

            if (newUrl.requireAuth && !session.user) {
                // User isn't authenticated
                $location.path("/login");
            }

        });
    }
]);
```

オンラインでngRouteをしたルーティングをむ
https://riptutorial.com/ja/angularjs/topic/2391/ngrouteをしたルーティング

14: ng-style

き

'ngStyle'では、HTMLのCSSスタイルをきでできます。AngularJSプロジェクトでHTMLに*style*をするとに、 `ng-style`をanglejsですると、ブールについてスタイルをできます。

- `<ANY ng-style="expression"></ANY >`
- `<ANY class="ng-style: expression;"> ... </ANY>`

Examples

ngの

のは、 "status"パラメータについてのをします。

```
<img class="img-responsive"  ng-src="{{imagesrc}}"
    ng-style="{'opacity' : (status == 2) ? 1 : 0.5}">
```

オンラインでng-styleをむ https://riptutorial.com/ja/angularjs/topic/8773/ng-style

15: TypeScriptでAngularJSをする

- $ scopeng.IScope - これは、ののをするためにtypescriptにあるです。

Examples

Typescriptの

AngularJS ドキュメンテーションでされているように

> コントローラがng-controllerディレクティブをしてDOMにされると、Angularはされた Controllerのコンストラクタをして、しいControllerオブジェクトをインスタンスします。しいスコープがされ、コントローラのコンストラクタへのなパラメータとして$ scopeとしてになります。

コントローラは、typescriptクラスをしてににできます。

```
module App.Controllers {
    class Address {
        line1: string;
        line2: string;
        city: string;
        state: string;
    }
    export class SampleController {
        firstName: string;
        lastName: string;
        age: number;
        address: Address;
        setUpWatches($scope: ng.IScope): void {
            $scope.$watch(() => this.firstName, (n, o) => {
                //n is string and so is o
            });
        };
        constructor($scope: ng.IScope) {
            this.setUpWatches($scope);
        }
    }
}
```

のJavascriptは

```
var App;
(function (App) {
    var Controllers;
    (function (Controllers) {
        var Address = (function () {
            function Address() {
            }
            return Address;
        }());
```

```
        var SampleController = (function () {
            function SampleController($scope) {
                this.setUpWatches($scope);
            }
            SampleController.prototype.setUpWatches = function ($scope) {
                var _this = this;
                $scope.$watch(function () { return _this.firstName; }, function (n, o) {
                    //n is string and so is o
                });
            };
            ;
            return SampleController;
        }());
        Controllers.SampleController = SampleController;
    })(Controllers = App.Controllers || (App.Controllers = {}));
})(App || (App = {}));
//# sourceMappingURL=ExampleController.js.map
```

コントローラクラスをした、クラスをしてコントローラにするjsモジュールをにうことができます

```
app
 .module('app')
 .controller('exampleController', App.Controller.SampleController)
```

ControllerAsでのコントローラの

したコントローラは、コントローラを`controller as`して、インスタンスしてすることができます。これは、を`$scope`ではなくコントローラクラスにしているためです。

`controller as someNamecontroller as someName`ことは、 `$scope`からコントローラをすることです。したがって、$ scopeをコントローラのとしてするはありません。

な

```
// we are using $scope object.
app.controller('MyCtrl', function ($scope) {
  $scope.name = 'John';
});

<div ng-controller="MyCtrl">
  {{name}}
</div>
```

さて、 **`controller as` Syntax `controller as`**

```
// we are using the "this" Object instead of "$scope"
app.controller('MyCtrl', function() {
  this.name = 'John';
});

<div ng-controller="MyCtrl as info">
  {{info.name}}
</div>
```

JavaScriptで「クラス」をインスタンスするは、のようにします。

```
var jsClass = function () {
  this.name = 'John';
}
var jsObj = new jsClass();
```

だから、、たちはすることができます**jsObj**のメソッドやプロパティにアクセスするためにインスタンスを**jsClass**。

では、じタイプのthing.weをインスタンスのとしてします。

バンドル/ミニの

$スコープがコントローラのコンストラクタにされるは、 のオプションをしてするですが、できないため、ができていません。これは、システムがをし、アンガのがパラメータをしてがされるべきかをるためです。したがって、ExampleControllerのコンストラクタはのコードにされています。

```
function n(n){this.setUpWatches(n)
```

$scopeがnされました。
これをするために、$ inject string[] をすることができます。そののDIは、どのにコントローラのコンストラクタをするかをっています。
だからのタイスクリプトは

```
module App.Controllers {
    class Address {
        line1: string;
        line2: string;
        city: string;
        state: string;
    }
    export class SampleController {
        firstName: string;
        lastName: string;
        age: number;
        address: Address;
        setUpWatches($scope: ng.IScope): void {
            $scope.$watch(() => this.firstName, (n, o) => {
                //n is string and so is o
            });
        };
        static $inject : string[] = ['$scope'];
        constructor($scope: ng.IScope) {
            this.setUpWatches($scope);
        }
    }
}
```

なぜ**ControllerAs**ですか

コントローラ

コントローラはJavaScriptのコンストラクタだけではありません。したがって、ビューが`function context`ロードすると **`this`** コントローラオブジェクトにされます。

ケース1

```
this.constFunction = function() { ... }
```

これは`$scope`ではなく`controller object`にされ`$scope`。ビューはコントローラオブジェクトでされたにアクセスできません。

```
<a href="#123" ng-click="constFunction()"></a> // It will not work
```

ケース2

```
$scope.scopeFunction = function() { ... }
```

`$scope object`され、 `controller object`にはされません。ビューは`$scope`オブジェクトでされたにのみアクセスできます。

```
<a href="#123" ng-click="scopeFunction()"></a> // It will work
```

なぜControllerAsですか

- **`ControllerAs`**は、オブジェクトがされているをよりにします`oneCtrl.name`と`anotherCtrl.name`をすると、なるのためにのなるコントローラによってりてられた`name`をするのがはるかににになりますが、がじ`$scope.name`をし、ページの2つのなるHTMLが**`{{name}}`**バインドされている、どのコントローラからのものかをすることはです。

- `$scope`をし、 `intermediary object`してコントローラからビューにメンバーをし`intermediary object`。これを`this.*`にすることで、コントローラからビューにしたいものだけをすることができます。

  ```
  <div ng-controller="FirstCtrl">
      {{ name }}
      <div ng-controller="SecondCtrl">
          {{ name }}
          <div ng-controller="ThirdCtrl">
              {{ name }}
          </div>
      </div>
  </div>
  ```

ここでは、の`{{ name }}`はするのがにらわしく、どのコントローラーにするのかわかりません。

```
<div ng-controller="FirstCtrl as first">
    {{ first.name }}
    <div ng-controller="SecondCtrl as second">
        {{ second.name }}
        <div ng-controller="ThirdCtrl as third">
            {{ third.name }}
        </div>
    </div>
</div>
```

なぜ$スコープですか

- `$scope`、のような$スコープの1つまたはのメソッドにアクセスするがある`$watch`、 `$digest`、 `$emit`、 `$http`などを
- どのプロパティやメソッドが`$scope`にされているかをし、にじて`$scope`にします。

オンラインでTypeScriptでAngularJSをするをむ
https://riptutorial.com/ja/angularjs/topic/3477/typescriptでangularjsをする

16: イベント

パラ メーター

パラ メーター	の
イベント	オブジェクト{name "eventName"、targetScopeScope、defaultPreventedfalse、currentScopeChildScope}
args	イベントとともにされたデータ

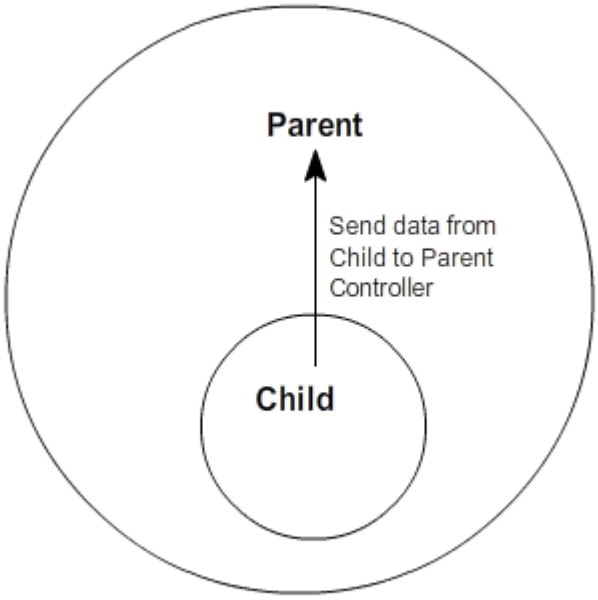
Examples

イベントシステムをする

$ scope。$ emit

`$scope.$emit`を`$scope.$emit`と、スコープをしてイベントがにかってし、 `$scope`されます。イベントライフサイクルは、 `$emit`がびされたスコープからします。

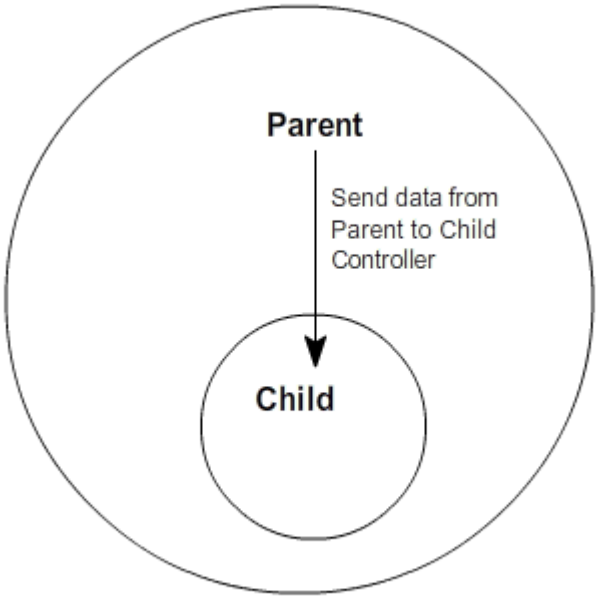
ワーキングワィヤー



$ scope。$ broadcast

`$scope.$broadcast`をすると`$scope.$broadcast`のイベントがし`$scope`。 `$scope.$on`をってこれらのイベントをくことができ`$scope.$on`

ワーキングワイヤー




```
// firing an event upwards
$scope.$emit('myCustomEvent', 'Data to send');

// firing an event downwards
$scope.$broadcast('myCustomEvent', {
  someProp: 'some value'
});

// listen for the event in the relevant $scope
$scope.$on('myCustomEvent', function (event, data) {
  console.log(data); // 'Data from the event'
});
```

`$scope`わりに`$scope $rootScope`をうことができます。その、そのコントローラのスコープになくすべてのコントローラでイベントをできます

されたイベントを**AngularJS**でクリーンアップする

コントローラーがされてされたイベントがされたために、されたイベントをクリーンアップするはまだきています。したがって、コードはしないものとしてされます。

```
// firing an event upwards
$rootScope.$emit('myEvent', 'Data to send');

// listening an event
var listenerEventHandler = $rootScope.$on('myEvent', function(){
    //handle code
});

$scope.$on('$destroy', function() {
    listenerEventHandler();
});
```

と

これらのイベントは、2つのコントローラのにできます。

`$emit`はスコープをにイベント`$emit`ディスパッチし、 `$broadcast` broadcastはすべてのスコープにイベントをディスパッチします 。これはここできれいにされています 。

コントローラでするには、に2つのタイプのシナリオがあります。

---

1. コントローラーにがある。 このようなシナリオでは、ほとんどの`$scope`をできます

---

2. コントローラーがいにしておらず、おいのについてのがな。 このようなシナリオでは `$rootScope`をできます

のeコマースのWebサイトで、ブランドをクリックするとリストページをする`ProductListController`とカートアイテムをする`CartController`があるとします。はカートにボタンをクリックすると、ウェブサイトのナビゲーションバーにしいカートアイテム/をできるように`CartController`にもする`CartController`があります。これは`$rootScope`をってできます。

`$scope.$emit`って`$scope.$emit`

```
<html>
  <head>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.4/angular.js"></script>
    <script>
        var app = angular.module('app', []);

        app.controller("FirstController", function ($scope) {
            $scope.$on('eventName', function (event, args) {
                $scope.message = args.message;
            });
        });

        app.controller("SecondController", function ($scope) {
```

```
            $scope.handleClick = function (msg) {
                $scope.$emit('eventName', {message: msg});
            };
        });

    </script>
  </head>
  <body ng-app="app">
    <div ng-controller="FirstController" style="border:2px ;padding:5px;">
        <h1>Parent Controller</h1>
        <p>Emit Message : {{message}}</p>
        <br />
        <div ng-controller="SecondController" style="border:2px;padding:5px;">
            <h1>Child Controller</h1>
            <input ng-model="msg">
            <button ng-click="handleClick(msg);">Emit</button>
        </div>
    </div>
  </body>
</html>
```

`$scope.$broadcast`って`$scope.$broadcast`

```
<html>
  <head>
    <title>Broadcasting</title>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.4/angular.js"></script>
    <script>
        var app = angular.module('app', []);

        app.controller("FirstController", function ($scope) {
            $scope.handleClick = function (msg) {
                $scope.$broadcast('eventName', {message: msg});
            };

        });

        app.controller("SecondController", function ($scope) {
            $scope.$on('eventName', function (event, args) {
                $scope.message = args.message;
            });
        });

    </script>
  </head>
  <body ng-app="app">
    <div ng-controller="FirstController" style="border:2px solid ; padding:5px;">
        <h1>Parent Controller</h1>
        <input ng-model="msg">
        <button ng-click="handleClick(msg);">Broadcast</button>
        <br /><br />
        <div ng-controller="SecondController" style="border:2px solid ;padding:5px;">
            <h1>Child Controller</h1>
            <p>Broadcast Message : {{message}}</p>
        </div>
    </div>
  </body>
</html>
```

スコープ$ destory イベントのリスナーでは$ rootScope。$をしてください

リスナーの$ rootScope。$は、のコントローラーにするとメモリにります。これにより、コントローラがになったにメモリリークがします。

しないでください

```
angular.module('app').controller('badExampleController', badExample);

badExample.$inject = ['$scope', '$rootScope'];
function badExample($scope, $rootScope) {

    $rootScope.$on('post:created', function postCreated(event, data) {});

}
```

う

```
angular.module('app').controller('goodExampleController', goodExample);

goodExample.$inject = ['$scope', '$rootScope'];
function goodExample($scope, $rootScope) {

    var deregister = $rootScope.$on('post:created', function postCreated(event, data) {});

    $scope.$on('$destroy', function destroyScope() {
        deregister();
    });

}
```

オンラインでイベントをむ https://riptutorial.com/ja/angularjs/topic/1922/イベント

17: カスタムディレクティブ

き

ここでは、AngularJSのディレクティブについてびます。にディレクティブがどのようなものであるかについてのと、ディレクティブのいのなとなをします。

パラメーター

パラメータ	
	ディレクティブのスコープをするプロパティ。これは、false、true、またはスコープとしてできます。{@、=、<、}
スコープ	ディレクティブはスコープをします。ディレクティブのスコープがされていません。
scopetrue	ディレクティブはスコープをプロトタイプにしいスコープとしてします。しいスコープをするじにのディレクティブがある、しいスコープを1つします。
スコープ{@}	ディレクティブスコーププロパティのDOMへのバインディングの1つの。でバインドされたは、ディレクティブでされます。
スコープ{=}	ディレクティブがされたはのをし、そののはをするバインディングです。
スコープ{<}	ディレクティブスコーププロパティとDOMのバインディングはでされます。これは、のアイデンティティをするので、のオブジェクトプロパティのはディレクティブにされません。ディレクティブのオブジェクトプロパティのは、ともじオブジェクトをするため、にされます
スコープ{}	ディレクティブがデータをにされるにせるようにします。
コンパイル	このは、リンクがされるにディレクティブテンプレートでDOMをするためにされます。 `tElement` ディレクティブ·テンプレートおよび`tAttr` ディレクティブでされたのリストを`tAttr`ます。スコープにアクセスすることはできません。 `post-link`としてされるをすことがあります。また、 `pre` `post-link`と`post-link`としてされた`pre-link`プロパティと`post`プロパティをつオブジェクトをすことがあります。
リンク/オブジェクト	リンクプロパティは、またはオブジェクトとしてできます。 scopeディレクティブスコープ、iElementディレクティブがされるDOM、iAttrsDOMのコレクション、controllerディレクティブでなコントローラの、

パラメータ	
	transcludeFnのをけることができます。これはに、DOMリスナの、のモデルプロパティの、DOMのにされます。テンプレートがされたにされます。コンパイルがないはしてされます。
リンク	リンクのにされるリンク。デフォルトでは、ディレクティブ·リンクはディレクティブ·リンクのにされ、プリリンクはがにリンクできるようにします。 1つのユースケースは、がからのデータをとするです。
ポストリンク	ののがにリンクするリンク。イベントハンドラのアタッチやディレクティブへのアクセスによくされますが、ディレクティブがすでにリンクされているため、ディレクティブでなデータはここでしないでください。
restrict	DOMからディレクティブをびすをします。なディレクティブとするとある`demoDirective E` - `<demo-directive></demo-directive>` 、 `A` - `<div demo-directive></div>` `C` -マッチングクラス `<div class="demo-directive"></div>` 、 `M` - コメント `<!-- directive: demo-directive -->` `restrict`プロパティはのオプションをサポートすることもできます。たとえば、 - `restrict: "AC"`は、ディレクティブを*Attribute*または*Class*にします。された、 デフォルトは`"EA"` またはです。
require 'demoDirective'	のでdemoDirectiveのコントローラをし、そのコントローラをリンキングの4としてします。つからなければエラーをげる。
require ' demoDirective'	demoDirectiveのコントローラーをつけようとするか、つからなければリンクfnにヌルをします。
require '^ demoDirective'	とそのをして、demoDirectiveのコントローラをします。つからなければエラーをげる。
require '^^ demoDirective'	のをして、demoDirectiveのコントローラをします。つからなければエラーをげる。
require '^^ demoDirective'	とそのをしてdemoDirectiveのコントローラをつけようとするか、つからなければリンクfnにnullをします。
require '^^ demoDirective'	のをしてdemoDirectiveのコントローラをしたり、つからなければリンクfnにnullをしたりします。

Examples

カスタムディレクティブのと

ディレクティブは、anglejのもなの1つです。カスタムanglejsディレクティブは、htmlタグにのを

するためにしいhtmlまたはカスタムをすることによって、htmlのをするためにされます。

directive.js

```
// Create the App module if you haven't created it yet
var demoApp= angular.module("demoApp", []);

// If you already have the app module created, comment the above line and create a reference
of the app module
var demoApp = angular.module("demoApp");


// Create a directive using the below syntax
// Directives are used to extend the capabilities of html element
// You can either create it as an Element/Attribute/class
// We are creating a directive named demoDirective. Notice it is in CamelCase when we are
defining the directive just like ngModel
// This directive will be activated as soon as any this element is encountered in html

demoApp.directive('demoDirective', function () {

  // This returns a directive definition object
  // A directive definition object is a simple JavaScript object used for configuring the
directive’s behaviour,template..etc
  return {
    // restrict: 'AE', signifies that directive is Element/Attribute directive,
    // "E" is for element, "A" is for attribute, "C" is for class, and "M" is for comment.
    // Attributes are going to be the main ones as far as adding behaviors that get used the
most.
    // If you don't specify the restrict property it will default to "A"
    restrict :'AE',

    // The values of scope property decides how the actual scope is created and used inside a
directive. These values can be either “false”, “true” or “{}”. This creates an isolate scope
for the directive.
    // '@' binding is for passing strings. These strings support {{}} expressions for
interpolated values.
    // '=' binding is for two-way model binding. The model in parent scope is linked to the
model in the directive's isolated scope.
    // '&' binding is for passing a method into your directive's scope so that it can be
called within your directive.
    // The method is pre-bound to the directive's parent scope, and supports arguments.
    scope: {
      name: "@",  // Always use small casing here even if it's a mix of 2-3 words
    },

    // template replaces the complete element with its text.
    template: "<div>Hello {{name}}!</div>",

    // compile is called during application initialization. AngularJS calls it once when html
page is loaded.
    compile: function(element, attributes) {
      element.css("border", "1px solid #cccccc");

      // linkFunction is linked with each element with scope to get the element specific data.
      var linkFunction = function($scope, element, attributes) {
        element.html("Name: <b>"+$scope.name +"</b>");
        element.css("background-color", "#ff00ff");
      };
      return linkFunction;
```

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

このディレクティブは、Appでのようにできます。

```
<html>

   <head>
      <title>Angular JS Directives</title>
   </head>
   <body>
   <script src =
"http://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>
   <script src="directive.js"></script>
    <div ng-app = "demoApp">
       <!-- Notice we are using Spinal Casing here -->
       <demo-directive name="World"></demo-directive>

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

ディレクティブオブジェクトテンプレート

```
demoApp.directive('demoDirective', function () {
  var directiveDefinitionObject = {
    multiElement:
    priority:
    terminal:
    scope: {},
    bindToController: {},
    controller:
    controllerAs:
    require:
    restrict:
    templateNamespace:
    template:
    templateUrl:
    transclude:
    compile:
    link: function(){}
  };
  return directiveDefinitionObject;
});
```

1. **multiElement** - trueにされ、ディレクティブのとののDOMノードがされ、ディレクティブとしてグループされます
2. **priority** - 1つのDOMにのディレクティブがされている、ディレクティブをするをできます。のきいディレクティブがにコンパイルされます。
3. **terminal** - trueにされ、のはするののセットになります
4. **scope** - ディレクティブのスコープをします。
5. **bind to controller** - スコープのプロパティをディレクティブコントローラにバインドする
6. **controller** - コントローラコンストラクタ

7. **require** - のをとし、そのコントローラをリンクの4としてする
8. **controllerAs** - ディレクティブスコープのコントローラへのをして、ディレクティブテンプレートからコントローラをできるようにします。
9. **restrict** - ディレクティブを、、クラス、またはコメントにする
10. **templateNameSpace** - テンプレートによってされるタイプをしますhtml、svg、またはmath。htmlはデフォルトです
11. **template** - デフォルトで、ディレクティブののをきえる、またはtranscludeがtrueのはディレクティブのをラップするHTMLマークアップ
12. **templateUrl** - **templateUrl**にしてでされるurl
13. **transclude** - ディレクティブがれるのをし、ディレクティブができるようにします。コンテンツはコンパイルされ、transclusionとしてディレクティブにされます。
14. **compile** - テンプレートDOMをする
15. **link** - コンパイルプロパティがされていないにのみされます。リンクは、DOMリスナのとDOMのをいます。これは、テンプレートがクローンされたにされます。

の

スーパーマンディレクティブ.**js**

```
angular.module('myApp', [])
  .directive('superman', function() {
    return {
      // restricts how the directive can be used
      restrict: 'E',
      templateUrl: 'superman-template.html',
      controller: function() {
        this.message = "I'm superman!"
      },
      controllerAs: 'supermanCtrl',
      // Executed after Angular's initialization. Use commonly
      // for adding event handlers and DOM manipulation
      link: function(scope, element, attributes) {
        element.on('click', function() {
          alert('I am superman!')
        });
      }
    }
  });
```

superman-template.html

```
<h2>{{supermanCtrl.message}}</h2>
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.5.0/angular.js"></script>
```

```
  <script src="superman-directive.js"><script/>
</head>
<body>
<div ng-app="myApp">
  <superman></superman>
</div>
</body>
</html>
```

ディレクティブの`restrict`と`link`のについては、 AngularJSのディレクティブにするドキュメント

ディレクティブをってresuableコンポーネントをする

AngularJSディレクティブは、AngularJSアプリケーションのHTMLのレンダリングをするものです。 HTML、、クラス、またはコメントにすることができます。ディレクティブは、DOMをするためにされ、HTMLへのしい、データバインディングなどをします。をするディレクティブのとしては、ng-model、ng-hide、ng-ifなどがあります。

に、のカスタムディレクティブをしてリジュームすることもできます。カスタムディレクティブのリファレンスをします。なディレクティブをするにあるは、anglejsがangle.jsをしているのとじように、あなたがしたのディレクティブ/コンポーネントをすることです。これらのなは、した、ルック・アンド・フィールをとするのアプリケーション/アプリケーションをつににちます。このようななコンポーネントのは、アプリケーションまたはなるアプリケーションでしたいが、じようにさせたり、じようにせたりするシンプルなツールバーにすることができます。

に、あなたのappフォルダにresuableComponentsというのフォルダをり、reusableModuleApp.jsをります

reusableModuleApp.js

```
(function(){

 var reusableModuleApp = angular.module('resuableModuleApp', ['ngSanitize']);

 //Remember whatever dependencies you have in here should be injected in the app module where
it is intended to be used or it's scripts should be included in your main app
                              //We will be injecting ng-sanitize

 resubaleModuleApp.directive('toolbar', toolbar)

 toolbar.$inject=['$sce'];

 function toolbar($sce){

     return{
     restrict :'AE',
        //Defining below isolate scope actually provides window for the directive to take data
from app that will be using this.
        scope : {
                value1: '=',
                value2: '=',
                },
```

```
        }
     template : '<ul>  <li><a ng-click="Add()" href="">{{value1}}</a></li>  <li><a ng-
click="Edit()" href="#">{{value2}}</a></li> </ul> ',
        link : function(scope, element, attrs){

             //Handle's Add function
             scope.Add = function(){


             };

             //Handle's Edit function
             scope.Edit = function(){

             };
     }
  }

});
```

mainApp.js

```
(function(){
   var mainApp = angular.module('mainApp', ['reusableModuleApp']); //Inject resuableModuleApp
in your application where you want to use toolbar component

   mainApp.controller('mainAppController', function($scope){
      $scope.value1 = "Add";
      $scope.value2 = "Edit";

   });

 });
```

index.html

```
 <!doctype html>
 <html ng-app="mainApp">
 <head>
  <title> Demo Making a reusable component
 <head>
  <body ng-controller="mainAppController">

     <!-- We are providing data to toolbar directive using mainApp'controller -->
     <toolbar value1="value1" value2="value2"></toolbar>

    <!-- We need to add the dependent js files on both apps here -->
     <script src="js/angular.js"></script>
     <script src="js/angular-sanitize.js"></script>

     <!-- your mainApp.js should be added afterwards --->
      <script src="mainApp.js"></script>

     <!-- Add your reusable component js files here -->
      <script src="resuableComponents/reusableModuleApp.js"></script>

  </body>
 </html>
```

ディレクティブはデフォルトでなコンポーネントです。のモジュールでディレクティブをすると、には、なるangularJsアプリケーションでエクスポートおよびがになります。しいディレクティブをreusableModuleApp.jsににすることができ、reusableModuleAppはのコントローラ、サービス、ディレクティブにDDOオブジェクトをち、をすることができます。

テンプレートとされたスコープをつディレクティブ

したスコープでカスタムディレクティブをすると、ディレクティブのスコープがスコープからされ、ディレクティブがってスコープのデータをしたり、スコープのプライベートデータをみないようにします。

されたスコープをし、カスタムディレクティブがスコープとできるようにするには、ディレクティブのスコープのバインディングをスコープにマップするをする`scope`オプションをできます。

のバインディングは、ディレクティブにされたのでわれます。バインドは、 `scope`オプションと、キーとのペアをつオブジェクトでされます。

- ディレクティブのスコーププロパティにするキー
- Angularに、のスコープをするにバインドするをす

スコープのディレクティブのな

```
var ProgressBar = function() {
  return {
    scope: { // This is how we define an isolated scope
      current: '=', // Create a REQUIRED bidirectional binding by using the 'current'
attribute
      full: '=?maxValue' // Create an OPTIONAL (Note the '?'): bidirectional binding using
'max-value' attribute to the `full` property in our directive isolated scope
    }
    template: '<div class="progress-back">' +
              '  <div class="progress-bar"' +
              '       ng-style="{width: getProgress()}">' +
              '  </div>' +
              '</div>',
    link: function(scope, el, attrs) {
      if (scope.full === undefined) {
        scope.full = 100;
      }
      scope.getProgress = function() {
        return (scope.current / scope.size * 100) + '%';
      }
    }
  }
}

ProgressBar.$inject = [];
angular.module('app').directive('progressBar', ProgressBar);
```

このディレクティブをしてコントローラのスコープからディレクティブのスコープにデータをバインドするの

コントロ―ラ

```
angular.module('app').controller('myCtrl', function($scope) {
    $scope.currentProgressValue = 39;
    $scope.maxProgressBarValue = 50;
});
```

```
<div ng-controller="myCtrl">
    <progress-bar current="currentProgressValue"></progress-bar>
    <progress-bar current="currentProgressValue" max-value="maxProgressBarValue"></progress-
bar>
</div>
```

なコンポ―ネントの

ディレクティブをして、なコンポ―ネントをすることができます。「ユ―ザ―ボックス」コンポ―ネントのをにします。

userBox.js

```
angular.module('simpleDirective', []).directive('userBox', function() {
  return {
    scope: {
        username: '=username',
        reputation: '=reputation'
    },
    templateUrl: '/path/to/app/directives/user-box.html'
  };
});
```

Controller.js

```
var myApp = angular.module('myApp', ['simpleDirective']);

myApp.controller('Controller', function($scope) {

    $scope.user = "John Doe";
    $scope.rep = 1250;

    $scope.user2 = "Andrew";
    $scope.rep2 = 2850;

});
```

myPage.js

```
<html lang="en" ng-app="myApp">
  <head>
    <script src="/path/to/app/angular.min.js"></script>
    <script src="/path/to/app/js/controllers/Controller.js"></script>
    <script src="/path/to/app/js/directives/userBox.js"></script>
  </head>

  <body>
```

```
    <div ng-controller="Controller">
        <user-box username="user" reputation="rep"></user-box>
        <user-box username="user2" reputation="rep2"></user-box>
    </div>

  </body>
</html>
```

user-box.html

```
<div>{{username}}</div>
<div>{{reputation}} reputation</div>
```

はのようになります。

```
John Doe
1250 reputation
Andrew
2850 reputation
```

ディレクティブデコレーター

によっては、のがなもあります。ディレクティブをきえコピーするわりに、ディレクティブのをすることができます。

デコレーターは$ injectでされます。

これをうには、モジューールに.configをしてください。このディレクティブはmyDirectiveとばれ、myDirectiveDirectiveをするがあります。 これはをめたコンベンション[プロバイダーについてむ]。

このでは、ディレクティブのtemplateUrlがされます。

```
angular.module('myApp').config(function($provide){
        $provide.decorator('myDirectiveDirective', function($delegate){
             var directive = $delegate[0]; // this is the actual delegated, your directive
             directive.templateUrl = 'newTemplate.html'; // you change the directive template
        return $delegate;
    })
});
```

このでは、クリックされたときにonClickイベントをディレクティブにします。これはコンパイルフェーズでします。

```
angular.module('myApp').config(function ($provide) {
        $provide.decorator('myDirectiveTwoDirective', function ($delegate) {
            var directive = $delegate[0];
            var link = directive.link; // this is directive link phase
            directive.compile = function () { // change the compile of that directive
                return function (scope, element, attrs) {
```

```
                    link.apply(this, arguments); // apply this at the link phase
                    element.on('click', function(){ // when add an onclick that log hello when
the directive is clicked.
                            console.log('hello!');
                    });
                };
            };
            return $delegate;
        });

    });
```

プロバイダとサービスののでのアプローチをできます。

のと

Angular jsディレクティブはネストするか、にすることができます。

このでは、ディレクティブAdirはBdirがAdirをとするため、ディレクティブBdirにコントローラ$ scopeをします。

```
angular.module('myApp',[]).directive('Adir', function () {
        return {
            restrict: 'AE',
            controller: ['$scope', function ($scope) {
                    $scope.logFn = function (val) {
                            console.log(val);
                        }
                  }]
            }
    })
```

require '^ Adir'をしてくださいのドキュメントをてください。いくつかのバージョンは^をとしません。

```
.directive('Bdir', function () {
        return {
            restrict: 'AE',
            require: '^Adir', // Bdir require Adir
            link: function (scope, elem, attr, Parent) {
            // Parent is Adir but can be an array of required directives.
                elem.on('click', function ($event) {
                    Parent.logFn("Hello!"); // will log "Hello! at parent dir scope
                    scope.$apply(); // apply to parent scope.
                });
            }
        }
    }]);
```

このでディレクティブをネストできます。

```
<div a-dir><span b-dir></span></div>
<a-dir><b-dir></b-dir> </a-dir>
```

ディレクティブが*HTML*にネストされているはありません。

オンラインでカスタムディレクティブをむ https://riptutorial.com/ja/angularjs/topic/965/カスタムディレクティブ

18: カスタムフィルタ

Examples

なフィルタの

フィルタは、ユーザーにするのをします。ビューテンプレート、コントローラ、またはサービスでできます。このでは、フィルタ `addZ` をしてビューでします。このフィルターは、のにの「z」をします。

example.js

```
angular.module('main', [])
    .filter('addZ', function() {
            return function(value) {
                return value + "Z";
            }
     })
    .controller('MyController', ['$scope', function($scope) {
        $scope.sample = "hello";
    }])
```

example.html

ビューのでは、フィルタはのでされます。 `{ variable | filter}` 。この、コントローラ`sample`でしたは、したフィルタ`addZ`によってフィルタリングされています。

```
<div ng-controller="MyController">
   <span>{{sample | addZ}}</span>
</div>
```

されるアウトプット

```
helloZ
```

コントローラー、サービス、またはフィルターでフィルターをする

あなたは`$filter`をするがあり`$filter`

```
angular
  .module('filters', [])
  .filter('percentage', function($filter) {
    return function (input) {
      return $filter('number')(input * 100) + ' %';
    };
```

```
  });
```

パラメータをしてフィルタをする

デフォルトでは、フィルタにはのパラメータ、つまりされるがあります。しかし、よりくのパラメータをにすることができます

```
angular
  .module('app', [])
  .controller('MyController', function($scope) {
    $scope.example = 0.098152;
  })
  .filter('percentage', function($filter) {
    return function (input, decimals) {
      return $filter('number')(input * 100, decimals) + ' %';
    };
  });
```

さて、 `percentage` フィルタにをえることができます

```
<span ng-controller="MyController">{{ example | percentage: 2 }}</span>
=> "9.81 %"
```

...のパラメータはオプションですが、ききデフォルトのフィルタをできます

```
<span ng-controller="MyController">{{ example | percentage }}</span>
=> "9.8152 %"
```

オンラインでカスタムフィルタをむ https://riptutorial.com/ja/angularjs/topic/2552/カスタムフィルタ

19: コントローラ

- <htmlElement ng-controller = "controllerName"> ... </ htmlElement>
- <script> app.controller 'controllerName'、controllerFunction; </ script>

Examples

のコントローラ

コントローラは、スコープをし、ページののアクションをするために、Angularでされるです。コントローラはHTMLビューとされています。

はAngularアプリのです

```
<!DOCTYPE html>

<html lang="en" ng-app='MyFirstApp'>
    <head>
        <title>My First App</title>

        <!-- angular source -->
        <script src="https://code.angularjs.org/1.5.3/angular.min.js"></script>

        <!-- Your custom controller code -->
        <script src="js/controllers.js"></script>
    </head>
    <body>
        <div ng-controller="MyController as mc">
            <h1>{{ mc.title }}</h1>
            <p>{{ mc.description }}</p>
            <button ng-click="mc.clicked()">
                Click Me!
            </button>
        </div>
    </body>
</html>
```

ここですべきがいくつかあります。

```
<html ng-app='MyFirstApp'>
```

`ng-app`アプリケーションをすると、のJavascriptファイルでアプリケーションにアクセスできます。これについては、でします。

```
<script src="js/controllers.js"></script>
```

コントローラとそのアクション/データをするJavascriptファイルがです。

```
<div ng-controller="MyController as mc">
```

ng-controllerは、そのDOMとそれよりのにであるすべてののコントロ―ラをします。

じコントロ―ラ―このはMyController のを... as mcすると、このコントロ―ラ―のインスタンスにをえることができます。

```
<h1>{{ mc.title }}</h1>
```

{{ ... }}はAngularです。この、これは<h1>のテキストをmc.titleのにmc.titleます。

Angularはデュアルウェイのデ―タバインディングをしています。 mc.title 、 mc.titleをするになく、コントロ―ラとペ―ジのにされます。

また、はコントロ―ラをするはありません。 Angularは、 {{ 1 + 2 }}や{{ "Hello " + "World" }}ようににできます。

```
<button ng-click="mc.clicked()">
```

ng-clickはAngularディレクティブです。この、ボタンのClickイベントをバインドして、MyControllerインスタンスのMyController clicked()をトリガしclicked()。

これらのことをにいて、 MyControllerコントロ―ラのをいてみましょう。のでは、このコ―ドをjs/controller.jsします。

まず、Javascriptでアプリをインスタンスするがあります。

```
var app = angular.module("MyFirstApp", []);
```

ここですは、 ng-appディレクティブでHTMLにしたとじng-app。

appオブジェクトができたので、それをってコントロ―ラをすることができます。

```
app.controller('MyController', function(){
    var ctrl = this;

    ctrl.title = "My First Angular App";
    ctrl.description = "This is my first Angular app!";

    ctrl.clicked = function(){
        alert("MyController.clicked()");
    };
});
```

コントロ―ラインスタンスのになりたいものについては、 thisキ―ワ―ドをします。

これは、なコントロ―ラをするためになすべてです。

コントローラの

```
angular
    .module('app')
    .controller('SampleController', SampleController)

SampleController.$inject = ['$log', '$scope'];
function SampleController($log, $scope){
    $log.debug('*****SampleController******');

    /* Your code below */
}
```

.$injectは、あなたのがにスクランブルされないようにします。また、きでにんでいることをしてください。

コントローラの、

コントローラのをからするには、いくつかのがあります。

は、インラインとばれます。のようになります。

```
var app = angular.module('app');
app.controller('sampleController', ['$scope', '$http', function(a, b){
    //logic here
}]);
```

コントローラメソッドの2のパラメータは、のをけれることができます。あなたがることができるようにがした$scopeと$httpれるコントローラのパラメータにしなければならないだろうa $scope、およびbなり$http。 アレイののはコントローラであるがあります。

2のオプションは、 $injectプロパティをしています。のようになります。

```
var app = angular.module('app');
app.controller('sampleController', sampleController);
sampleController.$inject = ['$scope', '$http'];
function sampleController(a, b) {
    //logic here
}
```

これは、インラインとじことをいますが、のオプションよりもきなものにしては、なるスタイルをします。

されたのはです

をしてをするときは、のリストがコントローラにされるのするリストとすることをしてください。

のでは、 $scopeと$httpがになっています。これにより、コードにがします。

```
// Intentional Bug: injected dependencies are reversed which will cause a problem
app.controller('sampleController', ['$scope', '$http',function($http, $scope) {
    $http.get('sample.json');
}]);
```

Angular JSにおけるControllerAsの

Angular `$scope`は、コントローラとビューののので、データバインディングのすべてのニーズにちます。コントローラーは、コントローラーとビューをバインドするもう1つので、にすることをおめします。には、Angularつまり、$ scopeとController Asの2つのコントローラです。

コントローラののい -

controllerAsビューの

```
<div ng-controller="CustomerController as customer">
    {{ customer.name }}
</div>
```

controllerAsコントローラの

```
function CustomerController() {
    this.name = {};
    this.sendMessage = function() { };
}
```

vmをえたコントローラ

```
function CustomerController() {
    /*jshint validthis: true */
    var vm = this;
    vm.name = {};
    vm.sendMessage = function() { };
}
```

`controllerAs`は`$scope`えるです。ビューにバインドして`$scope`メソッドにアクセスすることはできます。 `controllerAs`を`controllerAs`ことは、のコアチームによってされたベストプラクティスの1つです。これにはくのがありますが、そのほとんどが -

- `$scope`は、オブジェクトをしてコントローラからビューにメンバーをしています。これを`this.*`にすることで、コントローラからビューにしたいものだけをすることができます。また、これをするなJavaScriptのにいます。
- `controllerAs`を`controllerAs`と、わかりやすいコードがられ、 `$parent`をするわりに、コントローラのエイリアスをしてプロパティにアクセスできます。
- これは、よりにい、みやすく、 "をたずに"するのあるをける、ビューの「ドットき」オブジェクトへのバインディングのをするえば、のわりにcustomer.nameなど。

- ネストされたコントローラをつビューで$parentびしをしないようにします。
- controllerAsをするは、キャプチャをします。 ViewModelをすvmなどのしたをします。なぜなら、 thisキーワードはなものであり、コントローラのでされると、そのコンテキストをするがあるからです。これのコンテキストをキャプチャすることで、このがすることはありません。

controllerAsをcontrollerAsと、のコントローラへののスコープにされるので、フィールドとしてできます

```
<div ng-controller="Controller as vm>...</div>
```

vmは$scope.vmとしてできます。

ミニネーションセーフの

ミニセーフティコントローラをするには、 controllerパラメータをします。

module.controllerの2のにはをすがありmodule.controller のパラメータはコントローラで 、そののすべてのパラメータはされたのです。

これはのパラダイムとはなります。それはされたでコントローラをきけます。

えられた

```
var app = angular.module('myApp');
```

コントローラーはのようになります。

```
app.controller('ctrlInject',
    [
        /* Injected Parameters */
        '$Injectable1',
        '$Injectable2',
        /* Controller Function */
        function($injectable1Instance, $injectable2Instance) {
            /* Controller Content */
        }
    ]
);
```

されたパラメーターのはするはありませんが、どおりにバインドされます。

これはのようなものにされます

```
var
a=angular.module('myApp');a.controller('ctrlInject',['$Injectable1','$Injectable2',function(b,c){/*
Controller Content */}]);
```

プロセスは、のすべてのインスタンスにきえられます`app`して、のすべてのインスタンス`a` `$Injectable1Instance`と`b` 、およびのすべてのインスタンス`$Injectable2Instance`と`c`。

ネストされたコントローラ

ネスティングコントローラは`$scope`もチェーンします。 `$scope`ネストされたコントローラのは、じ`$scope`コントローラにを。

```
.controller('parentController', function ($scope) {
    $scope.parentVariable = "I'm the parent";
});

.controller('childController', function ($scope) {
    $scope.childVariable = "I'm the child";

    $scope.childFunction = function () {
        $scope.parentVariable = "I'm overriding you";
    };
});
```

さて、これらのをしようとしましょう。

```
<body ng-controller="parentController">
    What controller am I? {{parentVariable}}
    <div ng-controller="childController">
        What controller am I? {{childVariable}}
        <button ng-click="childFunction()"> Click me to override! </button>
    </div>
</body>
```

ネスティングコントローラにはメリットがあるかもしれませんが、そうするには1つがです。 `ngController`ディレクティブをびすと、コントローラのしいインスタンスがされ、やしないがじることがあります。

オンラインでコントローラをむ https://riptutorial.com/ja/angularjs/topic/601/コントローラ

20: コントローラのまたはこの

き

これはなパターンのであり、にAngularJSコードでられるベストプラクティスとみなされます。

Examples

のをする

"コントローラをシンタックスとして"する、ng-controllerディレクティブをすると、コントローラにhtmlのエイリアスがえられます。

```
<div ng-controller="MainCtrl as main">
</div>
```

コントローラのインスタンスをすメインからプロパティとメソッドにアクセスできます。たとえば、コントローラのグリーティングプロパティにアクセスし、それをにします。

```
<div ng-controller="MainCtrl as main">
    {{ main.greeting }}
</div>
```

さて、たちのコントローラでは、コントローラインスタンスのgreetingプロパティにをするがあります$ scopeやそれのもの。

```
angular
.module('ngNjOrg')
.controller('ForgotPasswordController',function ($log) {
  var self = this;

  self.greeting = "Hello World";
})
```

しくHTMLをつためにたちは、コントローラのこのにプロパティをするがありました。は***self***というのをしています。どうしてこのコードをえてみましょう

```
angular
.module('ngNjOrg')
.controller('ForgotPasswordController',function ($log) {
  var self = this;

  self.greeting = "Hello World";

  function itsLate () {
    this.greeting = "Goodnight";
  }
```

```
})
```

このコーードでは、メソッド*itsLate*がびされたときに、のテキストがされるとされるかもしれませんが、にはそうではありません。 JavaScriptはレベルのスコープをしているため、thisLateの "this"はメソッドのの "this"とはなるものをします。しかし、 ***self***をすると、ましいがられます。

```
 angular
.module('ngNjOrg')
.controller('ForgotPasswordController',function ($log) {
  var self = this;

  self.greeting = "Hello World";

  function itsLate () {
    self.greeting = "Goodnight";
  }

})
```

これはあなたのコントローラで ""をすることのしさです。コントローラのどこにでもアクセスでき、コントローラインスタンスをしていることがにできます。

オンラインでコントローラのまたはこのをむ https://riptutorial.com/ja/angularjs/topic/8867/コントローラのまたはこの

21: コンポーネント

パラメーター

パラメータ	
=	データバインディングをする。つまり、コンポーネントスコープでそのをすると、そのがスコープにされます。
<	スコープからをみり、それをしないのバインディング。
@	パラメータ。
そして、	コンポーネントがスコープにかをするがあるのコールバック。
-	-
ライフサイクルフック	angular.version> = 1.5.3が
$ onInit	のすべてのコントローラがされ、バインディングがされたに、コントローラでびされます。これはあなたのコントローラのコードをくのにしています。
$ onChanges changesObj	バインディングがされるたびにびされます。 `changesObj`はされたバウンドプロパティのをキーとするハッシュで、は`{ currentValue, previousValue, isFirstChange() }`ののオブジェクトです。
$ onDestroy	スコープをむスコープがされたときにコントローラでびされます。このフックをして、リソース、ウォッチ、イベントハンドラをします。
$ postLink	このコントローラのとそのがリンクされたにびされます。このフックは、Angular 2のngAfterViewInitフックとngAfterContentInitフックにています。
$ doCheck	ダイジェストサイクルのターンでびされます。をしてするをします。したにじてするアクションは、このフックからするがあります。 $ onChangesがびされたときにこれをしてもはありません。

コンポーネントは、コンポーネントベースのアプリケーションにした、よりシンプルなをするなディレクティブです。コンポーネントはAngular 1.5でされましたが、このセクションのはいAngularJSバージョンではしません 。

コンポーネントにするなガイドはhttps://docs.angularjs.org/guide/componentにされています

Examples

コンポーネントとライフサイクルフック

コンポーネントとはですか

- コンポーネントは、に、よりなをし、Angular 2のすべてであるコンポーネントベースのアーキテクチャにしたディレクティブです。コンポーネントをウィジェットとえるWebアプリケーションのいくつかのなるでできるHTMLコード。

```
angular.module('myApp', [])
    .component('helloWorld', {
        template: '<span>Hello World!</span>'
    });
```

マークアップ

```
<div ng-app="myApp">
  <hello-world> </hello-world>
</div>
```

ライブデモ

コンポーネントでのデータの

コンポーネントにをすパラメータをすると、のようにされます。

```
angular.module("myApp", [])
  .component("helloWorld",{
    template: '<span>Hello {{$ctrl.name}}!</span>',
    bindings: { name: '@' }
  });
```

マークアップ

```
<div ng-app="myApp">
  <hello-world name="'John'" > </hello-world>
</div>
```

ライブデモ

コンポーネントでのコントローラの

コントローラーをするをてみましょう。

```
angular.module("myApp", [])
  .component("helloWorld",{
      template: "Hello {{$ctrl.name}}, I'm {{$ctrl.myName}}!",
      bindings: { name: '@' },
      controller: function(){
        this.myName = 'Alain';
      }
  });
```

マークアップ

```
<div ng-app="myApp">
  <hello-world name="John"> </hello-world>
</div>
```

CodePenデモ

コンポーネントにされるパラメータは、 `$onInit` がAngularによってびされるにコントローラのスコープでできます。このをえてみましょう。

```
angular.module("myApp", [])
  .component("helloWorld",{
      template: "Hello {{$ctrl.name}}, I'm {{$ctrl.myName}}!",
      bindings: { name: '@' },
      controller: function(){
        this.$onInit = function() {
          this.myName = "Mac" + this.name;
        }
      }
  });
```

のテンプレートでは、これは "Hello John、はMacJohnです"とされます。

`$ctrl` は、されていない、 `controllerAs` Angularデフォルトです。

ライブデモ

オブジェクトとして「require」をう

によっては、コンポーネントのコンポーネントからデータにアクセスするがあるがあります。

これは、コンポーネントにコンポーネントがであることをすることによってできます。コンポーネントは、なコンポーネントコントローラへのをし、コントローラですることができますのを。

なコントローラは、$ onInitフックのでのみができていることにしてください。

```
angular.module("myApp", [])
```

```
  .component("helloWorld",{
      template: "Hello {{$ctrl.name}}, I'm {{$ctrl.myName}}!",
      bindings: { name: '@' },
      require: {
        parent: '^parentComponent'
      },
      controller: function () {
        // here this.parent might not be initiated yet

        this.$onInit = function() {
           // after $onInit, use this.parent to access required controller
           this.parent.foo();
        }

      }
  });
```

しかし、これはとののなつながりをりすことにしてください。

JSのコンポーネント

angularJSのコンポーネントはカスタムディレクティブHTMLディレクティブでは<html>これはカスタムディレクティブ<ANYTHING>となりますとしてできます。コンポーネントには、ビューとコントローラがまれています。コントローラには、ユーザーがるビューにバインドされたビジネスロジックがまれます。のがないため、はとはなります。このようにをすることができます。

```
angular.module("myApp",[]).component("customer", {})
```

コンポーネントはモジュールでされます。これらは2つのをみます.1つはコンポーネントので、もう1つはキーのペアをむオブジェクトで、これはどのビューとどのコントローラーをこのようにするかをします。

```
angular.module("myApp",[]).component("customer", {
    templateUrl : "customer.html", // your view here
    controller: customerController, //your controller here
    controllerAs: "cust"        //alternate name for your controller
})
```

"myApp"はビルドするアプリケーションの、customerはコンポーネントのです。これをメインのhtmlファイルでびすために、このようにします

```
<customer></customer>
```

このディレクティブは、したビューとコントローラにきんだビジネスロジックできえられます。

ディレクティブはとしてファクトリをる、コンポーネントは2としてオブジェクトをることをえておいてください。

オンラインでコンポーネントをむ https://riptutorial.com/ja/angularjs/topic/892/コンポーネント

22: サービス

Examples

サービスの

```
angular.module("app")
  .service("counterService", function(){

    var service = {
        number: 0
    };

    return service;
   });
```

サービスの

```
    angular.module("app")

        // Custom services are injected just like Angular's built-in services
        .controller("step1Controller", ['counterService', '$scope', function(counterService,
$scope) {
            counterService.number++;
            // bind to object (by reference), not to value, for automatic sync
            $scope.counter = counterService;
        })
```

このコントローラをするテンプレートでは、のようにします。

```
// editable
<input ng-model="counter.number" />
```

または

```
// read-only
<span ng-bind="counter.number"></span>
```

もちろん、のコードでは、コントローラのメソッドをしてサービスとやりとりします。コントローラのメソッドをすると、サービスにされます。のでは、コントローラがテンプレートでされるたびにカウンタがインクリメントされます。

Angularjsのサービスはシングルトンです

サービスは、アプリケーションごとに$ injectorによってだけインスタンスされ、ロードされたなにのみされるシングルトンオブジェクトです。

シングルトンは、の1つのインスタンスだけをできるクラスであり、インスタンスににアクセスできます。 ここでべたように

angular.factoryをしたサービスの

にサービスをしますこの、ファクトリパターンをします。

```
.factory('dataService', function() {
    var dataObject = {};
    var service = {
        // define the getter method
        get data() {
            return dataObject;
        },
        // define the setter method
        set data(value) {
            dataObject = value || {};
        }
    };
    // return the "service" object to expose the getter/setter
    return service;
})
```

これで、サービスをしてコントローラでデータをできます。

```
.controller('controllerOne', function(dataService) {
    // create a local reference to the dataService
    this.dataService = dataService;
    // create an object to store
    var someObject = {
        name: 'SomeObject',
        value: 1
    };
    // store the object
    this.dataService.data = someObject;
})

.controller('controllerTwo', function(dataService) {
    // create a local reference to the dataService
    this.dataService = dataService;
    // this will automatically update with any changes to the shared data object
    this.objectFromControllerOne = this.dataService.data;
})
```

$ sce - テンプレートのコンテンツとリソースのサニタイズとレンダリング

$ sce "Strict Contextual Escaping"は、テンプレートのコンテンツとソースをにサニタイズするビルトインのサービスです。

ソースとの**HTML**をテンプレートにするには、 `$sce`で`$sce`があります。

このでは、な**$ sce**フィルタをします、。

デモ

```
.filter('sanitizer', ['$sce', [function($sce) {
     return function(content) {
          return $sce.trustAsResourceUrl(content);
      };
}]);
```

テンプレートでの

```
<div ng-repeat="item in items">

    // Sanitize external sources
    <ifrmae ng-src="{{item.youtube_url | sanitizer}}">

    // Sanitaize and render HTML
    <div ng-bind-html="{{item.raw_html_content| sanitizer}}"></div>

</div>
```

'array syntax'をってをつサービスをする

```
angular.module("app")
  .service("counterService", ["fooService", "barService", function(anotherService,
barService){

    var service = {
        number: 0,
        foo: function () {
            return fooService.bazMethod(); // Use of 'fooService'
        },
        bar: function () {
            return barService.bazMethod(); // Use of 'barService'
        }
    };

    return service;
   }]);
```

サービスの

サービスをするもでなは、angular.module APIファクトリをします。

```
angular.module('myApp.services', []).factory('githubService', function() {
   var serviceInstance = {};
   // Our first service
   return serviceInstance;
});
```

サービスファクトリは、コントローラをするとじように、またはのいずれかです。

```
// Creating the factory through using the
// bracket notation
angular.module('myApp.services', [])
.factory('githubService', [function($http) {
}]);
```

たちのサーービスのメソッドをするには、それをサーービスオブジェクトのとしてすることができます。

```
angular.module('myApp.services', [])
    .factory('githubService', function($http) {
        var githubUrl = 'https://api.github.com';
        var runUserRequest = function(username, path) {
        // Return the promise from the $http service
        // that calls the Github API using JSONP
        return $http({
            method: 'JSONP',
            url: githubUrl + '/users/' +
            username + '/' +
            path + '?callback=JSON_CALLBACK'
        });
    }
// Return the service object with a single function
// events
return {
    events: function(username) {
    return runUserRequest(username, 'events');
    }
};
```

サーービスとのい

1サーービス

サーービスは、にnewでびされるconstructorです。これは、なjavascriptのとじように、 AngularJsがnewでびしているだけです。

サーービスのにえておくべき1つののルーールがあります

1. サーービスはnewものでばれるコンストラクタです

$httpサーービスをってのをし、それをコントローーラでするサーービスをするなをてみましょう

```
function StudentDetailsService($http) {
  this.getStudentDetails = function getStudentDetails() {
    return $http.get('/details');
  };
}

angular.module('myapp').service('StudentDetailsService', StudentDetailsService);
```

このサーービスをコントローーラにするだけです

```
function StudentController(StudentDetailsService) {
  StudentDetailsService.getStudentDetails().then(function (response) {
      // handle response
    });
}
angular.module('app').controller('StudentController', StudentController);
```

いつしますか

.service()は、コンストラクタをしたいでします。これは、 getStudentDetails()ようにパブリックAPIをするためにされます。しかし、コンストラクタをいたくなく、シンプルなAPIパターンをいたいは、 .service()があまりありません。

2

たちがしてすべてのものをることができたとしても.factory()々は、しているでしょう.services()それはなりません.factory()「とじ」.service() .service()よりもはるかにでがあり.service()

.factory()は、をすためにされるデザインパターンです。

のにえておくべき2つののルールがあります

1. ファクトリがをす
2. ファクトリがオブジェクトをするのオブジェクト

.factory()をってできることのいくつかのをて.factory()う。

リテラルをす

なRevealingモジュールパターンをしてオブジェクトをすためにファクトリがされるをてみましょう

```
function StudentDetailsService($http) {
  function getStudentDetails() {
    return $http.get('/details');
  }
  return {
    getStudentDetails: getStudentDetails
  };
}

angular.module('myapp').factory('StudentDetailsService', StudentDetailsService);
```

コントローラの

```
function StudentController(StudentDetailsService) {
  StudentDetailsService.getStudentDetails().then(function (response) {
      // handle response
    });
}
angular.module('app').controller('StudentController', StudentController);
```

る

とはですか

クロージャーは、ローカルでされているが、みスコープでされているをするです。

はクロージャのです

```
function closureFunction(name) {
  function innerClosureFunction(age) { // innerClosureFunction() is the inner function, a
closure
    // Here you can manipulate 'age' AND 'name' variables both
  };
};
```

「らしい」は、スコープの`name`アクセスできることです。

`.factory()`でののをします。

```
function StudentDetailsService($http) {
  function closureFunction(name) {
  function innerClosureFunction(age) {
    // Here you can manipulate 'age' AND 'name' variables
    };
  };
};

angular.module('myapp').factory('StudentDetailsService', StudentDetailsService);
```

コントローラの

```
function StudentController(StudentDetailsService) {
  var myClosure = StudentDetailsService('Student Name'); // This now HAS the
innerClosureFunction()
  var callMyClosure = myClosure(24); // This calls the innerClosureFunction()
};

angular.module('app').controller('StudentController', StudentController);
```

コンストラクタ/インスタンスの

`.service()`は、のように`new`をびすコンストラクタをします。 `.factory()`は、 `new`へのびしでコンストラクタをすることもできます

これをするのをてみましょう

```
function StudentDetailsService($http) {
  function Student() {
    this.age = function () {
        return 'This is my age';
    };
  }
  Student.prototype.address = function () {
        return 'This is my address';
  };
  return Student;
};

angular.module('myapp').factory('StudentDetailsService', StudentDetailsService);
```

コントローラの

```
function StudentController(StudentDetailsService) {
  var newStudent = new StudentDetailsService();

  //Now the instance has been created. Its properties can be accessed.

 newStudent.age();
 newStudent.address();

};

angular.module('app').controller('StudentController', StudentController);
```

オンラインでサービスをむ https://riptutorial.com/ja/angularjs/topic/1486/サービス

23: サービスとファクトリの

Examples

ファクトリーVSサービスりのサービス

により

サービスはにコンストラクタです。らは 'this'キーワードをします。

ファクトリはななので、オブジェクトをす。

フードの

ファクトリはにプロバイダをびします。

サービスはにファクトリをびします。

ディベート

ファクトリは、オブジェクトリテラルをすにコードをできます。

しかしに、サービスはオブジェクトリテラルをすようにかれ、されるにコードをすることもできます。サービスはコンストラクタとしてするようにされているので、それはのです。

、JavaScriptのコンストラクタは、なものをすことができます。

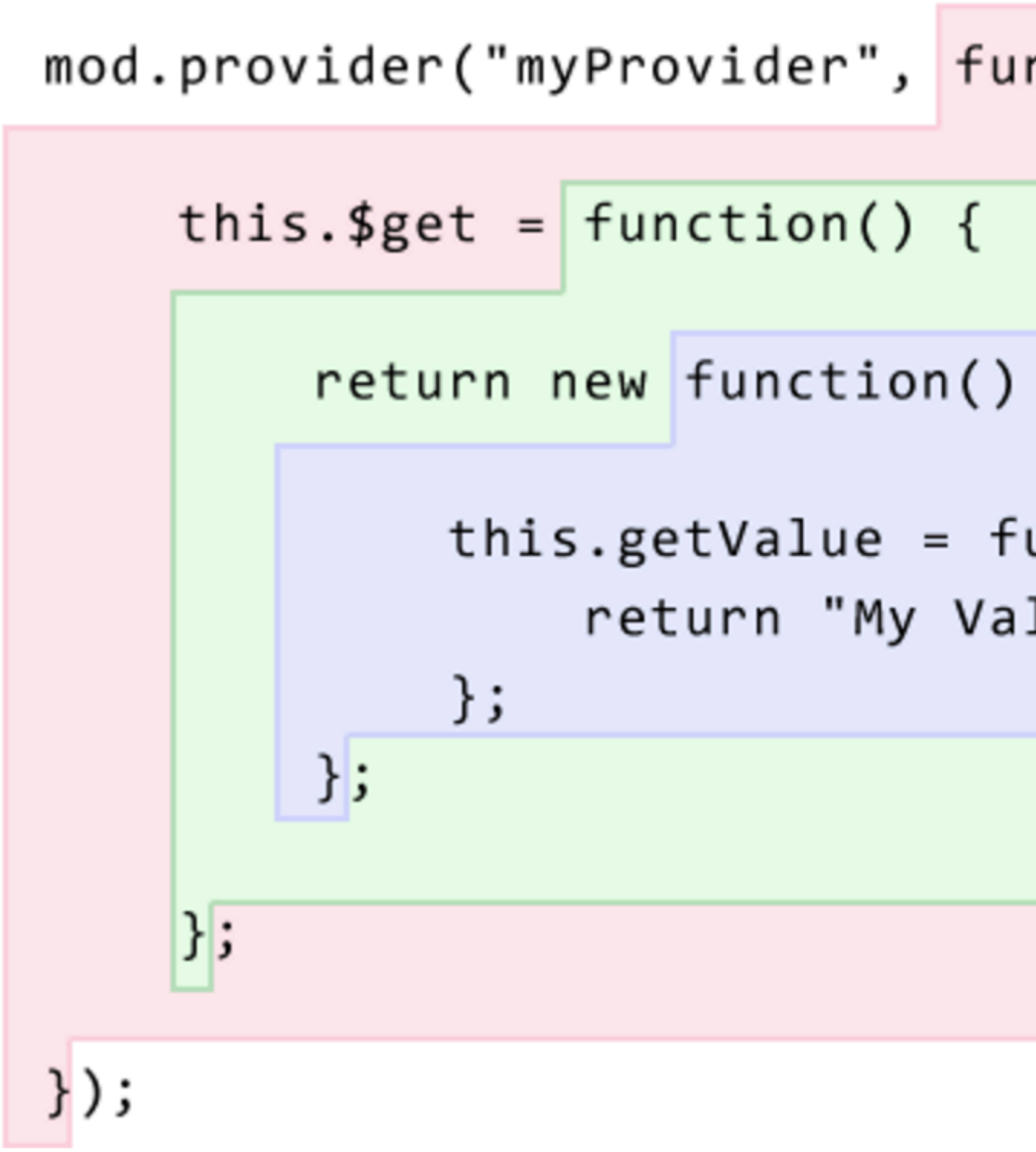
だからどちらがいですか

サービスのコンストラクタは、ES6のクラスにいものです。はです。

すると、プロバイダ、ファクトリ、サービスはすべてプロバイダです。

ファクトリは、プロバイダでなものが$ getである、プロバイダのなケースです。ないコードでくことができます。

サービスは、しいオブジェクトのインスタンスをすときに、ファクトリのなケースです。コードをくこととじがあります。



オンラインでサービスとファクトリのをむ https://riptutorial.com/ja/angularjs/topic/7099/サービスとファクトリの

24: セッション

Examples

angularjsをしたサービスによるセッションの

セッションストレージサービス

キーについてされたセッションデータをおよびすのファクトリサービス。

```
 'use strict';

/**
 * @ngdoc factory
 * @name app.factory:storageService
 * @description This function will communicate with HTML5 sessionStorage via Factory Service.
 */

app.factory('storageService', ['$rootScope', function($rootScope) {

    return {
        get: function(key) {
            return sessionStorage.getItem(key);
        },
        save: function(key, data) {
            sessionStorage.setItem(key, data);
        }
    };
}]);
```

コントローラ

コントローラのstorageServiceをして、セッションストレージからデータをしてします。

```
app.controller('myCtrl',['storageService',function(storageService) {

  // Save session data to storageService
  storageService.save('key', 'value');

  // Get saved session data from storageService
  var sessionData = storageService.get('key');

});
```

オンラインでセッションをむ https://riptutorial.com/ja/angularjs/topic/8201/セッション

25: ダイジェストループのウォークスルー

- $ scope。$ watchwatchExpression、callback、[deep compare]
- $ scope。$ digest
- $ scope。$ apply[exp]

Examples

データバインディング

Angularはそのフードのでいくつかのをっています。 DOMをのjsにバインドすることができます。

Angularは、 " ダイジェストループ "というのループをします。これは、DOMをするびしコールバックのにびされます。

たとえば、 `ng-model`ディレクティブは、 `keyup` eventListenerをこのにします。

```
<input ng-model="variable" />
```

`keyup`イベントがするたびに、 ダイジェストループがされます。

あるで、 ダイジェストループはこのスパンのをするコールバックをします

```
<span>{{variable}}</span>
```

こののなライフサイクルは、にこのきをまとめたものです::

1. スキャンhtml
    - `ng-model`ディレクティブは、に`keyup`リスナーをします
    - spanの`expression`がダイジェストサイクルへのコールバックをする
2. ユーザーはとする
    - `keyup`リスナーがダイジェストサイクルをする
    - ダイジェストサイクルはコールバックをびします
    - コールバックのスパン

$ digestと$ watch

ののをるためにデータバインディングをするには、の2つのコアをできます。

- **$ digest**はユーザーのやりりのにびされますDOM =>をバインドします
- **$ watch**は、のにびされるコールバックをしますvariable => DOM

のコードではなくデモです

```
<input id="input"/>
<span id="span"></span>
```

な2つの

```
var $watches = [];
function $digest(){
    $watches.forEach(function($w){
        var val = $w.val();
        if($w.prevVal !== val){
            $w.callback(val, $w.prevVal);
            $w.prevVal = val;
        }
    })
}
function $watch(val, callback){
    $watches.push({val:val, callback:callback, prevVal: val() })
}
```

これで、これらのをしてをDOMにフックできますにはみみディレクティブがしています。

```
var realVar;
//this is usually done by ng-model directive
input1.addEventListener('keyup',function(e){
    realVar=e.target.value;
    $digest()
}, true);

//this is usually done with {{expressions}} or ng-bind directive
$watch(function(){return realVar},function(val){
    span1.innerHTML = val;
});
```

もちろん、のはよりで、バインドするやするなどのパラメ―タをサポ―トしています

のがここにあります https : //jsfiddle.net/azofxd4j/

$スコープツリー

のは、のhtmlをのにバインドするがあるにです。

には、くのをくのにバインドするがあります。

```
<span ng-repeat="number in [1,2,3,4,5]">{{number}}</span>
```

この`ng-repeat`は、5を`number`という5つのにバインドします。それぞれのにはなるがされます。

がこのるいをするは、々のをとするにのコンテキストをしています。このコンテキストはスコープとばれます。

スコープにはDOMにバインドされたであるプロパティがあり、 `$digest`と`$watch`はスコープのメソ

ッドとしてされています。

DOMはツリーであり、はツリーのなるレベルでするがあります。

```
<div>
    <input ng-model="person.name" />
    <span ng-repeat="number in [1,2,3,4,5]">{{number}} {{person.name}}</span>
</div>
```

しかし、われわれがたように、 `ng-repeat`ののコンテキストまたはスコープは、それをるコンテキストとはなります。これをするために、はスコープをツリーとしてします。

スコープにはのがあり、 `$digest`メソッドをびすと、すべてのの`$digest`メソッドがされます。

このでは、をした、 `$digest`がdivのスコープにしてびされ、5のの`$digest`がされ、コンテンツがされます。

スコープのなは、のようになります。

```
function $scope(){
    this.$children = [];
    this.$watches = [];
}

$scope.prototype.$digest = function(){
    this.$watches.forEach(function($w){
        var val = $w.val();
        if($w.prevVal !== val){
            $w.callback(val, $w.prevVal);
          $w.prevVal = val;
        }
    });
    this.$children.forEach(function(c){
        c.$digest();
    });
}

$scope.prototype.$watch = function(val, callback){
    this.$watches.push({val:val, callback:callback, prevVal: val() })
}
```

のコードではなくデモです

オンラインでダイジェストループのウォークスルーをむ
https://riptutorial.com/ja/angularjs/topic/3156/ダイジェストループのウォークスルー

26: データの

Angularでするときのよくあるは、コントローラでデータをするです。 サービスのがもにわれるレスポンスで、これは2つのコントローラでのタイプのデータオブジェクトをするためのファクトリパターンをすなです。これはオブジェクトであるため、あるコントローラのは、そのサービスをするのすべてのコントローラでちににになります。サービスとのとのプロバイダにしてください。

Examples

ngStorageをってデータをする

まず、index.htmlにngStorageソースをめます。

`ngStorage` srcをするは、のようになります。

```
<head>
    <title>Angular JS ngStorage</title>
    <script src =
"http://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>
    <script src="https://rawgithub.com/gsklee/ngStorage/master/ngStorage.js"></script>
</head>
```

`ngStorage`は2つのストレージ、すなわち`$localStorage`と`$sessionStorage`ます。 ngStorageをし、サービスをするがあります。

`ng-app="myApp"`、のように`ngStorage`をするとします。

```
var app = angular.module('myApp', ['ngStorage']);
        app.controller('controllerOne', function($localStorage,$sessionStorage) {
        // an object to share
        var sampleObject = {
            name: 'angularjs',
            value: 1
        };
       $localStorage.valueToShare = sampleObject;
       $sessionStorage.valueToShare = sampleObject;
    })
.controller('controllerTwo', function($localStorage,$sessionStorage) {
    console.log('localStorage: '+ $localStorage +'sessionStorage: '+$sessionStorage);
})
```

`$localStorage`と`$sessionStorage`は、それらのサービスをコントローラにするり、どのコントローラからでもグローバルにアクセスできます。

`HTML5 localStorage`と`sessionStorage`をすることもでき`HTML5`。ただし、 `HTML5 localStorage`をするには、オブジェクトをまたはするに、オブジェクトをおよびシリアルするがあります。

えば

```
var myObj = {
    firstname: "Nic",
    lastname: "Raboy",
    website: "https://www.google.com"
}
//if you wanted to save into localStorage, serialize it
window.localStorage.set("saved", JSON.stringify(myObj));

//unserialize to get object
var myObj = JSON.parse(window.localStorage.get("saved"));
```

サービスをしてコントローラでデータをする

controllersでデータをsetしてgetするserviceをし、それをしたいコントローラにそのサービスをすることができます。

サービス

```
app.service('setGetData', function() {
  var data = '';
    getData: function() { return data; },
    setData: function(requestData) { data = requestData; }
});
```

コントローラ

```
app.controller('myCtrl1', ['setGetData',function(setGetData) {

  // To set the data from the one controller
  var data = 'Hello World !!';
  setGetData.setData(data);

}]);

app.controller('myCtrl2', ['setGetData',function(setGetData) {

  // To get the data from the another controller
  var res = setGetData.getData();
  console.log(res); // Hello World !!

}]);
```

ここで、 myCtrl1はデータのsettingされ、 myCtrl2はデータのgettingされるgettingがmyCtrl2ます。そこで、あるコントローラのデータをこのようなコントローラにすることができます。

オンラインでデータのをむ https://riptutorial.com/ja/angularjs/topic/1923/データの

27: データバインディングのしくみ

したがって、このデータバインディングのコンセプトはにとってはですが、Angularはすべてのイベントのをリッスンしてダイジェストサイクルをするため、ブラウザではかなりいです。このため、ビューにモデルをするときは、スコープができるだけされていることをしてください

Examples

データバインディングの

```
<p ng-bind="message"></p>
```

この'メッセージ'は、のコントローラのスコープにアタッチされているがあります。

```
$scope.message = "Hello World";
```

でメッセージモデルがされたとしても、そのされたはHTMLにされます。をコンパイルすると、テンプレート "Hello World"がののinnerHTMLにされます。 Angularは、ビューにアタッチされたすべてのディレクティブのWatchingメカニズムをします。 WatchersをするDigest Cycleメカニズムがあり、モデルののにがあった、DOMがされます。

スコープには、オブジェクトにがえられているかどうかがにチェックされていません。スコープにされたオブジェクトのすべてがされるわけではありません。スコープはプロトタイプに**$$ WatchersArray**をします。スコープは、$ digestがびされたときにのみ、このWatchersArrayをします。

AngularはWatchersArrayにウォッチャーをします

1. {{}} - テンプレートおよびがあるまたはng-modelをするとき。
2. $ scope。$ watch 'expression / function' - あなたのJavaScriptでは、ウォーニングのためのスコープオブジェクトをすることができます。

$ watchは3つのパラメータをります

1. はオブジェクトをすウォッチャーか、をするだけです。
2. 2つは、オブジェクトにがあったときにびされるリスナーです。 DOMののようなものはすべてこのでされます。
3. 3のはブールをとるオプションのパラメータです。ののいディープがオブジェクトをし、そののがオブジェクトをするだけの。 $ watchのいはのようになります

```
Scope.prototype.$watch = function(watchFn, listenerFn) {
   var watcher = {
```

```
        watchFn: watchFn,
        listenerFn: listenerFn || function() { },
        last: initWatchVal  // initWatchVal is typically undefined
    };
    this.$$watchers.push(watcher); // pushing the Watcher Object to Watchers
};
```

AngularにはDigest Cycleといういことがあります。 $ digestは$ scope。$ digestのびしのとしてされます。 ng-clickディレクティブをしてハンドラの$ scopeモデルをするとします。この、AngularJSはに$ digestをびすことで$ digestサイクルをします.ng-clickのに、モデルをするためのいくつかのみみディレクティブ/サービスがありますng-model、$ timeoutなど $ダイジェストサイクルをにします。 $ digestのまかなはのようになります。

```
Scope.prototype.$digest = function() {
       var dirty;
       do {
           dirty = this.$$digestOnce();
       } while (dirty);
}
Scope.prototype.$$digestOnce = function() {
    var self = this;
    var newValue, oldValue, dirty;
    _.forEach(this.$$watchers, function(watcher) {
           newValue = watcher.watchFn(self);
           oldValue = watcher.last;   // It just remembers the last value for dirty checking
           if (newValue !== oldValue) { //Dirty checking of References
    // For Deep checking the object , code of Value
    // based checking of Object should be implemented here
              watcher.last = newValue;
              watcher.listenerFn(newValue,
                   (oldValue === initWatchVal ? newValue : oldValue),
                    self);
           dirty = true;
           }
      });
    return dirty;
 };
```

JavaScriptの**setTimeout**をしてスコープモデルをすると、Angularはするのあることをることができません。この、$ applyをでびすのはたちのです。これは$ダイジェストサイクルをトリガーします。に、DOMイベントリスナーをし、ハンドラのいくつかのモデルをするがあるは、$ applyをびしてがになるようにするがあります。 $ applyのきなアイデアは、Angularをしないコードをできることです。そのコードはスコープのものをするがあります。そのコードを$ applyにラップすると、$ digestのびしがされます。 $ applyのまかな。

```
Scope.prototype.$apply = function(expr) {
       try {
         return this.$eval(expr); //Evaluating code in the context of Scope
       } finally {
         this.$digest();
       }
};
```

オンラインでデータバインディングのしくみをむ https://riptutorial.com/ja/angularjs/topic/2342/デ

28: データフィルタ、ページネーションなどの anglejs

き

フィルタとページングなどのデータにするプロバイダのとクエリについてはAngularjsでします。

Examples

Angularjsは、フィルタ、ページきのデータをします。

```
<div ng-app="MainApp" ng-controller="SampleController">
    <input ng-model="dishName" id="search" class="form-control" placeholder="Filter text">
    <ul>
        <li dir-paginate="dish in dishes |  filter : dishName | itemsPerPage: pageSize"
pagination-id="flights">{{dish}}</li>
    </ul>
    <dir-pagination-controls boundary-links="true" on-page-
change="changeHandler(newPageNumber)" pagination-id="flights"></dir-pagination-controls>
</div>
<script type="text/javascript" src="angular.min.js"></script>
<script type="text/javascript" src="pagination.js"></script>
<script type="text/javascript">

var MainApp = angular.module('MainApp', ['angularUtils.directives.dirPagination'])
MainApp.controller('SampleController', ['$scope', '$filter', function ($scope, $filter) {

    $scope.pageSize = 5;

    $scope.dishes = [
    'noodles',
    'sausage',
    'beans on toast',
    'cheeseburger',
    'battered mars bar',
    'crisp butty',
    'yorkshire pudding',
    'wiener schnitzel',
    'sauerkraut mit ei',
    'salad',
    'onion soup',
    'bak choi',
    'avacado maki'
    ];

    $scope.changeHandler = function (newPage) { };
}]);
</script>
```

オンラインでデータフィルタ、ページネーションなどのanglejsをむ
https://riptutorial.com/ja/angularjs/topic/10821/データフィルタ-ページネーションなどのanglejs

29: デコレータ

- デコレータ、デコレータ;

  Decoratorは、 サービス 、 ファクトリ 、 ディレクティブまたはフィルタをにできるようにするです。デコレータは、サービスのをまたはするためにされます。デコレータのりは、のサービス、またはのサービスをきえたり、のサービスにラップしてしたりするしいサービスです。

すべてのは 、アプリケーションの`config`フェーズで、 `$provide`をし、 `$provide.decorator`をして`$provide`あります。

> デコレータには、デコレータのセレクタにするサービスへのアクセスをするために`$delegate`オブジェクトがされています。この`$delegate`はあなたがっているサービスになります。デコレータにされるのりは、サービス、ディレクティブ、またはフィルタがされたでわれます。

のがでないや、あまりにもであることがしたにのみ、デコレータのをするがあります。なアプリケーションがじサービスをしており、1つのがサービスのをしているは、そのプロセスでやバグをするのはです。

なは、アップグレードできないがそれほどのやがなのがあるです。

Examples

るサービス、

はサービスデコレータのであり、サービスによってされる`null`オーバーライドし`null`。

```
angular.module('app', [])
  .config(function($provide) {
    $provide.decorator('myService', function($delegate) {
      $delegate.getDate = function() { // override with actual date object
        return new Date();
      };
      return $delegate;
    });
  })
  .service('myService', function() {
    this.getDate = function() {
      return null; // w/o decoration we'll be returning null
    };
  })
  .controller('myController', function(myService) {
    var vm = this;
    vm.date = myService.getDate();
  });
```

```
<body ng-controller="myController as vm">
  <div ng-bind="vm.date | date:'fullDate'"></div>
</body>
```

> Saturday, August 6, 2016

ディレクティブをる

ディレクティブはサービスとにでき、そののいずれかをまたはきえることができます。ディレクティブは$ delegateのposition 0でアクセスされ、デコレータのnameパラメータには`Directive`サフィックスをがまれているがあることにしてください。

したがって、ディレクティブが`myDate`とばれるは、 `$delegate[0]`をして`myDateDirective`をしてアクセスできます。

は、ディレクティブがのをするなです。のを1でします。がなければ、いつもじがされます。

```
<body>
  <my-date></my-date>
</body>
```

```
angular.module('app', [])
  .config(function($provide) {
    $provide.decorator('myDateDirective', function($delegate, $interval) {
      var directive = $delegate[0]; // access directive

      directive.compile = function() { // modify compile fn
        return function(scope) {
          directive.link.apply(this, arguments);
          $interval(function() {
            scope.date = new Date(); // update date every second
          }, 1000);
        };
      };

      return $delegate;
    });
  })
  .directive('myDate', function() {
    return {
      restrict: 'E',
      template: '<span>Current time is {{ date | date:\'MM:ss\' }}</span>',
      link: function(scope) {
        scope.date = new Date(); // get current date
      }
    };
  });
```

> Current time is 08:33

フィルタをる

フィルタをデコレートするときは、パラメータに`Filter`サフィックスとををめるがあります。filterが`repeat`とばれる、デコレータパラメータは`repeatFilter`です。では、されたをnりしてがになるカスタムフィルタをデコレートします。また、フレームワークのにをえるがあるため、Angleのみみフィルタもにデコレートできますされません。

```
<body>
  <div ng-bind="'i can haz cheeseburger ' | repeat:2"></div>
</body>

angular.module('app', [])
  .config(function($provide) {
    $provide.decorator('repeatFilter', function($delegate) {
      return function reverse(input, count) {
        // reverse repeated string
        return ($delegate(input, count)).split('').reverse().join('');
      };
    });
  })
  .filter('repeat', function() {
    return function(input, count) {
      // repeat string n times
      return (input || '').repeat(count || 1);
    };
  });
```

> i can haz cheeseburger i can haz cheeseburger

> regrubeseehc zah nac i regrubeseehc zah nac i

オンラインでデコレータをむ https://riptutorial.com/ja/angularjs/topic/5255/デコレータ

30: デバッグ

Examples

マークアップでのなデバッグ

スコープのテストとモデルの

```
<div ng-app="demoApp" ng-controller="mainController as ctrl">
    {{$id}}
    <ul>
        <li ng-repeat="item in ctrl.items">
            {{$id}}<br/>
            {{item.text}}
        </li>
    </ul>
    {{$id}}
    <pre>
       {{ctrl.items | json : 2}}
    </pre>
</div>


angular.module('demoApp', [])
.controller('mainController', MainController);

function MainController() {
    var vm = this;
    vm.items = [{
        id: 0,
        text: 'first'
    },
    {
        id: 1,
        text: 'second'
    },
    {
        id: 2,
        text: 'third'
    }];
}
```

スコープのをするしいスコープがあるかどうかをするのにつことがあります。 `$scope.$id`は、しい$スコープがあるかどうかをするために、マークアップのどこでもでできます。

このでは、ulタグのがじスコープ$ id = 2であり、 `ng-repeat`にりしのしいスコープがあることがわかります。

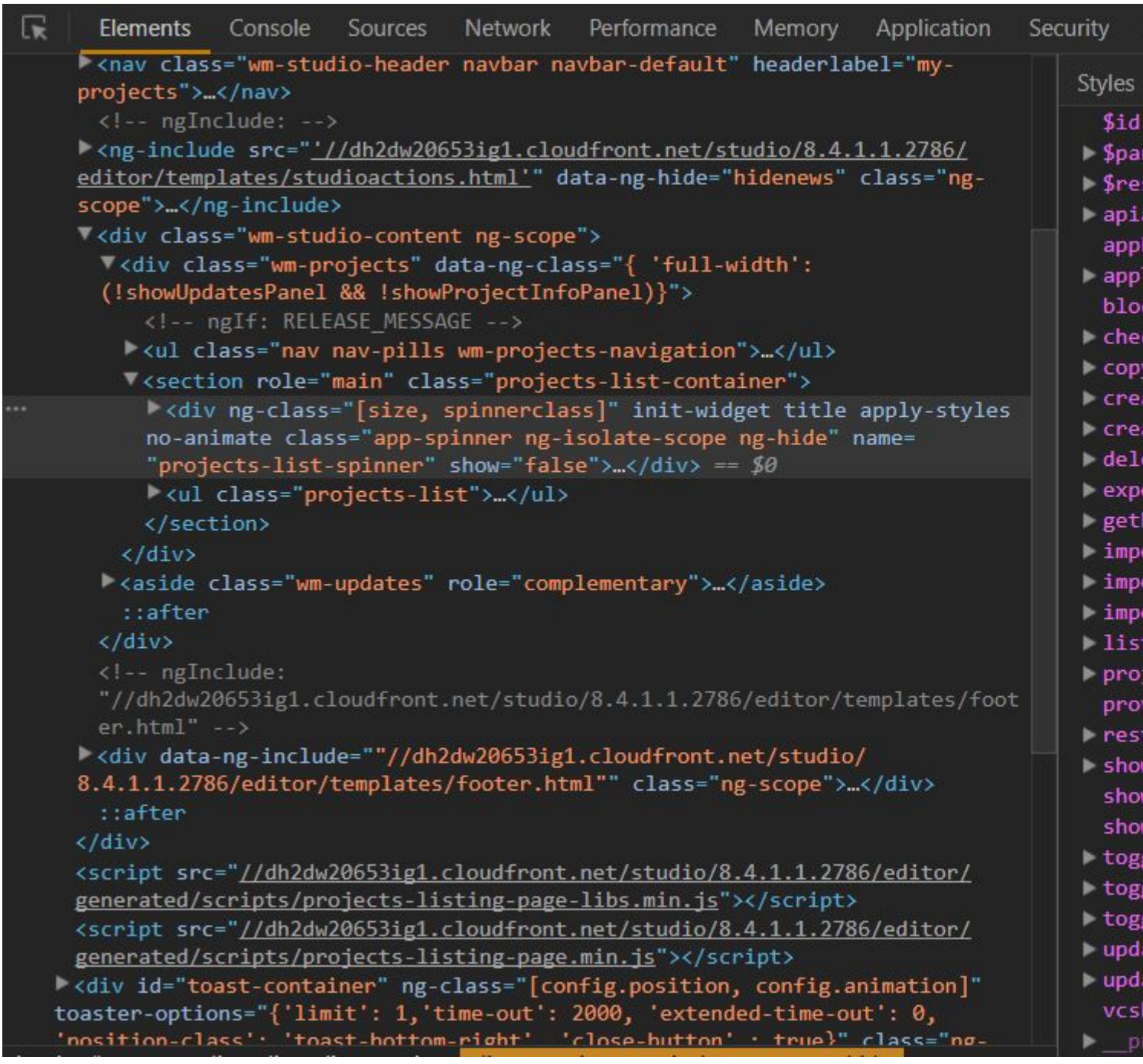
プリタグのモデルのは、モデルののデータをするのにです。 `json`フィルターはたのいフォーマットされたをします。そのタグのに`\n`がしくされるので、pre-tagがされます。

デモ

ng-inspect chrome extensionをする

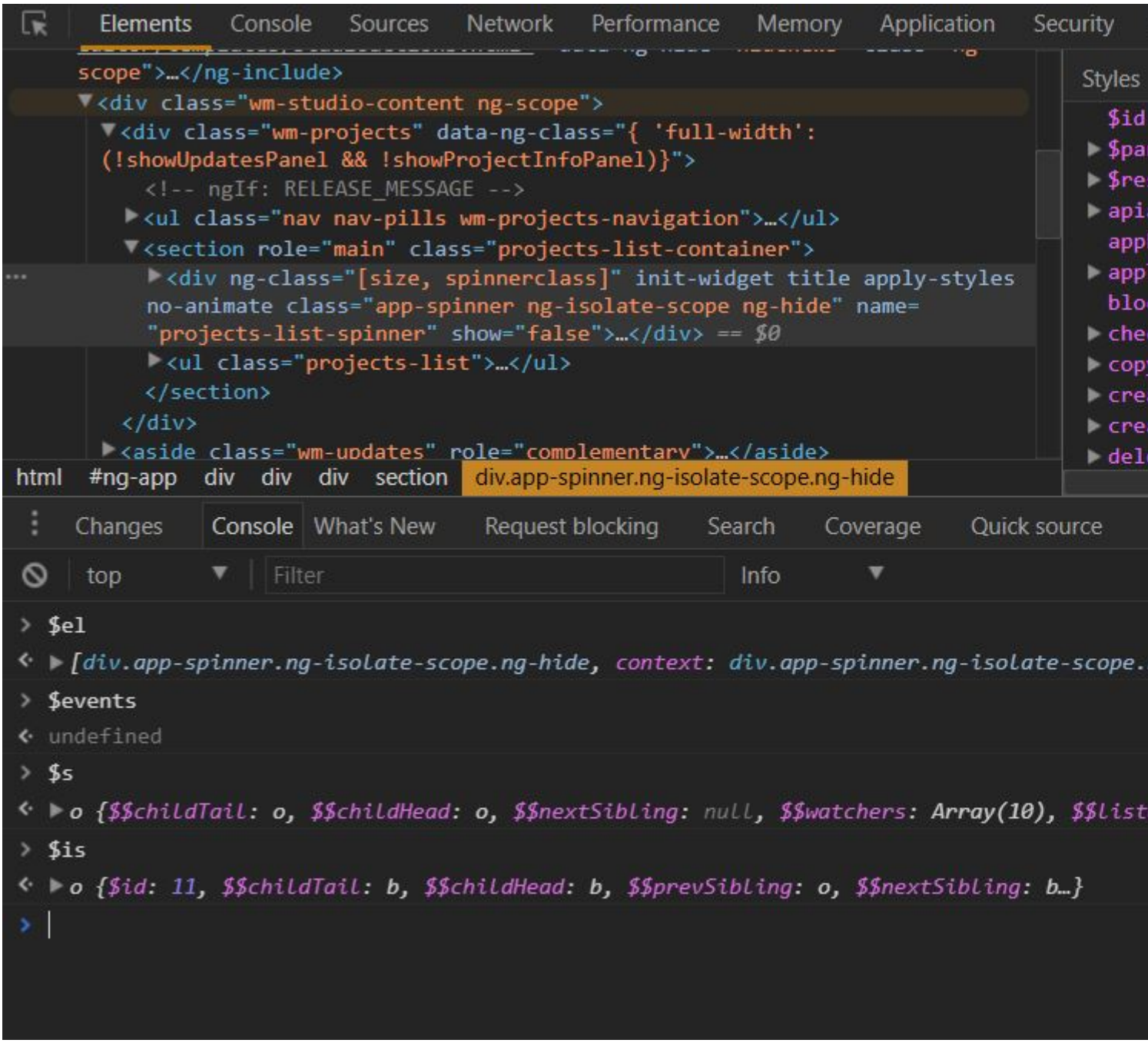
ng-inspectは、AngularJSアプリケーションをデバッグするためのChromeモジュールです。

エレメントパネルからノードをすると、スコープがng-inspectパネルにされます。



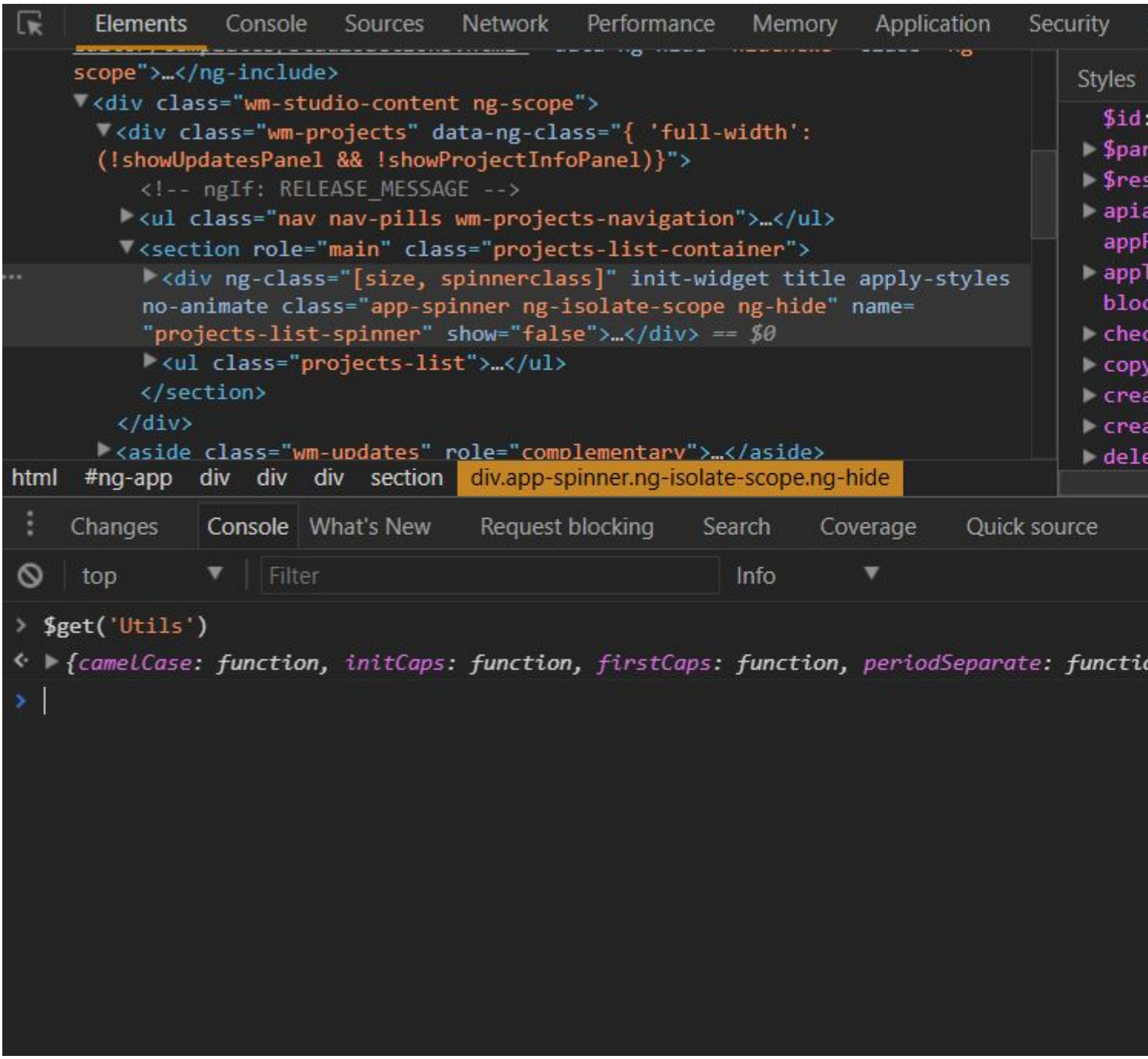
`scope/isolateScope`すばやくアクセスするためのグローバルはほとんどありませ`scope/isolateScope`。

```
$s      -- scope of the selected node
$is     -- isolateScope of the selected node
$el     -- jQuery element reference of the selected node (requiers jQuery)
$events -- events present on the selected node (requires jQuery)
```



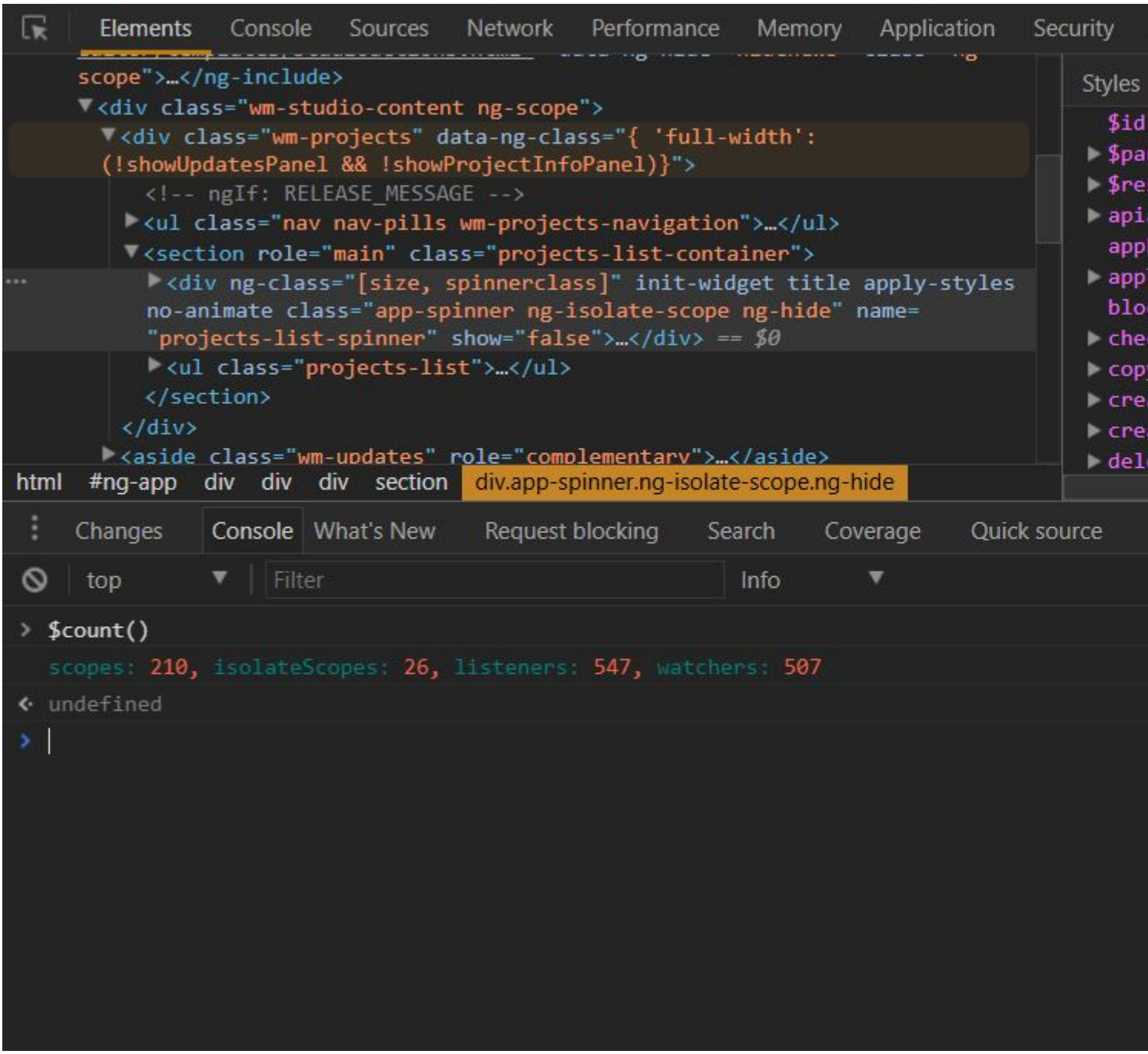
サービス/ににアクセスできます。

`$get()`をして、サービス/ファクトリのインスタンスをでします。



スコープ、アイソレートスコープ、ウォッチャー、およびリスナーのをアプリケーションでカウントすることによって、アプリケーションのパフォーマンスをできます。

スコープ、isolateScopes、ウォッチャー、およびリスナーのをするには、 `$count()` をします。



このは、debugInfoがなにのみします。

ng-inspectをここからダウンロードする

のをする

のあるアプリではすべてがスコープのりにあります。エレメントのスコープをできれば、アプリをデバッグするのはです。のにアクセスする

```
angular.element(myDomElement).scope();
e.g.
angular.element(document.getElementById('yourElementId')).scope()  //accessing by ID
```

コントローラのをする -

```
angular.element('[ng-controller=ctrl]').scope()
```

コンソールからDOMにアクセスするもう1つのなは、 "タブでクリックすることですに$ 0としてされます。

```
angular.element($0).scope();
```

オンラインでデバッグをむ https://riptutorial.com/ja/angularjs/topic/4761/デバッグ

31: パフォーマンスプロファイリング

Examples

プロファイリングにするすべて

プロファイリングとはですか

により、プロファイリングは、えば、プログラムのメモリまたはのさ、のの、またはびしのおよびをするプログラムのである。

それはなぜなのですか

プロファイリングはです。これは、のプログラムがのをやしていることをるまでにすることができないからです。プログラムのプロファイリングをすることなく、にしたかどうかはわかりません。

ツールとテクニック

1. Chromeのツール

   これには、プロファイリングにするなツールがまれています.JavaScriptファイル、CSSファイル、アニメーション、CPU、メモリリーク、ネットワーク、セキュリティなどのボトルネックをくつけることができます。

   タイムラインをし、わしくいスクリプトのイベントをします。いずれかがつかったは、JSプロファイラをにしてをやりして、どのJSがびされたか、それぞれのについてのをることができます。 きをむ...

2. FireBug Firefoxで

3. Dynatrace IEで

4. Batarang Chromeで

   それはしているが、アプリケーションのモデル、パフォーマンス、をするためにすることができるが、クロムブラウザのいアドオンです。なアプリケーションでうまくし、スコープがさまざまなレベルでをしているかのをることができます。それは、アクティブウォッチャー、をたり、アプリでコレクションをることができます。

5. ウォッチャー Chromeで

   のあるアプリでウォッチャーのをえる、きれいでシンプルなUI。

6. のコードをして、きアプリのウォッチャーのをでします @Words Like Jared Number of watchers

```
(function() {
    var root = angular.element(document.getElementsByTagName('body')),
        watchers = [],
        f = function(element) {
        angular.forEach(['$scope', '$isolateScope'], function(scopeProperty) {
            if(element.data() && element.data().hasOwnProperty(scopeProperty)) {
                angular.forEach(element.data()[scopeProperty].$$watchers, function(watcher) {
                watchers.push(watcher);
                });
            }
        });

        angular.forEach(element.children(), function(childElement) {
            f(angular.element(childElement));
        });
    };

    f(root);

    // Remove duplicate watchers
    var watchersWithoutDuplicates = [];
    angular.forEach(watchers, function(item) {
        if(watchersWithoutDuplicates.indexOf(item) < 0) {
            watchersWithoutDuplicates.push(item);
        }
    });
    console.log(watchersWithoutDuplicates.length);
})();
```

7. さまざまなをしてアプリケーションのプロファイルをするオンラインツールやウェブサイトがいくつかあります。

   そのようなサイトの1つがhttps://www.webpagetest.org/です。

   これにより、のブラウザIEとChromeとのコンシューマをして、ののからのウェブサイトテストをできます。シンプルなテストをしたり、マルチステップトランザクション、ビデオキャプチャ、コンテンツブロッキングなどのなテストをすることができます。

のステップ

プロファイリングをしました。それはのりのをもたらすだけです。のは、をにアクションにえてアプリケーションをすることです。なテクニックでアプリのパフォーマンスをさせるについては、このドキュメントをしてください 。

ハッピーコーディング:)

オンラインでパフォーマンスプロファイリングをむ https://riptutorial.com/ja/angularjs/topic/7033/パフォーマンスプロファイリング

32: ビルトインディレクティブ

Examples

- テキストと

このでは、inputに`type="text"`および`type="number"`をすると、がどのようにされるかをしています。のコントロ―ラとビュ―をえてみましょう。

コントロ―ラ

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

app.controller('ctrl', function($scope) {
    $scope.textInput = {
        value: '5'
    };
    $scope.numberInput = {
        value: 5
    };
});
```

ビュ―

```
<div ng-app="app" ng-controller="ctrl">
    <input type="text" ng-model="textInput.value">
    {{ textInput.value + 5 }}
    <input type="number" ng-model="numberInput.value">
    {{ numberInput.value + 5 }}
</div>
```

- テキストにバインドされたで+をすると、はをしの、に55 *します。
- にバインドされたで+をすると、はのをし2の、に*します* 。

* - ユ―ザ―がフィ―ルドのをするまでです。その、がそれにじてされます。

ngRepeat

`ng-repeat`は、またはオブジェクトをりしできるようにする、Angularのみみディレクティブです。コレクションのアイテムにしてをりすことができます。

ng-をりす

```
<ul>
    <li ng-repeat="item in itemCollection">
       {{item.Name}}
    </li>
</ul>
```

item =コレクションの々のアイテム
itemCollection =する

ng-オブジェクトをりす

```
<ul>
    <li ng-repeat="(key, value) in myObject">
       {{key}} : {{value}}
    </li>
</ul>
```

key =プロパティ
value =プロパティの
myObject =しているオブジェクト

あなたの**ng-repeat**をユ―ザでフィルタリングする

```
<input type="text" ng-model="searchText">
<ul>
    <li ng-repeat="string in stringArray | filter:searchText">
       {{string}}
    </li>
</ul>
```

searchText =ユ―ザ―がリストをフィルタリングするテキスト
stringArray =の `['string', 'array']`

フィルター―に`as aliasName`というエイリアスをりてることによって、のでフィルター―されたをまたはすることもできます。

```
<input type="text" ng-model="searchText">
<ul>
    <li ng-repeat="string in stringArray | filter:searchText as filteredStrings">
       {{string}}
    </li>
</ul>
<p>There are {{filteredStrings.length}} matching results</p>
```

ng-repeat-startおよび**ng-repeat-end**

とをしてのDOMをりすには、 `ng-repeat-start`および`ng-repeat-end`ディレクティブをします。

```
<ul>
    <li ng-repeat-start="item in [{a: 1, b: 2}, {a: 3, b:4}]">
        {{item.a}}
    </li>
    <li ng-repeat-end>
        {{item.b}}
    </li>
</ul>
```

- 1
- 2
- 3
- 4

に`ng-repeat-start`を`ng-repeat-end`することがです。

`ng-repeat`は、これらのものにします

	タイプ	
`$index`		ののインデックスとしくなります$ index === 0はのでとされます; `$first`してください
`$first`	ブール	のでとする
`$last`	ブール	のでとする
`$middle`	ブール	が`$first`と`$last`にある、trueにされます。
`$even`	ブール	のでとします `$index%2===0`
`$odd`	ブール	のでとされます `$index%2===1`と

パフォーマンスの

になコレクションをする、 `ngRepeat`レンダリングがくなることがあります。

コレクションのオブジェクトにプロパティがある、オブジェクトではなくでに`track by`するがあります。これはデフォルトのです。がしないは、にみ`$index`でき`$index`。

```
<div ng-repeat="item in itemCollection track by item.id">
<div ng-repeat="item in itemCollection track by $index">
```

ngRepeatの

ngRepeatはにしたスコープをするので、スコープをりしでアクセスするがあるはがです。

ここでは、 ngRepeatのクリックイベントからスコープのをするをすなをします。

```
scope val:  {{val}}<br/>
ctrlAs val: {{ctrl.val}}
<ul>
    <li ng-repeat="item in itemCollection">
        <a href="#" ng-click="$parent.val=item.value; ctrl.val=item.value;">
            {{item.label}} {{item.value}}
        </a>
    </li>
</ul>

$scope.val = 0;
this.val = 0;

$scope.itemCollection = [{
    id: 0,
    value: 4.99,
    label: 'Football'
},
{
    id: 1,
    value: 6.99,
    label: 'Baseball'
},
{
    id: 2,
    value: 9.99,
    label: 'Basketball'
}];
```

ng-clickでval = item.valueだけがあった、されたスコープのためにスコープのvalはされません。そのため、スコープは$parentまたはcontrollerAsたとえば、 ng-controller="mainController as ctrl" でng-controller="mainController as ctrl"ます。

ネストされたng-repeat

ネストされたng-repeatをすることもできます。

```
<div ng-repeat="values in test">
    <div ng-repeat="i in values">
      [{{$parent.$index}},{{$index}}] {{i}}
    </div>
</div>

var app = angular.module("myApp", []);
app.controller("ctrl", function($scope) {
  $scope.test = [
    ['a', 'b', 'c'],
    ['d', 'e', 'f']
  ];
});
```

ここでは、ng-repeatのng-repeatのインデックスにアクセスするために、 `$parent.$index`できます。

ngShowとngHide

`ng-show`ディレクティブは、されたがかかについてHTMLをまたはにします。のがであれば、それはれます。ならばそれはされます。

`ng-hide`ディレクティブはています。ただし、がであればHTMLがされます。がなら、それはそれをすでしょう。

JSBinの

コントローラ

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

angular.module('app')
  .controller('ExampleController', ExampleController);

function ExampleController() {

  var vm = this;

  //Binding the username to HTML element
  vm.username = '';

  //A taken username
  vm.taken_username = 'StackOverflow';

}
```

ビュー

```
<section ng-controller="ExampleController as main">

    <p>Enter Password</p>
    <input ng-model="main.username" type="text">

    <hr>

    <!-- Will always show as long as StackOverflow is not typed in -->
    <!-- The expression is always true when it is not StackOverflow -->
    <div style="color:green;" ng-show="main.username != main.taken_username">
      Your username is free to use!
    </div>

    <!-- Will only show when StackOverflow is typed in -->
    <!-- The expression value becomes falsy -->
    <div style="color:red;" ng-hide="main.username != main.taken_username">
      Your username is taken!
    </div>

    <p>Enter 'StackOverflow' in username field to show ngHide directive.</p>
```

```
</section>
```

ngOptions

ngOptionsは、モデルにされるからアイテムをするためのhtmlドロップダウンボックスのをにするです。 ngOptionsは、ngOptionsのをしてられたまたはオブジェクトをして、 <select>の<option>のリストをにするためにされます。

ng-optionsすると、マークアップをselectタグだけにらすことができ、ディレクティブはじselectをします

```
<select ng-model="selectedFruitNgOptions"
        ng-options="curFruit as curFruit.label for curFruit in fruit">
</select>
```

ng-repeatをしてselectオプションをselectもありますが、for ng-repeat forEachだけをループさせるようななのためにng-repeatをすることはおめしません。 ng-optionsはにselectタグオプションをselectためのものです。

のではng-repeatをしています

```
<select ng-model="selectedFruit">
    <option ng-repeat="curFruit in fruit" value="{{curFruit}}">
        {{curFruit.label}}
    </option>
</select>
```

な

のもいくつかのをめてにしています。

こののデータモデル

```
$scope.fruit = [
    { label: "Apples", value: 4, id: 2 },
    { label: "Oranges", value: 2, id: 1 },
    { label: "Limes", value: 4, id: 4 },
    { label: "Lemons", value: 5, id: 3 }
];
```

```
<!-- label for value in array -->
<select ng-options="f.label for f in fruit" ng-model="selectedFruit"></select>
```

にされるオプションタグ

```
 <option value="{ label: "Apples", value: 4, id: 2 }"> Apples </option>
```

f.labelは<option>ラベルになり、にはオブジェクトがまれます。

な

```
<!-- select as label for value in array -->
<select ng-options="f.value as f.label for f in fruit" ng-model="selectedFruit"></select>
```

にされるオプションタグ

```
 <option value="4"> Apples </option>
```

`f.value` 4はラベルがじであるののになります。

な

```
<!-- label group by group for value in array -->
<select ng-options="f.label group by f.value for f in fruit" ng-
model="selectedFruit"></select>
```

にされるオプションタグ

```
<option value="{ label: "Apples", value: 4, id: 2 }"> Apples </option>
```

オプションはその`value`づいてグループされ`value`。 じ`value`をつオプションは1つのカテゴリにされます

な

```
<!-- label disable when disable for value in array -->
<select ng-options="f.label disable when f.value == 4 for f in fruit" ng-
model="selectedFruit"></select>
```

にされるオプションタグ

```
<option disabled="" value="{ label: "Apples", value: 4, id: 2 }"> Apples </option>
```

`disable when f.value==4` 、 `disable when f.value==4`ののため、 "Apples"と "Limes"はになりますできません。 `value=4`すべてのオプションはにする

な

```
<!-- label group by group for value in array track by trackexpr -->
<select ng-options="f.value as f.label group by f.value for f in fruit track by f.id" ng-
model="selectedFruit"></select>
```

にされるオプションタグ

```
<option value="4"> Apples </option>
```

`trackBy`をしているときはなはありませんが、Angularはではなく`id`によるをします。

な

```
<!-- label for value in array | orderBy:orderexpr track by trackexpr -->
<select ng-options="f.label for f in fruit | orderBy:'id' track by f.id" ng-
model="selectedFruit"></select>
```

にされるオプションタグ

```
<option disabled="" value="{ label: "Apples", value: 4, id: 2 }"> Apples </option>
```

`orderBy`はAngularJSのフィルタで、オプションをデフォルトにべるので、 `id` = 1であるため、このの「オレンジ」は1にされます。

な

`ng-options`をむすべての**`<select>`**は**`ng-model`**されているが**`ng-options`**。

ngModel

ng-modelでは、のタイプのフィールドにをバインドできます。 `{{myAge}}` 、をしてをできます。

```
<input type="text" ng-model="myName">
<p>{{myName}}</p>
```

フィールドにするか、らかのですると、パラグラフのがにされます。

このng-modelは、 `$scope.myName`としてコントローラでできます。 `controllerAs`をしている

```
<div ng-controller="myCtrl as mc">
    <input type="text" ng-model="mc.myName">
    <p>{{mc.myName}}</p>
</div>
```

ng-controllerでされているコントローラのエイリアスをng-modelにあらかじめすることによって、コントローラのをするがあります。このでは、ng-modelをするためにコントローラに`$scope`をするはなく、コントローラので`this.myName`としてをできます。

クラス

ユーザーのステータスをするがあり、できるCSSクラスがいくつかあるとします。 Angularは、をむオブジェクトリストをできるいくつかのクラスのリストからすることをににします。は、のにづいてしいクラスをすることができます。

オブジェクトにはキーとのペアがまれているがあります。キーは、きがtrueとされたときにされるクラスです。

```
<style>
    .active { background-color: green; color: white; }
    .inactive { background-color: gray; color: white; }
    .adminUser { font-weight: bold; color: yellow; }
    .regularUser { color: white; }
</style>

<span ng-class="{
    active: user.active,
    inactive: !user.active,
    adminUser: user.level === 1,
    regularUser: user.level === 2
}">John Smith</span>
```

Angularは`$scope.user`オブジェクトをチェックして、 `active`ステータスと`level`をします。これらののにじて、Angularはスタイルを`<span>`します。

ngIf

`ng-if`は**`ng-show`**とていますが、に**`ng-show`**するのではなく、DOMからをまたはします。 Angular 1.1.5では、ng-Ifがされました。 ng-ifディレクティブを1.1.5のバージョンでできます。これはAngularがされた`ng-if`ののダイジェストをしないため、になデータバインディングの`ng-if` 、Angularのをする`ng-if`です。

`ng-show`とはなり、 `ng-if`ディレクティブはプロトタイプをするスコープをします。つまり、スコープのプリミティブをすることはにはされません。スコープのプリミティブをするには、スコープの`$parent`プロパティをするがあります。

JavaScript

```
angular.module('MyApp', []);

angular.module('MyApp').controller('myController', ['$scope', '$window', function
myController($scope, $window) {
    $scope.currentUser= $window.localStorage.getItem('userName');
}]);
```

ビュー

```
<div ng-controller="myController">
    <div ng-if="currentUser">
        Hello, {{currentUser}}
    </div>
    <div ng-if="!currentUser">
        <a href="/login">Log In</a>
```

```
        <a href="/register">Register</a>
    </div>
</div>
```

currentUserがののDOM

```
<div ng-controller="myController">
    <div ng-if="currentUser">
        Hello, {{currentUser}}
    </div>
    <!-- ng-if: !currentUser -->
</div>
```

currentUserがののDOM

```
<div ng-controller="myController">
    <!-- ng-if: currentUser -->
    <div ng-if="!currentUser">
        <a href="/login">Log In</a>
        <a href="/register">Register</a>
    </div>
</div>
```

の

ngIfディレクティブは、にtrueまたはfalseをすがあるもけれます。

```
<div ng-if="myFunction()">
    <span>Span text</span>
</div>
```

スパンテキストは、がtrueをすにのみされます。

```
$scope.myFunction = function() {
    var result = false;
    // Code to determine the boolean value of result
    return result;
};
```

Angularのように、このはののをけれます。

ngMouseenterとngMouseleave

ng-mouseenterおよびng-mouseleaveディレクティブは、イベントをしたり、DOMのにりするときにCSSのスタイルをしたりするのにです。

ng-mouseenterディレクティブは、マウスがマウスイベントをするをしますこのディレクティブがするDOMのにマウスポインタをng-mouseenterと

HTML

```
<div ng-mouseenter="applyStyle = true" ng-class="{'active': applyStyle}">
```

のでは、ユーザーがdivにマウスをapplyStyleがtrueにわり、 ng-classで.active CSSクラスがされます。

ng-mouseleaveディレクティブは、マウスexitイベントの1つのをしますこのディレクティブがするDOMからマウスカーソルをしたとき

HTML

```
<div ng-mouseenter="applyStyle = true" ng-mouseleaver="applyStyle = false" ng-
class="{'active': applyStyle}">
```

のをすると、ユーザーがマウスポインタをdivからした.activeクラスがされるようになりました。

ngDisabled

このディレクティブは、ののにづいてイベントをするのにです。

ng-disabledディレクティブは、かのかをするをけれ、します。

ng-disabledは、 inputにdisabledをきでするためにされます。

HTML

```
<input type="text" ng-model="vm.name">

<button ng-disabled="vm.name.length===0" ng-click="vm.submitMe">Submit</button>
```

vm.name.length===0は、 inputのさが0のはtrueに、ボタンをにするはng-clickクリックイベントをさせない

ngDblclick

ng-dblclickディレクティブは、ダブルクリックイベントをDOMにバインドするにです。

このディレクティブはをけります

HTML

```
<input type="number" ng-model="num = num + 1" ng-init="num=0">

<button ng-dblclick="num++">Double click me</button>
```

のでは、ボタンをダブルクリックすると、 `input`されているがインクリメントされます。

みディレクティブチ―トシ―ト

`ng-app` AngularJSセクションをします。

`ng-init`デフォルトのをします。

`ng-bind` {{}}テンプレ―トのわりにします。

`ng-bind-template`のをビュ―にバインドします。

`ng-non-bindable`デ―タがバインドでないことをします。

`ng-bind-html` HTMLのHTMLプロパティをバインドします。

`ng-change`ユ―ザ―がをしたときに、されたをします。

`ng-checked`ボックスをします。

`ng-class` cssクラスをにします。

`ng-cloak` AngularJSがをするまでコンテンツをしないようにします。

`ng-click`がクリックされたときにメソッドまたはをします。

`ng-controller`ビュ―にコントロ―ラクラスをします。

`ng-disabled`フォ―ムのdisabledプロパティをします

`ng-form`する

`ng-href` AngularJSをhrefににバインドします。

`ng-include`HTMLフラグメントをフェッチ、コンパイル、およびペ―ジへのみみ`ng-include`します。

`ng-if`にじてDOMのをまたはする

`ng-switch`するにづいてきでをりえます。

`ng-model` input、select、textareaなどのをmodelプロパティでバインドします。

`ng-readonly`にみりをするためにします。

`ng-repeat`しいテンプレ―トをするために、コレクションのをル―プするときにします。

`ng-selected`にされたオプションをするためにします。

`ng-show/ng-hide`にづいてを/ `ng-show/ng-hide`します。

`ng-src` AngularJSをsrcににバインドします。

`ng-submit`をonsubmitイベントにバインドします。

`ng-value`を`ng-value`バインドします。

`ng-required`をonsubmitイベントにバインドします。

`ng-style` HTMLにCSSスタイルをします。

`ng-pattern`パターンバリデーターをngModelにします。

`ng-maxlength` maxlengthバリデーターをngModelにします。

`ng-minlength`にminlengthバリデータをします。

`ng-classeven` ngRepeatとみわせてし、にのみです。

`ng-classodd` ngRepeatとみわせてし、にのみです。

`ng-cut`カットイベントのカスタムをする`ng-cut`します。

`ng-copy`コピーイベントのカスタムをするためにします。

`ng-paste`ペーストイベントのカスタムをするためにします。

`ng-options`ののリストをにするためにされます。

`ng-list`されたりについてをリストにするためにします。

`ng-open` ngOpenのがである、のオープンをする`ng-open`されます。

ソースし

ngClick

`ng-click`ディレクティブは、クリックイベントをDOMにけます。

`ng-click`ディレクティブをすると、DOMのがクリックされたときのカスタムをできます。

これは、ボタンでクリックイベントをアタッチし、コントローラでするにです。

このディレクティブは、イベントオブジェクトを`$event`としてできるをけります

HTML

```
<input ng-click="onClick($event)">Click me</input>
```

コントローラ

```
.controller("ctrl", function($scope) {
    $scope.onClick = function(evt) {
        console.debug("Hello click event: %o ",evt);
    }
})
```

HTML

```
<button ng-click="count = count + 1" ng-init="count=0">
  Increment
</button>
<span>
  count: {{count}}
</span>
```

HTML

```
<button ng-click="count()" ng-init="count=0">
  Increment
</button>
<span>
  count: {{count}}
</span>
```

コントローラ

```
...

$scope.count = function(){
    $scope.count = $scope.count + 1;
}
...
```

ボタンをクリックすると、 `onClick`をびすと、「Hello click event」とそれにくィベントオブジェクトがされます。

ngRequired

`ng-required`は、の`required`をまたはし`input`。 これにより、 `input require`キーがまたはになり`input`。

`input`がでないをつがあるかどうかをオプションでするためにされます。このディレクティブは、なHTMLフォームのをするにちます。

HTML

```
<input type="checkbox" ng-model="someBooleanValue">
<input type="text" ng-model="username" ng-required="someBooleanValue">
```

ng-model-options

ng-model-optionsのデフォルトのにすることができますng-model 、このディレクティブはNG-モデルがされたときにするとデバウンスをするイベントをすることができます。

このディレクティブは、オブジェクトまたはスコープへのにされるをけれます。

```
<input type="text" ng-model="myValue" ng-model-options="{'debounce': 500}">
```

のでは、の500ミリのデバウンスをするmyValueユーザがしわったに、モデル500ミリをするとなる 、 input 、 つまりmyValueをえました。

なオブジェクトのプロパティ

1. updateOn にバインドするイベントをします。

   ```
   ng-model-options="{ updateOn: 'blur'}" // will update on blur
   ```

2. debounce モデルにけてミリのをする

   ```
   ng-model-options="{'debounce': 500}" // will update the model after 1/2 second
   ```

3. allowInvalid デフォルトのフォームをして、モデルになをするブールのフラグです。デフォルトでは、これらのはundefinedとしてわれます。

4. getterSetter ng-modelをモデルではなくゲッター/セッターとしてうかどうかをすbooleanフラグ。がされ、モデルがされます。

   ```
   <input type="text" ng-model="myFunc" ng-model-options="{'getterSetter': true}">

   $scope.myFunc = function() {return "value";}
   ```

5. timezone がdateまたはtime、モデルのタイムゾーンをします。タイプ

クローク

ngCloakディレクティブは、アプリケーションがロードされているに、 ngCloak HTMLテンプレートがのコンパイルされていないフォームでブラウザによってされないようにするためにされます。 - ソースを

HTML

```
<div ng-cloak>
  <h1>Hello {{ name }}</h1>
</div>
```

ngCloakをbodyにできますが、ブラウザビューのプログレッシブレンダリングをにするために、ページのさなにのngCloakディレクティブをすることをおめします。

ngCloakディレクティブにはパラメータはありません。

ちらつき

インクルージョン

ng-includeをすると、ページののコントロールをのコントローラにできます。そのコンポーネントのさが、コントローラのすべてのロジックをカプセルするようになっているので、これをうことができます。

はのとおりです。

```
  <div ng-include
       src="'/gridview'"
       ng-controller='gridController as gc'>
  </div>
```

/gridviewは、Webサーバによってのなurlとしてされるがあることにしてください。

また、 src -attributeはAngularをけれます。これは、またはびし、たとえばこののようにであるがあります。この、ソース**URL**をでむようにするがあります 。そのため、としてされます。これはのなです。

/gridview htmlでは、 gridControllerをページのりにgridControllerれているかのようにできます。

```
<div class="row">
  <button type="button" class="btn btn-default" ng-click="gc.doSomething()"></button>
</div>
```

ngSrc

srcで{{hash}}ようなアングルマークアップをするとしくしません。ブラウザは、リテラルテキストとURLからフェッチする{{hash}}がをするまで{{hash}}。 ng-srcディレクティブはイメージタグののsrcをオーバーライドし、をします

```
<div ng-init="pic = 'pic_angular.jpg'">
    <h1>Angular</h1>
    <img ng-src="{{pic}}">
</div>
```

ngPattern

ng-patternディレクティブは、パターンにされるをけり、そのパターンをしてテキストをします。

<input>がng-modelがなIPアドレスであるときににになるようにしたいとしましょう。

テンプレート

```
<input type="text" ng-model="ipAddr" ng-pattern="ipRegex" name="ip" required>
```

コントローラ

```
$scope.ipRegex = /\b(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\b/;
```

ngValue

ng-repeat ngValueでにされるのは、ngRepeatをしてラジオボタンのリストをにするにです

```
<script>
   angular.module('valueExample', [])
     .controller('ExampleController', ['$scope', function($scope) {
       $scope.names = ['pizza', 'unicorns', 'robots'];
       $scope.my = { favorite: 'unicorns' };
     }]);
</script>
 <form ng-controller="ExampleController">
   <h2>Which is your favorite?</h2>
     <label ng-repeat="name in names" for="{{name}}">
       {{name}}
       <input type="radio"
              ng-model="my.favorite"
              ng-value="name"
              id="{{name}}"
              name="favorite">
     </label>
   <div>You chose {{my.favorite}}</div>
 </form>
```

〈 plnkr

ngCopy

ngCopyディレクティブは、コピーイベントでされるをします。

ユーザーがデータをコピーしないようにする

```
<p ng-copy="blockCopy($event)">This paragraph cannot be copied</p>
```

コントローラ

```
$scope.blockCopy = function(event) {
    event.preventDefault();
    console.log("Copying won't work");
}
```

ペースト

ngPasteディレクティブは、ユーザーがコンテンツをりけるときにするカスタムをします

```
<input ng-paste="paste=true" ng-init="paste=false" placeholder='paste here'>
pasted: {{paste}}
```

ngHref

hrefのにがある、hrefのわりにngHrefがされます。 ngHrefディレクティブは、タグ、タグなどのhrefをして、htmlタグののhrefをオーバーライドします。

ngHrefディレクティブは、AngularJSがコードをするにユーザーがリンクをクリックしてもリンクがれていないことをします。

1

```
<div ng-init="linkValue = 'http://stackoverflow.com'">
    <p>Go to <a ng-href="{{linkValue}}">{{linkValue}}</a>!</p>
</div>
```

2このでは、ボックスからhrefをにし、hrefとしてロードします。

```
<input ng-model="value" />
<a id="link" ng-href="{{value}}">link</a>
```

3

```
<script>
angular.module('angularDoc', [])
.controller('myController', function($scope) {
  // Set some scope value.
  // Here we set bootstrap version.
  $scope.bootstrap_version = '3.3.7';

  // Set the default layout value
  $scope.layout = 'normal';
});
</script>
<!-- Insert it into Angular Code -->
<link rel="stylesheet" ng-href="//maxcdn.bootstrapcdn.com/bootstrap/{{ bootstrap_version
}}/css/bootstrap.min.css">
<link rel="stylesheet" ng-href="layout-{{ layout }}.css">
```

ngList

`ng-list`ディレクティブは、られたをテキストからのに、またはそのにするためにされます。

`ng-list`ディレクティブは、デフォルトり", " コンマスペースをします。

あなたはりてることにより、でのりをすることができ`ng-list`このようなり`ng-list="; "`。

この、りはセミコロンとそれにくスペースにされます。

デフォルトでは、 `ng-list`は`ng-trim`というがあり、trueにされています。 `ng-trim`すると、デリミ

タのがされます。デフォルトでは、 `ng-list ng-trim="false"`しないり、 `ng-list`はをしません。

```
angular.module('test', [])
  .controller('ngListExample', ['$scope', function($scope) {
    $scope.list = ['angular', 'is', 'cool!'];
}]);
```

りはのようにされます; 。ボックスのモデルは、スコープにされたにされます。

```
  <body ng-app="test" ng-controller="ngListExample">
    <input ng-model="list" ng-list="; " ng-trim="false">
  </body>
```

ボックスにはコンテンツがされます `angular; is; cool!`

オンラインでビルトインディレクティブをむ https://riptutorial.com/ja/angularjs/topic/706/ビルトインディレクティブ

33: フィルタ

Examples

あなたののフィルター—

フィルタは、かがペ—ジにどのようにされるかをする、またはをフィルタリングするためにできるなタイプの、または`ng-repeat`アクションです。フィルタをするには、 `app.filter()`メソッドをびし、とをします。のについては、のをしてください。

たとえば、をすべてに`.toUpperCase()` javascriptのラッパー—`.toUpperCase()`するフィルタをします。

```
var app = angular.module("MyApp", []);

// just like making a controller, you must give the
// filter a unique name, in this case "toUppercase"
app.filter('toUppercase', function(){
    // all the filter does is return a function,
    // which acts as the "filtering" function
    return function(rawString){
        // The filter function takes in the value,
        // which we modify in some way, then return
        // back.
        return rawString.toUpperCase();
    };
});
```

でがこっているのかをしくてみましょう。

まず、コントロ—ラ—のような "toUppercase"というフィルター—をします。 `app.filter(...)`。に、そのフィルタのがのフィルタをします。このは、フィルタリングされるオブジェクトであるのオブジェクトをり、そのオブジェクトのフィルタリングされたバ—ジョンをすがあります。

このでは、フィルタにされるオブジェクトがであるとしているため、ににのみフィルタをすることがわかります。つまり、オブジェクトをル—プしての、のすべてのをにするフィルタをさらにすることができます。

に、しいフィルタをにしましょう。たちのフィルタは、テンプレ—トまたはJavaScriptAngularの2つのでできます。

Javascript

のある`$filter`オブジェクトをコントロ—ラにし、それをってそのをってフィルタをします。

```
app.controller("MyController", function($scope, $filter){
    this.rawString = "Foo";
```

```
    this.capsString = $filter("toUppercase")(this.rawString);
});
```

HTML

のディレクティブのは、ののにディレクティブのフィルタのあとにパイプ | をけてします。たとえば、 `MyController`というコントロ―ラがあり、そのコントロ―ラのとして`rawString`というがあるとします。

```
<div ng-controller="MyController as ctrl">
    <span>Capital rawString: {{ ctrl.rawString | toUppercase }}</span>
</div>
```

エディタのノ―ト *Angular*には、をむのフィルタがみまれています。また、「*toUppercase*」フィルタは、フィルタのをにすデモとしてのみ されますが、ののをするはありません。

をするカスタムフィルタ

フィルタのなは、からをすることです。このでは、をし、そのにつかったすべてのnullをりき、をします。

```
function removeNulls() {
    return function(list) {
        for (var i = list.length - 1; i >= 0; i--) {
            if (typeof list[i] === 'undefined' ||
                    list[i] === null) {
                list.splice(i, 1);
            }
        }
        return list;
    };
}
```

これは、HTMLのようにされる

```
{{listOfItems | removeNulls}}
```

またはコントロ―ラのような

```
listOfItems = removeNullsFilter(listOfItems);
```

をフォ―マットするカスタムフィルタ

フィルター―ののは、のをフォ―マットすることです。このでは、をし、なのブ―ルをします。

```
function convertToBooleanValue() {
    return function(input) {
```

```
        if (typeof input !== 'undefined' &&
                input !== null &&
                (input === true || input === 1 || input === '1' || input
                        .toString().toLowerCase() === 'true')) {
            return true;
        }
        return false;
    };
}
```

どのようなHTMLでこのようにされる

```
{{isAvailable | convertToBooleanValue}}
```

またはコントローラのような

```
var available = convertToBooleanValueFilter(isAvailable);
```

でフィルタをする

このは、カスタムフィルタをとせずにでディープフィルタをするをすためにったものです。

コントローラ

```
(function() {
  "use strict";
  angular
    .module('app', [])
    .controller('mainCtrl', mainCtrl);

  function mainCtrl() {
    var vm = this;

    vm.classifications = ["Saloons", "Sedans", "Commercial vehicle", "Sport car"];
    vm.cars = [
       {
          "name":"car1",
          "classifications":[
             {
                "name":"Saloons"
             },
             {
                "name":"Sedans"
             }
          ]
       },
       {
          "name":"car2",
          "classifications":[
             {
                "name":"Saloons"
             },
             {
                "name":"Commercial vehicle"
             }
```

```
            ]
        },
        {
            "name":"car3",
            "classifications":[
                {
                    "name":"Sport car"
                },
                {
                    "name":"Sedans"
                }
            ]
        }
    ];
  }
})();
```

```
<body ng-app="app" ng-controller="mainCtrl as main">
  Filter car by classification:
  <select ng-model="classificationName"
          ng-options="classification for classification in main.classifications"></select>
  <br>
  <ul>
    <li ng-repeat="car in main.cars |
                   filter: { classifications: { name: classificationName } } track by $index"
                  ng-bind-template="{{car.name}} - {{car.classifications | json}}">
    </li>
  </ul>
</body>
```

なDEMOをしてください。

コントローラーまたはサービスでのフィルター

$filterすることで、Angularモジュールのみフィルタは、コントローラ、サービス、ディレクティブ、またはのフィルタでもできます。

```
angular.module("app")
  .service("users", usersService)
  .controller("UsersController", UsersController);

function usersService () {
  this.getAll = function () {
    return [{
      id: 1,
      username: "john"
    }, {
      id: 2,
      username: "will"
    }, {
      id: 3,
      username: "jack"
    }];
  };
}

function UsersController ($filter, users) {
```

```
  var orderByFilter = $filter("orderBy");

  this.users = orderByFilter(users.getAll(), "username");
  // Now the users are ordered by their usernames: jack, john, will

  this.users = orderByFilter(users.getAll(), "username", true);
  // Now the users are ordered by their usernames, in reverse order: will, john, jack
}
```

ng-repeatのからフィルタリングされたリストにアクセスする

によっては、 `ng-repeat` からフィルタのにアクセスし、おそらくされたのをすことができます。これは、 `ng-repeat as [variablename]as [variablename]` できます。

```
<ul>
  <li ng-repeat="item in vm.listItems | filter:vm.myFilter as filtered">
    {{item.name}}
  </li>
</ul>
<span>Showing {{filtered.length}} of {{vm.listItems.length}}</span>
```

オンラインでフィルタをむ https://riptutorial.com/ja/angularjs/topic/1401/フィルタ

34: フォーム

Examples

フォーム

Angularのみの1つは、クライアントのフォームです。

なフォームをい、のあるjQueryスタイルのをするがありますが、がかかり、かいことがあります。 Angularでは、プロのインタラクティブフォームをにできます。

ng-modelディレクティブはフィールドとのバインディングをし、は**novalidate**もフォームにされ、ブラウザがネイティブをわないようにします。

したがって、なフォームはのようになります。

```
<form name="form" novalidate>
  <label name="email"> Your email </label>
  <input type="email" name="email" ng-model="email" />
</form>
```

Angularがをするには、 **ng-model**をしてスコープでバインドするをするは、のとまったくじをします。のではメールがされています。をするには、はのようになります。

```
<input type="number" name="postalcode" ng-model="zipcode" />
```

なフォームののステップは、デフォルトのフォームのをにするのではなく、 **ng-submit**をしてコントローラのフォームにすることです。これはではありませんが、がスコープですでにであり、サブミットでできるため、はされます。フォームにをけることは、はいです。これらのにより、のがされます。

```
<form name="signup_form" ng-submit="submitFunc()" novalidate>
  <label name="email"> Your email </label>
  <input type="email" name="email" ng-model="email" />
  <button type="submit">Signup</button>
</form>
```

このコードはですが、Angularがするのもあります。

のステップでは、Angularが、 **ng- pristine** 、 **ng-dirty** 、 **ng-valid**および**ng-invalid**をしてフォームにクラスをすることをすることです 。あなたのCSSでこれらのクラスをすると、 *I*および*I*れたフィールドをスタイルすることができ、ユーザーがフォームにデータをするときにプレゼンテーションをできます。

フォームと

のあるフォームとには、コンテンツのにつさまざまながあります

`$touched`	フィールドにれた
`$untouched`	フィールドにはれられていません
`$pristine`	フィールドはされていません
`$dirty`	フィールドがされました
`$valid`	フィールドのはです
`$invalid`	フィールドコンテンツがです

のはすべてブールのプロパティであり、trueまたはfalseのいずれかになります。

これにより、ユーザーにメッセージをすることはにです。

```
<form name="myForm" novalidate>
    <input name="myName" ng-model="myName" required>
    <span ng-show="myForm.myName.$touched && myForm.myName.$invalid">This name is
invalid</span>
</form>
```

ここでは、 `ng-show`ディレクティブをして、フォームをしたがなにユーザーにメッセージをします。

CSSクラス

Angularは、フォームとのにじていくつかのCSSクラスをします

クラス	
`ng-touched`	フィールドにれた
`ng-untouched`	フィールドにはれられていません
`ng-pristine`	フィールドはされていません
`ng-dirty`	フィールドがされました
`ng-valid`	フィールドはです
`ng-invalid`	フィールドがです

これらのクラスをして、フォームにスタイルをすることができます

```
input.ng-invalid {
  background-color: crimson;
}
input.ng-valid {
  background-color: green;
}
```

ngMessages

`ngMessages`は、ビュ―にメッセ―ジをするためのスタイルをするためにされます。

のアプロ―チ

`ngMessages`に、たちはAngularのされたディレクティブ`ng-class`をってバリデ―ションメッセ―ジをします。このアプロ―チはゴミと`repetitive`でした。

さて、 `ngMessages`をすることでのカスタムメッセ―ジをできます。

Html

```
<form name="ngMessagesDemo">
  <input name="firstname" type="text" ng-model="firstname" required>
  <div ng-messages="ngMessagesDemo.firstname.$error">
    <div ng-message="required">Firstname is required.</div>
  </div>
</form>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.3.16/angular.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.3.16/angular-
messages.min.js"></script>
```

JS

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

app.controller('mainCtrl', function ($scope) {
    $scope.firstname = "Rohit";
});
```

カスタムフォ―ム

によっては、なではです。サポ―トのカスタムは、バリデ―タを`ngModelController` `$validators`オブジェクトに`ngModelController`ます

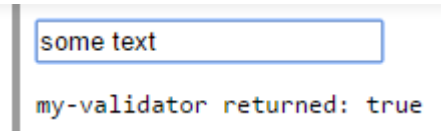
```
angular.module('app', [])
  .directive('myValidator', function() {
    return {
      // element must have ng-model attribute
      // or $validators does not work
```

```
      require: 'ngModel',
      link: function(scope, elm, attrs, ctrl) {
        ctrl.$validators.myValidator = function(modelValue, viewValue) {
            // validate viewValue with your custom logic
            var valid = (viewValue && viewValue.length > 0) || false;
            return valid;
        };
      }
    };
```

バリデーターは、`ngModel`をとするディレクティブとしてされています。バリデーターをするには、カスタム·ディレクティブをフォーム·コントローにするだけです。

```
<form name="form">
  <input type="text"
         ng-model="model"
         name="model"
         my-validator>
  <pre ng-bind="'my-validator returned: ' + form.model.$valid"></pre>
</form>
```

また`my-validator`はネイティブフォームコントローにするはありません。そのの`ng-model`であれば、どのでもかまいません。これは、カスタムビルドUIコンポーネントがあるにです。



ネストされたフォーム

コントローとをページににグループするでフォームをネストすることがましいことがあります。ただし、HTML5フォームはネストしないでください。わりにAngular Supplyを`ng-form`にします。

```
<form name="myForm" noValidate>
  <!-- nested form can be referenced via 'myForm.myNestedForm' -->
  <ng-form name="myNestedForm" noValidate>
    <input name="myInput1" ng-minlength="1" ng-model="input1" required />
    <input name="myInput2" ng-minlength="1" ng-model="input2" required />
  </ng-form>

  <!-- show errors for the nested subform here -->
  <div ng-messages="myForm.myNestedForm.$error">
    <!-- note that this will show if either input does not meet the minimum -->
    <div ng-message="minlength">Length is not at least 1</div>
  </div>
</form>

<!-- status of the form -->
<p>Has any field on my form been edited? {{myForm.$dirty}}</p>
<p>Is my nested form valid? {{myForm.myNestedForm.$valid}}</p>
<p>Is myInput1 valid? {{myForm.myNestedForm.myInput1.$valid}}</p>
```

フォームのは、フォームのにします。したがって、`myInput1` 1つがされて`$dirty`、それをむフォームも`$dirty`ます。これはフォームをむフォームにカスケードするので、 `myNestedForm`と`myForm`が`$dirty`ます。

バリデーター

バリデーターをすると、バックエンドにしてフォームをできます$ httpを。

こののバリデーターは、ユーザーやそののデータベースなど、さまざまなでクライアントにはないサーバーにアクセスするがあるにです。

バリデータをするには、 `input ng-model`にアクセスし、 `$asyncValidators`プロパティのコールバックをします。

のでは、されたがすでにするかどうかをします。がすでにする、またはされていない、バックエンドはをするステータスをします。がしない、されたがされます。

```
ngModel.$asyncValidators.usernameValidate = function (name) {
     if (name) {
          return AuthenticationService.checkIfNameExists(name); // returns a promise
     } else {
          return $q.reject("This username is already taken!"); // rejected promise
     }
};
```

の`ng-model`がされるたびに、このがされ、とともにがされます。

オンラインでフォームをむ https://riptutorial.com/ja/angularjs/topic/3979/フォーム

35: プロバイダー

- 、。
- 、。
- 、$ getFn;
- サービス、コンストラクタ。
- プロバイダ、プロバイダ。

プロバイダは、えば、のサービス、コントローラ、およびにできるシングルトンオブジェクトです。すべてのプロバイダは、なる「レシピ」をしてされてい`Provider`。 `Provider`は、ものいものです。すべてのなレシピは

-
-
-
- サービス
- プロバイダ

サービス、ファクトリ、プロバイダはすべてされ、コンポーネントはアプリケーションがしているにのみされます。

デコレータはプロバイダとにしています。デコレータは、サービスのやファクトリのをしたり、そのをしたりするためにされます。

Examples

`Constant`は、コンフィギュレーションフェーズとフェーズのでできます。

```
angular.module('app',[])
  .constant('endpoint', 'http://some.rest.endpoint') // define
  .config(function(endpoint) {
    // do something with endpoint
    // available in both config- and run phases
  })
  .controller('MainCtrl', function(endpoint) {       // inject
    var vm = this;
    vm.endpoint = endpoint;                          // usage
  });
```

```
<body ng-controller="MainCtrl as vm">
  <div>endpoint = {{ ::vm.endpoint }}</div>
</body>
```

エンドポイント= http//some.rest.endpoint

`Value`は、フェーズとフェーズのでできます。

```
angular.module('app',[])
  .value('endpoint', 'http://some.rest.endpoint') // define
  .run(function(endpoint) {
    // do something with endpoint
    // only available in run phase
  })
  .controller('MainCtrl', function(endpoint) {    // inject
    var vm = this;
    vm.endpoint = endpoint;                       // usage
  });
```

```
<body ng-controller="MainCtrl as vm">
  <div>endpoint = {{ ::vm.endpoint }}</div>
</body>
```

エンドポイント= http//some.rest.endpoint

`Factory`はでです。

Factoryレシピは、0のをつをしてしいサーービスをしますこれらはのサーービスにします。こののりは、このレシピによってされたサーービスインスタンスです。

ファクトリは、プリミティブ、オブジェクトリテラル、、またはカスタムタイプのインスタンスであっても、あらゆるタイプのサーービスをできます。

```
angular.module('app',[])
  .factory('endpointFactory', function() {
    return {
      get: function() {
        return 'http://some.rest.endpoint';
      }
    };
  })
  .controller('MainCtrl', function(endpointFactory) {
    var vm = this;
    vm.endpoint = endpointFactory.get();
  });
```

```
<body ng-controller="MainCtrl as vm">
  <div>endpoint = {{::vm.endpoint }}</div>
</body>
```

エンドポイント= http//some.rest.endpoint

サーービス

`Service`はでです。

サーービスレシピは、レシピまたはレシピとじようにサーービスをしますが、しいオペレ

—タでコンストラクタをびすことによってサ—ビスレシピがされます。コンストラクタは、こののインスタンスになをす0つのをることができます。

```
angular.module('app',[])
  .service('endpointService', function() {
    this.get = function() {
      return 'http://some.rest.endpoint';
    };
  })
  .controller('MainCtrl', function(endpointService) {
    var vm = this;
    vm.endpoint = endpointService.get();
  });
```

```
<body ng-controller="MainCtrl as vm">
  <div>endpoint = {{::vm.endpoint }}</div>
</body>
```

エンドポイント= http//some.rest.endpoint

プロバイダ

`Provider`は、フェ—ズとフェ—ズのでできます。

プロバイダレシピは、 `$get`メソッドをするカスタムとしてにされています。

プロバイダレシピは、アプリケ—ションをするにわなければならないアプリケ—ションのにAPIをするにのみするがあります。これは、アプリケ—ションでがわずかになるがあるなサ—ビスにしてのみいです。

```
angular.module('app',[])
  .provider('endpointProvider', function() {
    var uri = 'n/a';

    this.set = function(value) {
      uri = value;
    };

    this.$get = function() {
      return {
        get: function() {
          return uri;
        }
      };
    };
  })
  .config(function(endpointProviderProvider) {
    endpointProviderProvider.set('http://some.rest.endpoint');
  })
  .controller('MainCtrl', function(endpointProvider) {
    var vm = this;
    vm.endpoint = endpointProvider.get();
  });
```

```
<body ng-controller="MainCtrl as vm">
  <div>endpoint = {{::vm.endpoint }}</div>
</body>
```

エンドポイント= http//some.rest.endpoint

`config`フェーズなしでは

= n / a

オンラインでプロバイダーをむ https://riptutorial.com/ja/angularjs/topic/5169/プロバイダー

36: プロファイリングとパフォーマンス

Examples

7つのなパフォーマンスの

1ng-repeatをにする

ビューで`ng-repeat`をすると、にネストされた`ng-repeat`がある、パフォーマンスがします。

これはスローです

```
<div ng-repeat="user in userCollection">
  <div ng-repeat="details in user">
    {{details}}
  </div>
</div>
```

ネストされたリピートをなりけるようにしてください。 `ng-repeat`パフォーマンスをさせる1つのは`track by $index`またはのidフィールドで`track by $index`をすることです。デフォルトでは、 `ng-repeat`はオブジェクト`ng-repeat`します。 `track by` 、Angularはオブジェクトを`$index`またはobject idだける。

```
<div ng-repeat="user in userCollection track by $index">
  {{user.data}}
</div>
```

ページネーション 、 バーチャルスクロール 、 スクロール 、 limitToなどののアプローチをしてください 。であれば、なコレクションのをけるようにしてください。

2バインドする

Angularはデータバインディングをっています。それはあまりにもくするとくなるコストがしています。

いパフォーマンス

```
<!-- Default data binding has a performance cost -->
<div>{{ my.data }}</div>
```

よりいパフォーマンス AngularJS> = 1.3

```
<!-- Bind once is much faster -->
<div>{{ ::my.data }}</div>
```

```
<div ng-bind="::my.data"></div>

<!-- Use single binding notation in ng-repeat where only list display is needed  -->
<div ng-repeat="user in ::userCollection">
  {{::user.data}}
</div>
```

「バインド」をすると、ののダイジェストサイクルにがするのをつように、Angularにします。AngularはそのをDOMでし、すべてのウォッチャーをしてなにし、もはやモデルにバインドされなくなります。

`{{}}`ははるかにいです。

この`ng-bind`はディレクティブであり、されたにウォッチャーをします。したがって、されたがにされたにのみ、 `ng-bind`がされます。

、は、でないでも、すべての`$digest`でれチェックとリフレッシュされます。

3スコープとフィルタにがかかる

AngularJSにはダイジェストループがあります。ダイジェスト・サイクルがされるたびに、すべてのがビューにあり、フィルターがされます。ダイジェストループは、モデルがされたときにされ、アプリケーションがくなるがありますページがみまれるにフィルタがヒットするがあります。

これをする

```
<div ng-controller="bigCalulations as calc">
  <p>{{calc.calculateMe()}}</p>
  <p>{{calc.data | heavyFilter}}</p>
</div>
```

よりいアプローチ

```
<div ng-controller="bigCalulations as calc">
  <p>{{calc.preCalculatedValue}}</p>
  <p>{{calc.data | lightFilter}}</p>
</div>
```

コントローラーができる

```
app.controller('bigCalulations', function(valueService) {
    // bad, because this is called in every digest loop
    this.calculateMe = function() {
        var t = 0;
        for(i = 0; i < 1000; i++) {
            t += i;
        }
        return t;
    }
```

```
    // good, because this is executed just once and logic is separated in service to    keep
the controller light
    this.preCalulatedValue = valueService.valueCalculation(); // returns 499500
});
```

4のウォッチャー

ウォッチャーはパフォーマンスをにとします。ウォッチャーがえるほどダイジェストループがくなり、UIがくなります。ウォッチャーがをすると、ダイジェストループがされ、ビューがされます。

Angularのなをでするは3つあります。

`$watch()` - のをする

`$watchCollection()` - コレクションのをしますの`$watch`を`$watch`

`$watch(..., true)` - これをできるだけけ 、 "い"をし、パフォーマンスを`watchCollection`ます`watchCollection`をします

しいをするビューでをバインドするは、 `{{::variable}}`をして、にループでをしないようにしてください。

その、しているウォッチャーのをするがあります。このスクリプトでウォッチャーをカウントすることができますクレジットに@Wordsジャレド ウォッチャーの

```
(function() {
    var root = angular.element(document.getElementsByTagName('body')),
        watchers = [],
        f = function(element) {
        angular.forEach(['$scope', '$isolateScope'], function(scopeProperty) {
            if(element.data() && element.data().hasOwnProperty(scopeProperty)) {
                angular.forEach(element.data()[scopeProperty].$$watchers, function(watcher) {
                watchers.push(watcher);
                });
            }
        });

        angular.forEach(element.children(), function(childElement) {
            f(angular.element(childElement));
        });
    };

    f(root);

    // Remove duplicate watchers
    var watchersWithoutDuplicates = [];
    angular.forEach(watchers, function(item) {
        if(watchersWithoutDuplicates.indexOf(item) < 0) {
            watchersWithoutDuplicates.push(item);
        }
    });
```

```
    console.log(watchersWithoutDuplicates.length);
})();
```

5ng-if / ng-show

これらのはがにています。 `ng-if`はDOMからをしますが、 `ng-show`はをすだけですが、すべてのハンドラをします。したくないコードのがあるは、 `ng-if`してください。

それはのタイプにしますが、しばしばがよりもしています。

- がでないは、 `ng-if`
- オン/オフをすばやくりえるには、 `ng-show/ng-hide`します

```
<div ng-repeat="user in userCollection">
  <p ng-if="user.hasTreeLegs">I am special<!-- some complicated DOM --></p>
  <p ng-show="user.hasSubscribed">I am awesome<!-- switch this setting on and off --></p>
</div>
```

があれば、 `ng-if`をってテストしてください

6デバッグをにする

デフォルトでは、バインド·ディレクティブとスコープは、コードになクラスとマークアップをし、さまざまなデバッグ·ツールをします。このオプションをにすると、ダイジェストサイクルにこれらのさまざまなをレンダリングしなくなります。

```
angular.module('exampleApp', []).config(['$compileProvider', function ($compileProvider) {
    $compileProvider.debugInfoEnabled(false);
}]);
```

7をしてリソースをする

Dependency Injectionは、オブジェクトがオブジェクトをするのではなく、そのがえられているソフトウェアパターンです。ハードコーディングされたをし、にしてすることができます。

すべてのなのこのようなにするパフォーマンスのコストについてはにうかもしれません。 Angularは、めて$ injectプロパティをキャッシュすることでこれをします。したがって、がびされるがあるたびにこれはしません。

PROヒントのパフォーマンスでアプローチをしているは、$ injectプロパティアノテーションアプローチをしてください。このアプローチは、このロジックがアノテートののチェックにラップされるため、のをにします。if$ inject = fn。$ inject。 $ injectがすでにであれば、はです

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

var DemoController = function (s, h) {
    h.get('https://api.github.com/users/angular/repos').success(function (repos) {
        s.repos = repos;
    });
}
// $inject property annotation
DemoController['$inject'] = ['$scope', '$http'];

app.controller('DemoController', DemoController);
```

PROヒント2 `ng-app`とじに`ng-strict-di`ディレクティブをしてなDIモードをすることができます。これは、サービスがのをしようとするたびにエラーをします。

```
<html ng-app="DemoApp" ng-strict-di>
```

または、ブートストラップをする

```
angular.bootstrap(document, ['DemoApp'], {
    strictDi: true
});
```

Angularは、らしいデータバインディングをつことでいをています。デフォルトでは、モデルまたはビューコンポーネントのデータがされるたびに、モデルコンポーネントとビューコンポーネントのでバインドされたがにされます。

これはあまりにもくするとしくなるというコストがいます。これにより、パフォーマンスがにします。

パフォーマンスがい `{{my.data}}`

1りのバインディングをするには、のに2つのコロン`::`をします。この、はmy.dataがされるとされます。データのをしないようにしています。 Angularはのチェックをしないため、ダイジェストサイクルでされるがなくなります。

1りのバインディングをしたれたパフォーマンスの

```
{{::my.data}}
<span ng-bind="::my.data"></span>
<span ng-if="::my.data"></span>
<span ng-repeat="item in ::my.data">{{item}}</span>
<span ng-class="::{ 'my-class': my.data }"></div>
```

これにより、 `my.data`のデータバインディングがされるため、アプリケーションでこのフィールドがされたでも、そのフィールドはにビューにされません。したがって、アプリケーションのライフサイクルにわたってしないにしてのみしてください 。

スコープとフィルタ

AngularJSにはダイジェスト･ルーープがあり、ダイジェスト･サイクルがされるたびにビューのすべてのがされます。ダイジェストルーープは、モデルがされたときにされ、アプリケーションがくなるがありますページがロードされるにフィルタがヒットするがあります。

あなたはこれをけるべきです

```
<div ng-controller="bigCalulations as calc">
  <p>{{calc.calculateMe()}}</p>
  <p>{{calc.data | heavyFilter}}</p>
</div>
```

よりいアプローチ

```
<div ng-controller="bigCalulations as calc">
  <p>{{calc.preCalculatedValue}}</p>
  <p>{{calc.data | lightFilter}}</p>
</div>
```

コントローラーのサンプルはのとおりです。

```
.controller("bigCalulations", function(valueService) {
  // bad, because this is called in every digest loop
  this.calculateMe = function() {
    var t = 0;
    for(i = 0; i < 1000; i++) {
      t = t + i;
    }
    return t;
  }
  //good, because it is executed just once and logic is separated in service to    keep the
controller light
  this.preCalulatedValue = valueService.caluclateSumm(); // returns 499500
});
```

ウォッチャー

ウォッチャはらかのをし、このがされたことをするがありました。

`$watch()`または`$watchCollection`びした、しいウォッチャーはのスコープのウォッチャーコレクションにします。

では、ウォッチャーとはですか

Watcherはなで、ダイジェストサイクルごとにびされ、をします。 Angularは、のびしとじでないはりをチェックします。 `$watch()`または`$watchCollection`に2のパラメータでされたコールバックがされます。

```
(function() {
  angular.module("app", []).controller("ctrl", function($scope) {
    $scope.value = 10;
    $scope.$watch(
```

```
      function() { return $scope.value; },
      function() { console.log("value changed"); }
    );
  }
})();
```

ウォッチャーはパフォーマンスのキラーです。あなたがつウォッチャーがいほど、ダイジェストループをるのにがかかり、UIがくなります。ウォッチャーがをすると、ダイジェストループがされますすべてのでされます

のをでするは3つあります。

`$watch()` - のだけをする

`$watchCollection()` - コレクションのをしますの$ watchをします

`$watch(..., true)` - なりけ 、 "い"をし、パフォーマンスをにしますwatchCollectionをします

ビューでをバインドするは、しいウォッチャーをしていることにしてください。に`{{::variable}}`をしてウォッチャーをしないでください

その、しているウォッチャーのをするがあります。あなたはこのスクリプトでウォッチャーをえることができます Jaredのように@Wordsのように - ページのウォッチのをえる

```
(function() {
  var root = angular.element(document.getElementsByTagName("body")),
      watchers = [];

  var f = function(element) {

    angular.forEach(["$scope", "$isolateScope"], function(scopeProperty) {
      if(element.data() && element.data().hasOwnProperty(scopeProperty)) {
        angular.forEach(element.data()[scopeProperty].$$watchers, function(watcher) {
          watchers.push(watcher);
        });
      }
    });

    angular.forEach(element.children(), function(childElement) {
      f(angular.element(childElement));
    });

  };

  f(root);

  // Remove duplicate watchers
  var watchersWithoutDuplicates = [];
  angular.forEach(watchers, function(item) {
    if(watchersWithoutDuplicates.indexOf(item) < 0) {
      watchersWithoutDuplicates.push(item);
    }
  });

  console.log(watchersWithoutDuplicates.length);
```

```
})();
```

のスクリプトをしたくないは、 ng-statsというオープンソースユーティリティがあります。このユーティリティはページにめまれたリアルタイムチャートをして、Angularがしているのととともにダイジェストサイクルのおよび。このユーティリティは`showAngularStats`というグローバルをしています。このをびすと、チャートのをできます。

```
showAngularStats({
  "position": "topleft",
  "digestTimeThreshold": 16,
  "autoload": true,
  "logDigest": true,
  "logWatches": true
});
```

のサンプルコードは、にページにのグラフをします インタラクティブなデモ 。

ng-ifng-show

これらのはがにています。いは、 `ng-if`がDOMからをすることです。されないコードのがある`ng-if`は、 `ng-if`がくです。 `ng-show`はをすだけですが、すべてのハンドラをします。

ng-if

ngIfディレクティブは、についてDOMツリーのをまたはします。 ngIfにりてられたがのにされた、はDOMからされます。そうでない、のクローンがDOMにされます。

ng-show

ngShowディレクティブは、ngShowにされたについて、されたHTMLをまたはにします。をまたはにするには、ng-hide CSSクラスをするか、にします。

```
<div ng-repeat="user in userCollection">
  <p ng-if="user.hasTreeLegs">I am special
    <!-- some complicated DOM -->
  </p>
  <p ng-show="user.hasSubscribed">I am aweosme
    <!-- switch this setting on and off -->
  </p>
</div>
```

それはのタイプにしますが、しばしばがよりもですえば、エレメントがでないの95が`ng-if`する`ng-if`、DOMエレメントのをりえるがあるは、`ng-show`。

があるは、`ng-if`をしてテストしてください

`ng-if`はしいスコープをしますが、`ng-show`および`ng-hide`はしません。スコーププロパティにアクセスできないは、`$parent.property`します。

あなたのモデルをする

```
<div ng-controller="ExampleController">
    <form name="userForm">
        Name:
        <input type="text" name="userName"
           ng-model="user.name"
           ng-model-options="{ debounce: 1000 }" />
            <button ng-click="userForm.userName.$rollbackViewValue();
user.name=''">Clear</button><br />
    </form>
    <pre>user.name = </pre>
</div>
```

のでは、1の1000ミリのデバウンスをしています。これはかなりれていますが、くの`$digest`サイクルで`ng-model`がりしされるのをぐことができます。

フィールドやそののでがなでデバウンスをすると、Angularアプリケーションのパフォーマンスをにさせることができます。をずらすことができるだけでなく、アクションがトリガーされたときにすることもできます。すべてのキーストロークでモデルをしたくないは、ぼかしもすることができます。

のスコープのスコープにされているリスナーは、にする

にすように、のスコープのスコープをするがあります。

```
//always deregister these
$rootScope.$on(...);
$scope.$parent.$on(...);
```

あなたはがそれをするので、のスコープでリスナーをするはありません

```
//no need to deregister this
$scope.$on(...);
```

リスナーの`$rootScope.$on`、のコントローラーにするとメモリにります。これにより、コントローラがになったにメモリリークがします。

しないでください

```
angular.module('app').controller('badExampleController', badExample);
```

```
badExample.$inject = ['$scope', '$rootScope'];

function badExample($scope, $rootScope) {
    $rootScope.$on('post:created', function postCreated(event, data) {});
}
```

う

```
angular.module('app').controller('goodExampleController', goodExample);
goodExample.$inject = ['$scope', '$rootScope'];

function goodExample($scope, $rootScope) {
    var deregister = $rootScope.$on('post:created', function postCreated(event, data) {});

    $scope.$on('$destroy', function destroyScope() {
        deregister();
    });
}
```

オンラインでプロファイリングとパフォーマンスをむ
https://riptutorial.com/ja/angularjs/topic/1921/プロファイリングとパフォーマンス

37: モジュール

Examples

モジュール

モジュールは、コントローラ、サービス、フィルタ、ディレクティブなど、アプリケーションのさまざまなのコンテナとしてします。モジュールは、Angularのをしてのモジュールによってできます。

モジュールの

```
angular
    .module('app', []);
```

は[]のでされたモジュールのリストされたappするがしない、々はの、すなわちをします[]

のモジュールのとしてモジュールをする

```
angular.module('app', [
    'app.auth',
    'app.dashboard'
]);
```

モジュールの

```
angular
    .module('app');
```

モジュール

モジュールは、コントローラ、サービス、フィルタ、ディレクティブなど、アプリケーションのさまざまなのコンテナです。

モジュールをする
ほとんどのアプリケーションには、アプリケーションのさまざまなをインスタンスしてするながあります。
アプリにはなはありません。
しかし、AngularJsでは、なプロセスはしやすく、コードをなモジュールとしてパッケージすることもできます。
モジュールはをらせるため、モジュールはのでロードできます。

モジュールをする

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

```
// Empty array is list of modules myApp is depends on.
// if there are any required dependancies,
// then you can add in module, Like ['ngAnimate']

app.controller('myController', function() {

  // write your business logic here
});
```

モジュールのみみと

1. ブロック - プロバイダおよびフェーズにされます。

```
angular.module('myModule', []).
config(function(injectables) {
  // here you can only inject providers in to config blocks.
});
```

2. Run Blocks - インジェクタがされたにされ、アプリケーションのにされます。

```
angular.module('myModule', []).
run(function(injectables) {
  // here you can only inject instances in to config blocks.
});
```

オンラインでモジュールをむ https://riptutorial.com/ja/angularjs/topic/844/モジュール

38: ユー・ルーター

ui-routerとはですか

> Angular UI-Routerは、AngularJSのクライアントのシングルページアプリケーションルーティングフレームワークです。
>
> SPAのルーティングフレームワークは、ユーザーがアプリをナビゲートするにブラウザのURLをします。これとはに、ブラウザのURLをしてアプリをナビゲートすることができるため、ユーザーはSPAのいにブックマークをすることができます。
>
> UI-Routerアプリケーションは、のツリーとしてモデルされています。 UI-Routerは、それらのアプリケーションのをトランザクションのようにするステートマシンをします。

につリンク

ここでのAPIドキュメントをつけることができます。 `ui-router` VS `ng-router`にするについては、ここでになえをつけることができます。 `ng-router`は、ui-routerとじようなをサポートする `ngNewRouter` Angular 1.5 + / 2.0とがありますとばれるしいng-routerアップデートをリリースしました。 ここで `ngNewRouter`をむことができます 。

Examples

な

app.js

```
angular.module('myApp', ['ui.router'])
  .controller('controllerOne', function() {
    this.message = 'Hello world from Controller One!';
  })
  .controller('controllerTwo', function() {
    this.message = 'Hello world from Controller Two!';
  })
  .controller('controllerThree', function() {
    this.message = 'Hello world from Controller Three!';
  })
  .config(function($stateProvider, $urlRouterProvider) {
    $stateProvider
      .state('one', {
        url: "/one",
        templateUrl: "view-one.html",
        controller: 'controllerOne',
        controllerAs: 'ctrlOne'
      })
      .state('two', {
        url: "/two",
        templateUrl: "view-two.html",
        controller: 'controllerTwo',
```

```
        controllerAs: 'ctrlTwo'
      })
      .state('three', {
        url: "/three",
        templateUrl: "view-three.html",
        controller: 'controllerThree',
        controllerAs: 'ctrlThree'
      });

      $urlRouterProvider.otherwise('/one');
  });
```

index.html

```
<div ng-app="myApp">
  <nav>
    <!-- links to switch routes -->
    <a ui-sref="one">View One</a>
    <a ui-sref="two">View Two</a>
    <a ui-sref="three">View Three</a>
  </nav>
  <!-- views will be injected here -->
  <div ui-view></div>
  <!-- templates can live in normal html files -->
  <script type="text/ng-template" id="view-one.html">
    <h1>{{ctrlOne.message}}</h1>
  </script>

  <script type="text/ng-template" id="view-two.html">
    <h1>{{ctrlTwo.message}}</h1>
  </script>

  <script type="text/ng-template" id="view-three.html">
    <h1>{{ctrlThree.message}}</h1>
  </script>
</div>
```

のビュー

app.js

```
angular.module('myApp', ['ui.router'])
  .controller('controllerOne', function() {
    this.message = 'Hello world from Controller One!';
  })
  .controller('controllerTwo', function() {
    this.message = 'Hello world from Controller Two!';
  })
  .controller('controllerThree', function() {
    this.message = 'Hello world from Controller Three!';
  })
  .controller('controllerFour', function() {
    this.message = 'Hello world from Controller Four!';
  })
  .config(function($stateProvider, $urlRouterProvider) {
    $stateProvider
      .state('one', {
        url: "/one",
```

```
        views: {
          "viewA": {
            templateUrl: "view-one.html",
            controller: 'controllerOne',
            controllerAs: 'ctrlOne'
          },
          "viewB": {
            templateUrl: "view-two.html",
            controller: 'controllerTwo',
            controllerAs: 'ctrlTwo'
          }
        }
      })
      .state('two', {
        url: "/two",
        views: {
          "viewA": {
            templateUrl: "view-three.html",
            controller: 'controllerThree',
            controllerAs: 'ctrlThree'
          },
          "viewB": {
            templateUrl: "view-four.html",
            controller: 'controllerFour',
            controllerAs: 'ctrlFour'
          }
        }
      });

    $urlRouterProvider.otherwise('/one');
  });
```

index.html

```
<div ng-app="myApp">
  <nav>
    <!-- links to switch routes -->
    <a ui-sref="one">Route One</a>
    <a ui-sref="two">Route Two</a>
  </nav>
  <!-- views will be injected here -->
  <div ui-view="viewA"></div>
  <div ui-view="viewB"></div>
  <!-- templates can live in normal html files -->
  <script type="text/ng-template" id="view-one.html">
    <h1>{{ctrlOne.message}}</h1>
  </script>

  <script type="text/ng-template" id="view-two.html">
    <h1>{{ctrlTwo.message}}</h1>
  </script>

  <script type="text/ng-template" id="view-three.html">
    <h1>{{ctrlThree.message}}</h1>
  </script>

  <script type="text/ng-template" id="view-four.html">
    <h1>{{ctrlFour.message}}</h1>
  </script>
</div>
```

データをロードするためのの

app.js

```
angular.module('myApp', ['ui.router'])
  .service('User', ['$http', function User ($http) {
    this.getProfile = function (id) {
      return $http.get(...) // method to load data from API
    };
  }])
  .controller('profileCtrl', ['profile', function profileCtrl (profile) {
    // inject resolved data under the name of the resolve function
    // data will already be returned and processed
    this.profile = profile;
  }])
  .config(['$stateProvider', '$urlRouterProvider', function ($stateProvider,
$urlRouterProvider) {
    $stateProvider
      .state('profile', {
        url: "/profile/:userId",
        templateUrl: "profile.html",
        controller: 'profileCtrl',
        controllerAs: 'vm',
        resolve: {
          profile: ['$stateParams', 'User', function ($stateParams, User) {
            // $stateParams will contain any parameter defined in your url
            return User.getProfile($stateParams.userId)
              // .then is only necessary if you need to process returned data
              .then(function (data) {
                return doSomeProcessing(data);
              });
          }]
        }
      }]);

      $urlRouterProvider.otherwise('/');
  });
```

profile.html

```
<ul>
  <li>Name: {{vm.profile.name}}</li>
  <li>Age: {{vm.profile.age}}</li>
  <li>Sex: {{vm.profile.sex}}</li>
</ul>
```

UIルーターのWikiエントリーをここでしてください 。

Resolveは、 `$stateChangeSuccess`イベントがするにされなければなりません。つまり、のすべてのがするまで、UIはみまれません。これは、データがコントローラとUIでできるようにするためのらしいです。ただし、UIをハングしないようにするには、をにするがあります。

ネストされたビュー/

app.js

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

app.config(function($stateProvider,$urlRouterProvider) {

    $stateProvider

    .state('home', {
        url: '/home',
        templateUrl: 'home.html',
        controller: function($scope){
            $scope.text = 'This is the Home'
        }
    })

    .state('home.nested1',{
        url: '/nested1',
        templateUrl:'nested1.html',
        controller: function($scope){
            $scope.text1 = 'This is the nested view 1'
        }
    })

    .state('home.nested2',{
        url: '/nested2',
        templateUrl:'nested2.html',
        controller: function($scope){
            $scope.fruits = ['apple','mango','oranges'];
        }
    });

    $urlRouterProvider.otherwise('/home');

});
```

index.html

```
  <div ui-view></div>
   <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.5.8/angular.min.js"></script>
   <script src="angular-ui-router.min.js"></script>
   <script src="app.js"></script>
```

home.html

```
<div>
<h1> {{text}} </h1>
<br>
    <a ui-sref="home.nested1">Show nested1</a>
    <br>
    <a ui-sref="home.nested2">Show nested2</a>
    <br>

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

nested1.html

```
<div>
<h1> {{text1}} </h1>
</div>
```

nested2.html

```
<div>
    <ul>
      <li ng-repeat="fruit in fruits">{{ fruit }}</li>
    </ul>
</div>
```

オンラインでユー·ルーターをむ https://riptutorial.com/ja/angularjs/topic/2545/ユー-ルーター

39: レイジーローディング

1. ロードされたにのみみがなは、しいでロードするようにしてください

```
angular.module('lazy', [
 'alreadyLoadedDependency1',
 'alreadyLoadedDependency2',
 ...
 {
  files: [
  'path/to/lazily/loaded/dependency1.js',
  'path/to/lazily/loaded/dependency2.js', //<--- requires lazily loaded dependency1
  'path/to/lazily/loaded/dependency.css'
  ],
  serie: true //Sequential load instead of parallel
 }
]);
```

Examples

ロードのためのプロジェクトの

むoclazyload.jsあなたのインデックスファイルには、ocLazyLoadでとしてapp.js

```
//Make sure you put the correct dependency! it is spelled different than the service!
angular.module('app', [
 'oc.lazyLoad',
 'ui-router'
])
```

ファイルをロードするには、 $ocLazyLoadサービスをコントローラまたはのサービスにします

```
.controller('someCtrl', function($ocLazyLoad) {
 $ocLazyLoad.load('path/to/file.js').then(...);
});
```

モジュールはににロードされます。

そののバリエーション

```
$ocLazyLoad.load([
  'bower_components/bootstrap/dist/js/bootstrap.js',
  'bower_components/bootstrap/dist/css/bootstrap.css',
  'partials/template1.html'
]);
```

バリエーションのは、 ドキュメントをごください

ルータでの

UI-Router

```
.state('profile', {
 url: '/profile',
 controller: 'profileCtrl as vm'
 resolve: {
  module: function($ocLazyLoad) {
   return $ocLazyLoad.load([
    'path/to/profile/module.js',
    'path/to/profile/style.css'
   ]);
  }
 }
});
```

ngRoute

```
.when('/profile', {
     controller: 'profileCtrl as vm'
     resolve: {
      module: function($ocLazyLoad) {
       return $ocLazyLoad.load([
        'path/to/profile/module.js',
        'path/to/profile/style.css'
       ]);
      }
     }
    });
```

をする

のでは、サ―ビスのになではなく、 `module.js`をできます

```
//lazy_module.js
angular.module('lazy', [
 'alreadyLoadedDependency1',
 'alreadyLoadedDependency2',
 ...
 [
  'path/to/lazily/loaded/dependency.js',
  'path/to/lazily/loaded/dependency.css'
 ]
]);
```

このはロ―ドされたモジュ―ルでのみします。

ディレクティブの

```
<div oc-lazy-load="['path/to/lazy/loaded/directive.js',
'path/to/lazy/loaded/directive.html']">

<!-- myDirective available here -->
```

```
<my-directive></my-directive>

</div>
```

オンラインでレイジーローディングをむ https://riptutorial.com/ja/angularjs/topic/6400/レイジーローディング

40:

- myApp.controller 'MyController'、function$ scope{...}; //コード
- myApp.controller 'MyController'、['$ scope'、function$ scope{...}]; //ミニネットをサポートする
- function MyController{}

  MyController $ inject = ['$ scope'];

  myApp.controller 'MyController'、MyController; // $をする
- $ injector.get 'injectable'; ///の

プロバイダは`run`ブロックにすることはできません。

サービスまたはを`config`ブロックにすることはできません。

あなたのがをけるようにして、コードがされないようにしてください。

Examples

Angularアプリケーションでののもな - `$scope`をAngular `Controller`する

```
angular.module('myModule', [])
.controller('myController', ['$scope', function($scope) {
    $scope.members = ['Alice', 'Bob'];
    ...
}])
```

は`controller` `$scope`をするをしていますが、モジュールをのモジュールにするかどうかはじです。プロセスはじです。

Angularのシステムは、あなたのをすることをしています。たとえば、サービスをするは、ののようにサービスをリストすることができ、それはになります。

DIは、コンポーネントをするであればどこでもできます。

のでは、「インラインアノテーション」とばれるものをしています。つまり、のをとしてにします。コードがProductionにされたときにアプリケーションがしないようにします。コードのは、ではなくのをし、をさせます。をすることで、Angularはなをることができます。

にです - のは、のパラメータとじでなければなりません。

このプロセスをし、これをするツールがあります。

ダイナミックインジェクション

コンポーネントをにするオプションもあります。あなたは`$injector`サービスをってそれをうことができます

```
myModule.controller('myController', ['$injector', function($injector) {
    var myService = $injector.get('myService');
}]);
```

このメソッドは、アプリケーションをすのあるのをぐためにできますが、をするのはベストプラクティスとはみなされません。は、、アプリケーションのアーキテクチャにがあることをしており、わりにそれにするがあります。

$ injectプロパティ

に、 `$inject`プロパティをしてとじようにすることができます

```
var MyController = function($scope) {
  // ...
}
MyController.$inject = ['$scope'];
myModule.controller('MyController', MyController);
```

バニラJavaScriptでAngularJSサービスをにロードする

AngularJS `injector()`メソッドをして、AngularJSサービスをバニラJavaScriptでみむことができます。 `angular.element()`をびしてりされたすべてのjqLiteには、インジェクタをするためにできるメソッド`injector()`があります。

```
var service;
var serviceName = 'myService';

var ngAppElement = angular.element(document.querySelector('[ng-app],[data-ng-app]') ||
document);
var injector = ngAppElement.injector();

if(injector && injector.has(serviceNameToInject)) {
    service = injector.get(serviceNameToInject);
}
```

のでは、AngularJSアプリケーション `ngAppElement` のルートをむjqLiteをしようとしています。これをうには、 `ng-app`または`data-ng-app`をむDOMをする`angular.element()`メソッドをし`document`。しないは、 `document`にり`document`。 `ngAppElement`をしてインジェクタインスタンスをします`ngAppElement.injector()`。 injectorインスタンスは、するサービスがするかどうか `injector.has()`をし、 `serviceinjector.get()` を`service`にロードするために`service`ます。

オンラインでをむ https://riptutorial.com/ja/angularjs/topic/1582/

41: テスト

このトピックでは、AngularJSのさまざまなをテストするためのをします。テストは、なのテストフレームワークであるJasmineをしてされることがよくあります。ユニットテストのアンギュラコンストラクトをするときは、ユニットテストをするときにとしてngMockをインクルードするがあります。

Examples

ユニットテストフィルタ

フィルタコード

```
angular.module('myModule', []).filter('multiplier', function() {
  return function(number, multiplier) {
    if (!angular.isNumber(number)) {
      throw new Error(number + " is not a number!");
    }
    if (!multiplier) {
      multiplier = 2;
    }
    return number * multiplier;
  }
});
```

テスト

```
describe('multiplierFilter', function() {
  var filter;

  beforeEach(function() {
    module('myModule');
    inject(function(multiplierFilter) {
      filter = multiplierFilter;
    });
  });

  it('multiply by 2 by default', function() {
    expect(filter(2)).toBe(4);
    expect(filter(3)).toBe(6);
  });

  it('allow to specify custom multiplier', function() {
    expect(filter(2, 4)).toBe(8);
  });

  it('throws error on invalid input', function() {
    expect(function() {
      filter(null);
    }).toThrow();
  });
});
```

テストの`inject`びしでは、フィルタ+ フィルタでフィルタをするがあります。このは、モジュールにフィルタをするたびに、そのに`Filter`されてAngularにされることです。

ユニットテストコンポーネント1.5+

コンポーネントコード

```
angular.module('myModule', []).component('myComponent', {
  bindings: {
    myValue: '<'
  },
  controller: function(MyService) {
    this.service = MyService;
    this.componentMethod = function() {
      return 2;
    };
  }
});
```

テスト

```
describe('myComponent', function() {
  var component;

  var MyServiceFake = jasmine.createSpyObj(['serviceMethod']);

  beforeEach(function() {
    module('myModule');
    inject(function($componentController) {
      // 1st - component name, 2nd - controller injections, 3rd - bindings
      component = $componentController('myComponent', {
        MyService: MyServiceFake
      }, {
        myValue: 3
      });
    });
  });

  /** Here you test the injector. Useless. */

  it('injects the binding', function() {
    expect(component.myValue).toBe(3);
  });

  it('has some cool behavior', function() {
    expect(component.componentMethod()).toBe(2);
  });
});
```

コントローラのユニットテスト

コントローラコード

```
angular.module('myModule', [])
  .controller('myController', function($scope) {
```

```
    $scope.num = 2;
    $scope.doSomething = function() {
      $scope.num += 2;
    }
  });
```

テスト

```
describe('myController', function() {
  var $scope;
  beforeEach(function() {
    module('myModule');
    inject(function($controller, $rootScope) {
      $scope = $rootScope.$new();
      $controller('myController', {
        '$scope': $scope
      })
    });
  });
  it('should increment `num` by 2', function() {
    expect($scope.num).toEqual(2);
    $scope.doSomething();
    expect($scope.num).toEqual(4);
  });
});
```

ユニットテストサービス

サービスコード

```
angular.module('myModule', [])
  .service('myService', function() {
    this.doSomething = function(someNumber) {
      return someNumber + 2;
    }
  });
```

テスト

```
describe('myService', function() {
  var myService;
  beforeEach(function() {
    module('myModule');
    inject(function(_myService_) {
      myService = _myService_;
    });
  });
  it('should increment `num` by 2', function() {
    var result = myService.doSomething(4);
    expect(result).toEqual(6);
  });
});
```

ユニットテストディレクティブ

コード

```
angular.module('myModule', [])
  .directive('myDirective', function() {
    return {
      template: '<div>{{greeting}} {{name}}!</div>',
      scope: {
        name: '=',
        greeting: '@'
      }
    };
  });
```

テスト

```
describe('myDirective', function() {
  var element, scope;
  beforeEach(function() {
    module('myModule');
    inject(function($compile, $rootScope) {
      scope = $rootScope.$new();
      element = angular.element("<my-directive name='name' greeting='Hello'></my-directive>");
      $compile(element)(scope);
      /* PLEASE NEVER USE scope.$digest(). scope.$apply use a protection to avoid to run a
digest loop when there is already one, so, use scope.$apply() instead. */
      scope.$apply();
    })
  });

  it('has the text attribute injected', function() {
    expect(element.html()).toContain('Hello');
  });

  it('should have proper message after scope change', function() {
    scope.name = 'John';
    scope.$apply();
    expect(element.html()).toContain("John");
    scope.name = 'Alice';
    expect(element.html()).toContain("John");
    scope.$apply();
    expect(element.html()).toContain("Alice");
  });
});
```

オンラインでテストをむ https://riptutorial.com/ja/angularjs/topic/1689/テスト

42:

cssファイルでng-hideクラスをします。 ng-show / hideはクラスなしではしません。

Examples

サービス

サービス

```
angular.module('core').factory('print_service', ['$rootScope', '$compile', '$http',
'$timeout','$q',
    function($rootScope, $compile, $http, $timeout,$q) {

        var printHtml = function (html) {
            var deferred = $q.defer();
            var hiddenFrame = $('<iframe style="display:
none"></iframe>').appendTo('body')[0];

            hiddenFrame.contentWindow.printAndRemove = function() {
                hiddenFrame.contentWindow.print();
                $(hiddenFrame).remove();
                deferred.resolve();
            };

            var htmlContent =   "<!doctype html>"+
                                "<html>"+
                                    '<head><link rel="stylesheet" type="text/css"
href="/style/css/print.css"/></head>'+
                                    '<body onload="printAndRemove();">' +
                                        html +
                                    '</body>'+
                                "</html>";

            var doc = hiddenFrame.contentWindow.document.open("text/html", "replace");
            doc.write(htmlContent);
            doc.close();
            return deferred.promise;
        };

        var openNewWindow = function (html) {
            var newWindow = window.open("debugPrint.html");
            newWindow.addEventListener('load', function(){
                $(newWindow.document.body).html(html);
            }, false);
        };

        var print = function (templateUrl, data) {

            $rootScope.isBeingPrinted = true;

            $http.get(templateUrl).success(function(template){
                var printScope = $rootScope.$new()
                angular.extend(printScope, data);
                var element = $compile($('<div>' + template + '</div>'))(printScope);
```

```
                var waitForRenderAndPrint = function() {
                    if(printScope.$$phase || $http.pendingRequests.length) {
                        $timeout(waitForRenderAndPrint, 1000);
                    } else {
                        // Replace printHtml with openNewWindow for debugging
                        printHtml(element.html());
                        printScope.$destroy();
                    }
                };
                waitForRenderAndPrint();
            });
        };

        var printFromScope = function (templateUrl, scope, afterPrint) {
            $rootScope.isBeingPrinted = true;
            $http.get(templateUrl).then(function(response){
                var template = response.data;
                var printScope = scope;
                var element = $compile($('<div>' + template + '</div>'))(printScope);
                var waitForRenderAndPrint = function() {
                    if (printScope.$$phase || $http.pendingRequests.length) {
                        $timeout(waitForRenderAndPrint);
                    } else {
                        // Replace printHtml with openNewWindow for debugging
                        printHtml(element.html()).then(function() {
                            $rootScope.isBeingPrinted = false;
                            if (afterPrint) {
                                afterPrint();
                            }
                        });
                    }
                };
                waitForRenderAndPrint();
            });
        };

        return {
            print : print,
            printFromScope : printFromScope
        }
    }
]);
```

コントローラ

```
var template_url = '/views/print.client.view.html';
print_service.printFromScope(template_url,$scope,function(){
      // Print Completed
});
```

オンラインでをむ https://riptutorial.com/ja/angularjs/topic/6750/

43:

あなたのをに でをくことは、くのでよくわれているベストプラクティスです。されたのをにすることもです。

`.controller('MyController', function($scope, Profile, EVENT))` 、あなたはすぐにそれをることができます

- `$scope`はです
- `Profile`はカスタムサービスまたはファクトリです
- `EVENT`は

Examples

のをする

```
angular
  .module('MyApp', [])
  .constant('VERSION', 1.0);
```

はされ、コントローラ、サービス、ファクトリ、プロバイダ、さらにはコンフィグレーションメソッドでもできます

```
angular
  .module('MyApp')
  .controller('FooterController', function(VERSION) {
    this.version = VERSION;
  });
```

```
<footer ng-controller="FooterController as Footer">{{ Footer.version }}</footer>
```

ユースケース

ここでははありませんが、は、アプリケーションやチームがしめたとき、またはにしいコードをくことをしているににです

- リファクタリングコード。イベントのの。アプリケーションでくのイベントをするは、イベントのはどこにでもあります。しいがあなたのチームにわったとき、はなるでのイベントにをつけます...あなたはにイベントのをでグループすることでこれをぐことができます

  ```
  angular
    .module('MyApp')
    .constant('EVENTS', {
      LOGIN_VALIDATE_FORM: 'login::click-validate',
      LOGIN_FORGOT_PASSWORD: 'login::click-forgot',
  ```

```
    LOGIN_ERROR: 'login::notify-error',
    ...
  });
```

```
angular
  .module('MyApp')
  .controller('LoginController', function($scope, EVENT) {
    $scope.$on(EVENT.LOGIN_VALIDATE_FORM, function() {
      ...
    });
  })
```

...そして、あなたのイベントのはオ―トコンプリ―トからをることができます

- をします。すべてのをじにします。

```
angular
  .module('MyApp')
  .constant('CONFIG', {
    BASE_URL: {
      APP: 'http://localhost:3000',
      API: 'http://localhost:3001'
    },
    STORAGE: 'S3',
    ...
  });
```

- をする。には、あなたがにりにっていないものがいくつかあります。たとえば、ハ―ドコ―ドされたです。あなたのメインコ―ドでそれらをさせるわりに、をすることができます

```
angular
  .module('MyApp')
  .constant('HARDCODED', {
    KEY: 'KEY',
    RELATION: 'has_many',
    VAT: 19.6
  });
```

...リファクタリングのようなもの

```
$scope.settings = {
  username: Profile.username,
  relation: 'has_many',
  vat: 19.6
}
```

に

```
$scope.settings = {
  username: Profile.username,
  relation: HARDCODED.RELATION,
```

```
  vat: HARDCODED.VAT
}
```

オンラインでをむ https://riptutorial.com/ja/angularjs/topic/3967/

44: な

Examples

アプリケーションをローカルでする

> のでは、 node.jsがインストールされ、 npmがであることがです。
> なコードは、GitHub @ https://github.com/mikkoviitala/angular-grunt-run-localからフォークすることができます。

、しいWebアプリケーションをするときににうことの1つは、ローカルですることです。

では、 grunt javascriptタスクランナー、 npm ノードパッケージマネージャー、 bower さらにのパッケージマネージャーをして、これをするなをします。

のアプリケーションファイルのには、のツールをしてサードパーティのをいくつかインストールするがあります。あなたのプロジェクトディレクトリ **root**がましい には、3つのファイルがです。

- package.jsonnpmでされる
- bower.jsonbowerによってされる
- gruntfile.jsハッキリしたタスク

プロジェクトディレクトリはのようになります

---

package.json

たちは**grunt**、 **matchdep**をインストールし、をでフィルタリングできるようにします。grunt **-express**は gruntと**grunt-open**でWebサーバをして、gruntタスクからurls /ファイルをきます。

したがって、これらのパッケージは、すべて「インフラストラクチャ」についてのものであり、々のアプリケーションをするヘルパーです。

```
{
  "name": "app",
  "version": "1.0.0",
  "dependencies": {},
  "devDependencies": {
    "grunt": "~0.4.1",
    "matchdep": "~0.1.2",
```

```
    "grunt-express": "~1.0.0-beta2",
    "grunt-open": "~0.2.1"
  },
  "scripts": {
    "postinstall": "bower install"
  }
}
```

bower.json

Bowerはフロントエンドにするものですなくとも、そうすべきです。それをしてをします。

```
{
  "name": "app",
  "version": "1.0.0",
  "dependencies": {
    "angular": "~1.3.x"
  },
  "devDependencies": {}
}
```

gruntfile.js

gruntfile.jsのには、 http// localhost9000 /にあるしいブラウザウィンドウでアプリケーションをく「ローカルでのアプリケーション」というがあります。

```
'use strict';

// see http://rhumaric.com/2013/07/renewing-the-grunt-livereload-magic/

module.exports = function(grunt) {
  require('matchdep').filterDev('grunt-*').forEach(grunt.loadNpmTasks);

  grunt.initConfig({
    express: {
      all: {
        options: {
          port: 9000,
          hostname: 'localhost',
          bases: [__dirname]
        }
      }
    },

    open: {
      all: {
        path: 'http://localhost:<%= express.all.options.port%>'
      }
    }
  });

  grunt.registerTask('app', [
    'express',
    'open',
    'express-keepalive'
  ]);
};
```

あなたのアプリケーションをからさせるには、のファイルをプロジェクトのルートディレクトリにしますのフォルダでもです。に、コンソール/コマンドラインをし、のようにしてなをすべてインストールします。

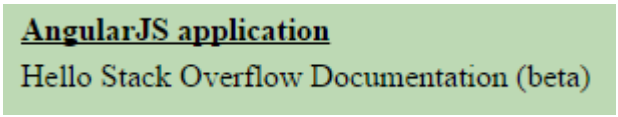
```
npm install -g grunt-cli bower
npm install
```

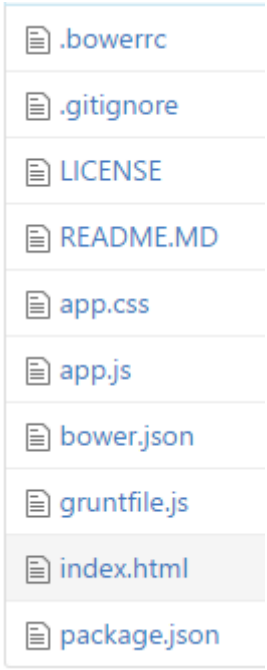
そして、

```
grunt app
```

はい、のアプリケーションファイルもになることにしてください。
ほぼのでは、こののでべたGitHubリポジトリをしてください。

はそれほどいはありません。 `index.html`テンプレート、 `app.js`コード、 `app.js`スタイルはほとんどあり`app.css`。のファイルは、Gitとエディタのとなものです。してみる





オンラインでなをむ https://riptutorial.com/ja/angularjs/topic/6077/な

45: の - グランツ

Examples

プリロードをする

のビューがされると、Angularはそのビューをするためにをいます。のプロジェクトでは、ビューがになり、アプリケーションのがするがあります。

いは、のプロジェクトですべてのビューをにプリロードすることです。なプロジェクトの、それらをいくつかののあるまとまりにすることはいことですが、をするためにはのいくつかのがです。このタスクをするには、GruntタスクまたはGulpタスクをするとです。

ビューをプリロードするために、オブジェクトをできます。これはオブジェクトであり、はサーバーからしたすべてのビューをします。

すべてのビューを1つのmodule-jsファイルにするモジュールをすることはです。に、そのモジュールをアプリケーションにするがあります。それはそれです。

すべてのビューのファイルをするには、このタスクをできます

```
module.exports = function (grunt) {
 //set up the location of your views here
 var viewLocation = ['app/views/**.html'];

 grunt.initConfig({
        pkg: require('./package.json'),
            //section that sets up the settings for concatenation of the html files into one
file
            html2js: {
                        options: {
                            base: '',
                            module: 'app.templates', //new module name
                            singleModule: true,
                            useStrict: true,
                            htmlmin: {
                                collapseBooleanAttributes: true,
                                collapseWhitespace: true
                            }
                        },
                        main: {
                            src: viewLocation,
                            dest: 'build/app.templates.js'
                        }
                    },
            //this section is watching for changes in view files, and if there was a change,
it will regenerate the production file. This task can be handy during development.
            watch: {
                views:{
                    files: viewLocation,
                    tasks: ['buildHTML']
                },
```

```
            }
        });

        //to automatically generate one view file
        grunt.loadNpmTasks('grunt-html2js');

        //to watch for changes and if the file has been changed, regenerate the file
        grunt.loadNpmTasks('grunt-contrib-watch');

        //just a task with friendly name to reference in watch
        grunt.registerTask('buildHTML', ['html2js']);
};
```

のこのをするには、2つのをうがあります。あなたに`index.html`ファイルあなたは、ビュ―のファイルをするがあります

```
<script src="build/app.templates.js"></script>
```

あなたのアプリケ―ションをしているファイルで、をするがあります

```
angular.module('app', ['app.templates'])
```

あなたが`ui-router`ようなルータをしている、ではありません。どのようにテンプレートをしていますか

```
    .state('home', {
        url: '/home',
        views: {
            "@": {
                controller: 'homeController',
                //this will be picked up from $templateCache
                templateUrl: 'app/views/home.html'
            },
        }

    })
```

スクリプトの

JSファイルをしてサイズをさくすることをおめします。なプロジェクトの、のJSファイルがするがあり、サ―バ―から々にファイルをロ―ドするためのながされます。

のためには、すべてのにをけることがです。になのは、なである。 は、とのがされ、ながわれなければがされます。

minificaitonに`$scope`と`myService`はのにきえられます。 Angular dependency injectionはにづいてするため、としてこれらのはわるべきではありません

```
.controller('myController', function($scope, myService){
})
```

Angularはのをします。これは、のリテラルをしないためです。

```
.controller('myController', ['$scope','myService', function($scope, myService){
}])
```

- まず、すべてのファイルをエンドツ―エンドでします。
- に、 `ng-annotate`モジュ―ルをして、のコ―ドをします
- に、 `uglify`モジュ―ルをします。

module.exports = functiongrunt{//コードでスクリプトをするためのスクリプトのをします。var scriptLocation = ['app / scripts / *。js'];

```
 grunt.initConfig({
        pkg: require('./package.json'),
            //add necessary annotations for safe minification
         ngAnnotate: {
            angular: {
                src: ['staging/concatenated.js'],
                dest: 'staging/anotated.js'
            }
        },
        //combines all the files into one file
        concat: {
                js: {
                    src: scriptLocation,
                    dest: 'staging/concatenated.js'
                }
            },
        //final uglifying
        uglify: {
            options: {
                report: 'min',
                mangle: false,
                sourceMap:true
            },
            my_target: {
                files: {
                    'build/app.min.js': ['staging/anotated.js']
                }
            }
        },

        //this section is watching for changes in JS files, and if there was a change, it will
regenerate the production file. You can choose not to do it, but I like to keep concatenated
version up to date
        watch: {
            scripts: {
                files: scriptLocation,
                tasks: ['buildJS']
            }
        }

});

    //module to make files less readable
    grunt.loadNpmTasks('grunt-contrib-uglify');
```

```
    //mdule to concatenate files together
    grunt.loadNpmTasks('grunt-contrib-concat');

    //module to make angularJS files ready for minification
    grunt.loadNpmTasks('grunt-ng-annotate');

    //to watch for changes and if the file has been changed, regenerate the file
    grunt.loadNpmTasks('grunt-contrib-watch');

    //task that sequentially executes all steps to prepare JS file for production
    //concatinate all JS files
    //annotate JS file (prepare for minification
    //uglify file
     grunt.registerTask('buildJS', ['concat:js', 'ngAnnotate', 'uglify']);
};
```

オンラインでの - グランツをむ https://riptutorial.com/ja/angularjs/topic/4434/の---グランツ

46: め

き

ng-viewは、ビュー―がりえられるコンテナとしてがされるみみディレクティブの1つです。 {info} ngRouteはもはやのangular.jsファイルのではないので、のjavascriptファイルのろにangular-route.jsファイルをめるがあります。 $ routeProviderの "when"をってをすることができます。にルーートをし、に2のパラメ―タでtemplateUrlプロパティとcontrollerプロパティをつオブジェクトをするがあります。

Examples

め

`ng-view`は`$route`メインペ―ジレイアウトのなをうためにされるです。このでは、Index.htmlがメインファイルであり、ユーザ―が "/"ルートにいたときに、template.html home.htmlが`ng-view`がされているIndex.htmlにレンダリングされます。

```
angular.module('ngApp', ['ngRoute'])

.config(function($routeProvider){
  $routeProvider.when("/",
    {
      templateUrl: "home.html",
      controller: "homeCtrl"
    }
  );
});

angular.module('ngApp').controller('homeCtrl',['$scope', function($scope) {
  $scope.welcome= "Welcome to stackoverflow!";
}]);

//Index.html
<body ng-app="ngApp">
    <div ng-view></div>
</body>

//Home Template URL or home.html
<div><h2>{{welcome}}</h2></div>
```

ナビゲ―ション

1. モジュ―ルをアプリケ―ションにインジェクトする

```
  var Registration=angular.module("myApp",["ngRoute"]);
```

2. は "ngRoute"の$ routeProviderをします

```
    Registration.config(function($routeProvider) {

});
```

3. に、ルートをし、アプリケーションが "/ add"をした、アプリケーションに "/ add"ルーティングをしてregi.htmにします

```
 Registration.config(function($routeProvider) {
    $routeProvider
    .when("/add", {
        templateUrl : "regi.htm"
    })
});
```

オンラインでめをむ https://riptutorial.com/ja/angularjs/topic/8833/め

47: みみディレクティブの

Examples

HTMLの/

このでは、show htmlをにします。

```
<!DOCTYPE html>
<html ng-app="myDemoApp">
  <head>
    <script src="https://code.angularjs.org/1.5.8/angular.min.js"></script>
    <script>

      function HideShowController() {
        var vm = this;
        vm.show=false;
        vm.toggle= function() {
          vm.show=!vm.show;
        }
      }

      angular.module("myDemoApp", [/* module dependencies go here */])
        .controller("hideShowController", [HideShowController]);
    </script>
  </head>
  <body ng-cloak>
    <div ng-controller="hideShowController as vm">
      <a style="cursor: pointer;" ng-show="vm.show" ng-click="vm.toggle()">Show Me!</a>
      <a style="cursor: pointer;" ng-hide="vm.show" ng-click="vm.toggle()">Hide Me!</a>
    </div>
  </body>
</html>
```

ライブデモ

ステップバイステップの

1. `ng-app="myDemoApp"`、ngApp ディレクティブは、DOMが "myDemoApp"というの angular.moduleによってされていることをにえます。
2. `<script src="[//angular include]">`みます。
3. `HideShowController`は、のをすのにつ、 `toggle`というののをんでされてい`toggle`。
4. `angular.module(...)`は、しいモジュールをします。
5. `.controller(...)` であり、のためのモジュールをします。
6. `ng-controller` は、Model-View-Controllerパターンのにあるをどのようにサポートするかのなです。
7. `ng-show` ディレクティブは、されたがtrueの、されたHTMLをします。
8. `ng-hide` ディレクティブは、されたがtrueの、されたHTMLをします。
9. `ng-click` ディレクティブは、コントローラのトグルをします。

オンラインでみみディレクティブのをむ https://riptutorial.com/ja/angularjs/topic/7644/みみディレクティブの

48: みみのヘルパー

Examples

angular.equals

`angular.equals`は、2つのオブジェクトまたはがしいかどうかをしてします。ののうちなくとも1つがたされているにのみ、 `angular.equals`はいをし、trueをします。

angular.equalsvalue1、value2

1. オブジェクトまたはが`===`をす
2. のオブジェクトまたはのがじで、そのプロパティのすべてが`angular.equals`をしてしい
3. のが`NaN`しい
4. のがじのをします。

このは、なるではなく、オブジェクトまたはをそのまたはでにするがあるにちます。

```
angular.equals(1, 1) // true
angular.equals(1, 2) // false
angular.equals({}, {}) // true, note that {}==={} is false
angular.equals({a: 1}, {a: 1}) // true
angular.equals({a: 1}, {a: 2}) // false
angular.equals(NaN, NaN) // true
```

angular.isString

`angular.isString`は、えられたオブジェクトまたはが`string`にtrueをし`string`

angular.isStringvalue1

```
angular.isString("hello") // true
angular.isString([1, 2]) // false
angular.isString(42) // false
```

これは、

```
typeof someValue === "string"
```

angular.isArray

`angular.isArray`は、されたオブジェクトまたはが`Array`であるにのみtrueをします。

angular.isArrayvalue

```
angular.isArray([]) // true
```

```
angular.isArray([2, 3]) // true
angular.isArray({}) // false
angular.isArray(17) // false
```

それは

```
Array.isArray(someValue)
```

angular.merge

angular.mergeは、オブジェクトをくするために、ソースオブジェクトからすべてのなプロパティをります。

このは、されているオブジェクトへのをします

angular.mergeり、り

```
angular.merge({}, {}) // {}
angular.merge({name: "king roland"}, {password: "12345"})
// {name: "king roland", password: "12345"}
angular.merge({a: 1}, [4, 5, 6]) // {0: 4, 1: 5, 2: 6, a: 1}
angular.merge({a: 1}, {b: {c: {d: 2}}}) // {"a":1,"b":{"c":{"d":2}}}
```

angular.isDefinedおよびangular.isUndefined

されている、`angular.isDefined`はをテストします。

angular.isDefinedsomeValue

これは、

```
value !== undefined; // will evaluate to true is value is defined
```

```
angular.isDefined(42) // true
angular.isDefined([1, 2]) // true
angular.isDefined(undefined) // false
angular.isDefined(null) // true
```

`angular.isUndefined`は、がであるかどうかをテストします `angular.isDefined`とにです

angular.isUndefinedsomeValue

これは、

```
value === undefined; // will evaluate to true is value is undefined
```

あるいはに

```
!angular.isDefined(value)
```

```
angular.isUndefined(42) // false
angular.isUndefined(undefined) // true
```

angular.isDate

`angular.isDate`は、されたオブジェクトがDateのにのみtrueをします。

angular.isDate

```
angular.isDate("lone star") // false
angular.isDate(new Date()) // true
```

angular.isNumber

`angular.isNumber`は、されたオブジェクトまたはがNumberのにのみtrueをします。これには+ Infinity、-Infinity、およびNaNがまれます。

angular.isNumber

このは、をきこさない

```
"23" == 23 // true
```

```
angular.isNumber("23") // false
angular.isNumber(23) // true
angular.isNumber(NaN) // true
angular.isNumber(Infinity) // true
```

このは、をきこさない

```
"23" == 23 // true
```

angular.isFunction

`angular.isFunction`は、されたがへのであるにのみtrueをしします。

このは、されているオブジェクトへのをします

angular.isFunctionfn

```
var onClick = function(e) {return e};
angular.isFunction(onClick); // true

var someArray = ["pizza", "the", "hut"];
angular.isFunction(someArray ); // false
```

angular.toJson

`angular.toJson`はオブジェクトをし、それをJSONのにシリアルします。

ネイティブ`JSON.stringify`とはなり、このは`$$`まるすべてのプロパティをしますは`$$`プロパティのにけられます

```
angular.toJson(object)
```

ネットワークをするにデータをシリアライズするがあるため、このはJSONにするデータをにするのにです。

このは、 `.toString`メソッドがするのとにするため、デバッグにもです。

```
angular.toJson({name: "barf", occupation: "mog", $$somebizzareproperty: 42})
// "{"name":"barf","occupation":"mog"}"
angular.toJson(42)
// "42"
angular.toJson([1, "2", 3, "4"])
// "[1,"2",3,"4"]"
var fn = function(value) {return value}
angular.toJson(fn)
// undefined, functions have no representation in JSON
```

angular.fromJson

`angular.fromJson`は、なJSONをシリアルし、ObjectまたはArrayをします。

angular.fromJsonstring | object

このはだけにされず、されたオブジェクトのをすることにしてください。

```
angular.fromJson("{\"yogurt\": \"strawberries\"}")
// Object {yogurt: "strawberries"}
angular.fromJson('{jam: "raspberries"}')
// will throw an exception as the string is not a valid JSON
angular.fromJson(this)
// Window {external: Object, chrome: Object, _gaq: Y, angular: Object, ng339: 3…}
angular.fromJson([1, 2])
// [1, 2]
typeof angular.fromJson(new Date())
// "object"
```

angular.noop

`angular.noop`はをしないです。もしないのをするがあるは`angular.noop`をします。

angular.noop

`angular.noop`のないは、にかがされたときにエラーをすにのコールバックをえることです。

```
$scope.onSomeChange = function(model, callback) {
    updateTheModel(model);
    if (angular.isFunction(callback)) {
        callback();
    } else {
        throw new Error("error: callback is not a function!");
    }
};

$scope.onSomeChange(42, function() {console.log("hello callback")});
// will update the model and print 'hello callback'
$scope.onSomeChange(42, angular.noop);
// will update the model
```

そのの

```
angular.noop() // undefined
angular.isFunction(angular.noop) // true
```

angular.isObject

`angular.isObject`は、されたがオブジェクトであるにのみtrueをします。このは、Arrayにしてもtrueをし、 `typeof null`が`object`あっても`null`にしてfalseをし`object`。

angular.isObject

このは、されたオブジェクトをするがあるときにタイプのチェックにちます。

```
angular.isObject({name: "skroob", job: "president"})
// true
angular.isObject(null)
// false
angular.isObject([null])
// true
angular.isObject(new Date())
// true
angular.isObject(undefined)
// false
```

angular.isElement

`angular.isElement`は、されたがDOMまたはjQueryラップされたであるにtrueをします。

angular.isElementelem

このは、されたがそのようにされるのであるかどうかをチェックするのにです。

```
angular.isElement(document.querySelector("body"))
// true
angular.isElement(document.querySelector("#some_id"))
// false if "some_id" is not using as an id inside the selected DOM
angular.isElement("<div></div>")
// false
```

angular.copy

`angular.copy`は、オブジェクト、、またはをとり、そのオブジェクトのいコピーをします。

angular.copy

オブジェクト

```
let obj = {name: "vespa", occupation: "princess"};
let cpy = angular.copy(obj);
cpy.name = "yogurt"
// obj = {name: "vespa", occupation: "princess"}
// cpy = {name: "yogurt", occupation: "princess"}
```

```
var w = [a, [b, [c, [d]]]];
var q = angular.copy(w);
// q = [a, [b, [c, [d]]]]
```

のでは、`.equals`をでテストするため`angular.equals(w, q)`はtrueとされます。`w === q`は、オブジェクトとのながによってわれるため、falseとされます。

`angular.identity`は、されたのをします。

angular.identity

このはプログラミングにです。されるがされなかったにこのをデフォルトとしてすることができます。

```
angular.identity(42) // 42
```

```
var mutate = function(fn, num) {
    return angular.isFunction(fn) ? fn(num) : angular.identity(num)
}

mutate(function(value) {return value-7}, 42) // 35
mutate(null, 42) // 42
mutate("mount. rushmore", 42) // 42
```

angular.forEach

`angular.forEach`は、オブジェクトとをけれます。に、オブジェクトのなプロパティ/にしてをします。このはにしてもします。

`Array.prototype.forEach`のJSバージョンとされたプロパティプロトタイププロパティをしませんが、は`null`または`undefined`をしないでします。

angular.forEachオブジェクト、、キー{//};

```
angular.forEach({"a": 12, "b": 34}, (value, key) => console.log("key: " + key + ", value: " +
```

```
value))
// key: a, value: 12
// key: b, value: 34
angular.forEach([2, 4, 6, 8, 10], (value, key) => console.log(key))
// will print the array indices: 1, 2, 3, 4, 5
angular.forEach([2, 4, 6, 8, 10], (value, key) => console.log(value))
// will print the array values: 2, 4, 6, 7, 10
angular.forEach(undefined, (value, key) => console.log("key: " + key + ", value: " + value))
// undefined
```

オンラインでみみのヘルパーをむ https://riptutorial.com/ja/angularjs/topic/3032/みみのヘルパー

49: $スコープ

Angularはスコープのツリーをして、コントローラ、ディレクティブなどのロジックをビューにバインドし、AngularJSののなメカニズムです。スコープのについては、 docs.angularjs.orgをしてください。

ツリーのルートは、なサービス**$ rootScope**でアクセスです 。すべての$スコープは、$スコープのメソッドとプロパティをし、がサービスをせずにメソッドにアクセスできるようにします。

Examples

$ scopeのな

```
angular.module('app', [])
.controller('myController', ['$scope', function($scope){
    $scope.person = { name: 'John Doe' };
}]);

<div ng-app="app" ng-conroller="myController">
    <input ng-model="person.name" />
    <div ng-repeat="number in [0,1,2,3]">
        {{person.name}} {{number}}
    </div>
</div>
```

このでは、ng-repeatディレクティブは、しくされたにしてしいスコープをします。

これらのされたスコープは、スコープのこのはmyControllerによってされたスコープです。したがって、それらはpersonなどのすべてのをします。

プリミティブのをける

JavaScriptでは、りてプリミティブ オブジェクト、、、などのくのよりは、りてられたをするメモリのアドレスをします。

2つのなるにプリミティブ、、ブール、またはシンボルをりて、そのをしても、がされることはありません。

```
var x = 5;
var y = x;
y = 6;
console.log(y === x, x, y); //false, 5, 6
```

しかし、プリミティブなでは、のがじオブジェクトへのをしているだけなので、のをするともうのがされます

```
var x = { name : 'John Doe' };
```

```
var y = x;
y.name = 'Jhon';
console.log(x.name === y.name, x.name, y.name); //true, John, John
```

angleでは、スコープがされると、そのスコープはのすべてのプロパティにりてられますが、そののプロパティのは、スコープがプリミティブのであるにのみスコープにします。

```
angular.module('app', [])
.controller('myController', ['$scope', function($scope){
    $scope.person = { name: 'John Doe' }; //non-primitive
    $scope.name = 'Jhon Doe'; //primitive
}])
.controller('myController1', ['$scope', function($scope){}]);

<div ng-app="app" ng-controller="myController">
    binding to input works: {{person.name}}<br/>
    binding to input does not work: {{name}}<br/>
    <div ng-controller="myController1">
        <input ng-model="person.name" />
        <input ng-model="name" />
    </div>
</div>
```

Angularスコープでは、みみディレクティブやカスタムディレクティブ、 `$scope.$new()`など、さまざまなでき、スコープツリーをすることはです。

スコープのプロパティとしてプリミティブなのみをすると、しないプロパティやスコープのをしているのケースがなをき、なにかれます。

アプリでな

このアプローチは、スコープツリーにがされているとスコープをすることをプログラマがえておくがあるため、アプリケーションのとしてはいとえられます。くの、サービスをすることがましいでしょう - スコープのとをします 。

このでは、アプリケーションをするベストプラクティスではなく、スコープをどのようにしてニーズをできるかをしています。

によっては、スコープのをしてをrootScopeのプロパティとしてすることもできます。このように、アプリのすべてのスコープされたスコープをくはこのをし、アプリのどこからでもびすことができます。

```
angular.module('app', [])
.run(['$rootScope', function($rootScope){
    var messages = []
    $rootScope.addMessage = function(msg){
        messages.push(msg);
    }
}]);

<div ng-app="app">
    <a ng-click="addMessage('hello world!')">it could be accsessed from here</a>
```

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

inner.html

```
<div>
    <button ng-click="addMessage('page')">and from here to!</button>
</div>
```

カスタム$ scopeイベントの

のHTMLとに、$スコープはのイベントをつことができます。 $ scopeイベントは、のですることができます

```
 $scope.$on('my-event', function(event, args) {
    console.log(args); // { custom: 'data' }
});
```

イベントリスナーのをするがある、 **$ on**はバインドをします。のをするには

```
var unregisterMyEvent = $scope.$on('my-event', function(event, args) {
    console.log(args); // { custom: 'data' }
    unregisterMyEvent();
});
```

の$スコープイベント**$ broadcast**と**$ emit**をトリガするには、2つのがあります。のイベントのスコープをにするには、 **$ emit**をします

```
$scope.$emit('my-event', { custom: 'data' });
```

のでは、スコープの`my-event`イベントリスナーをトリガーし、イベントの`stopPropagation`をリスナーがびさないり、スコープツリーを**$ rootScope**までけます。 **$ emit**でトリガされたイベントだけが`stopPropagation`ことが`stopPropagation`

$ emitのは**$ broadcast**です 。スコープのであるスコープツリーのすべてのスコープで、 **$ broadcast**をびしたイベントリスナーをトリガーします。

```
$scope.$broadcast('my-event', { custom: 'data' });
```

$ブロードキャストでトリガーされたイベントはキャンセルできません。

$ scopeをう

$ rootscopeのにはがありますが、$ scopeサービスによってされるコードのどのも$ scopeをすることもできます。たとえば、コントローラー。

コントローラ

```
myApp.controller('myController', ['$scope', function($scope){
    $scope.myFunction = function () {
        alert("You are in myFunction!");
    };
}]);
```

これでコントロ―ラからをびすことができます

```
$scope.myfunction();
```

または、そののコントロ―ラのにあるHTMLをして

```
<div ng-controller="myController">
    <button ng-click="myFunction()"> Click me! </button>
</div>
```

はスコ―プをできるのです

```
myApp.directive('triggerFunction', function() {
    return {
        scope: {
            triggerFunction: '&'
        },
        link: function(scope, element) {
            element.bind('mouseover', function() {
                scope.triggerFunction();
            });
        }
    };
});
```

あなたのHTMLコ―ドにじコントロ―ラので

```
<div ng-controller="myController">
    <button trigger-function="myFunction()"> Hover over me! </button>
</div>
```

もちろん、じことについてngMouseoverをすることもできますが、ディレクティブのなは、みのでカスタマイズできることです。そして、あなたは$ scopeをそれらのでうをっています。

どのようにディレクティブのをすることができますか、なぜこれをうでしょうか

スコ―プは、コントロ―ラ、ディレクティブ、およびディレクティブテンプレ―トのにする「グル―」としてされます。 AngularJSアプリケ―ションがブ―トストラップされるたびに、rootScopeオブジェクトがされます。コントロ―ラ、ディレクティブ、およびサ―ビスによってされるスコ―プは、rootScopeからプロトタイプにされます。

はい、のをできます。ディレクティブのためにしたスコ―プをすることで、これをうことができます。

ディレクティブスコープには3あります。

1. スコープFalseディレクティブはスコープをします
2. スコープTrueディレクティブはしいスコープをします
3. スコープ{}ディレクティブはしいスコープをします

しいスコープのディレクティブしいスコープをすると、スコープからされません。このしいスコープは、スコープからにりされているため、Isolatedスコープとばれます。どうしてされたスコープをするがあります。カスタムディレクティブをするは、ディレクティブがであり、アプリケーションのどこにでもされるようにするため、スコープをするがあります。スコープは、ディレクティブスコープにすることはありません。

されたスコープの

```
var app = angular.module("test",[]);

app.controller("Ctrl1",function($scope){
    $scope.name = "Prateek";
    $scope.reverseName = function(){
        $scope.name = $scope.name.split('').reverse().join('');
    };
});
app.directive("myDirective", function(){
    return {
        restrict: "EA",
        scope: {},
        template: "<div>Your name is : {{name}}</div>"+
        "Change your name : <input type='text' ng-model='name'/>"
    };
});
```

AngularJSはスコープに3のプレフィックスをしています

1. "@"テキストバインディング/バインディング
2. "="モデル/
3. ""ビヘイビアバインディング/メソッドバインディング

これらのはすべて、のようにのからデータをけります。

```
<div my-directive
  class="directive"
  name="{{name}}"
  reverse="reverseName()"
  color="color" >
</div>
```

オンラインで$スコープをむ https://riptutorial.com/ja/angularjs/topic/3157/-スコープ

50: 2+への

き

AngularJSはTypeScriptをしてにきされ 、ちょうどAngularにがされました。

AngularJSアプリには、プロセスをにするためにできることがたくさんあります。 のアップグレードガイドでは、アプリをリファクタリングしてしいにづけるためのいくつかの「ステップ」をできます。

Examples

あなたのAngularJSアプリをコンポーネントのにする

しいAngularフレームワークでは、 コンポーネントはユーザーインターフェイスをするなビルディングブロックです。したがって、AngularJSアプリケーションをしいAngularにするのにつのステップの1つは、それをよりコンポーネントのにリファクタリングすることです。

バージョン**1.5**のAngularJSにもコンポーネントがされました。 AngularJSアプリでコンポーネントをすると、がしいAngular 2+にづくだけでなく、モジュラーされ、しやすくなります。

さらにめるに、 コンポーネントのとについてよくされているAngularJSのドキュメントページをすることをおめします。

むしろい`ng-controller`のコードをしい`component`のスタイルにするについてのヒントをべたいといます。

あなたのアプリをコンポーネントにしめる

すべてのコンポーネントアプリケーションは、、のサブコンポーネントをむ1つまたはいくつかのコンポーネントをっています。だから、にあなたのアプリまたはそれのきなをむだけののコンポーネントをしないでください。

たちはのコントローラにりてられたコードの、っているとし`UserListController`たちがをけた、と々はそれのをりたい`UserListComponent`。

の**HTML**

```
<div ng-controller="UserListController as listctrl">
    <ul>
        <li ng-repeat="user in myUserList">
            {{ user }}
```

```
        </li>
    </ul>
</div>
```

の**JavaScript**

```
app.controller("UserListController", function($scope, SomeService) {

    $scope.myUserList = ['Shin', 'Helias', 'Kalhac'];

    this.someFunction = function() {
        // ...
    }

    // ...
}
```

しい**HTML**

```
<user-list></user-list>
```

しい**JavaScript**

```
app.component("UserList", {
    templateUrl: 'user-list.html',
    controller: UserListController
});

function UserListController(SomeService) {

    this.myUserList = ['Shin', 'Helias', 'Kalhac'];

    this.someFunction = function() {
        // ...
    }

    // ...
}
```

`$scope`をコントロ―ラにしなくても、 `$scope.myUserList`わりに`this.myUserList`をしていることに`this.myUserList $scope.myUserList`。

しいテンプレ―トファイル `user-list.component.html`

```
<ul>
    <li ng-repeat="user in $ctrl.myUserList">
        {{ user }}
    </li>
</ul>
```

たちは、をしているかにしてください`myUserList`して、コントロ―ラにし、 `$ctrl.myUserList`わりにHTMLから`$scope.myUserList`。

これは、ドキュメントをんだにおそらくしたように、テンプレートの`$ctrl`がコントローラをするようになったからです。

コントローラとルートはどうですか

あなたのコントローラが`ng-controller`わりにルーティングシステムをってテンプレートにバインドされているは、のようなものがあれば

```
$stateProvider
  .state('users', {
    url: '/users',
    templateUrl: 'user-list.html',
    controller: 'UserListController'
  })
  // ..
```

のをのようにすることができます

```
$stateProvider
    .state('users', {
      url: '/',
      template: '<user-list></user-list>'
    })
    // ..
```

はですか

アプリケーションをむコンポーネントアプリケーションまたはビューのなどがあるので、コンポーネントをのれコンポーネントにし、そのをしいサブコンポーネントにラップすることでするがあります、 々。

あなたはのようなComponentのをいめるべきです

- とのバインディング
- `$onInit()` 、 `$onChanges()`などのライフサイクルフック

コンポーネントのドキュメントをんだは、これらのコンポーネントのをすべてするをすでにっているはずですが、のなアプリケーションのがなは、 これをできます。

また、コンポーネントのコントローラのにくのロジックコードをするがある、そのロジックをサービスにすことをすることをおめします 。

コンポーネントベースのアプローチをすることで、AngularJSをづけてしいAngularフレームワー

クにさせることができますが、よりくのモジュラーをできます。

もちろん、しいAngular 2+のにんでいくためにはにもたくさんのステップがあります。これについては、のでします。

WebpackとES6モジュールの

Webpackのようなモジュールローダーをすることで、 ES6でなみみモジュールシステム TypeScriptとにをすることができます。に、 インポートとエクスポートのをして、アプリケーションのさまざまなでどのコードをできるかをできます。

アプリケーションをプロダクションにっていくと、モジュールローダによって、バッテリをめたバンドルへのパッケージがになります。

オンラインで2+へのをむ https://riptutorial.com/ja/angularjs/topic/9942/2plusへの

51: MVC

き

AngularJSでは、 **MVC**パターンはJavaScriptとHTMLでされています。ビューはHTMLでされ、モデルとコントローラはJavaScriptでされます。これらのコンポーネントをAngularJSにまとめるはいくつかありますが、もなフォームはビューからまります。

Examples

コントローラきスタティックビュー

mvcデモ

こんにちは

コントローラ

```
var indexController = myApp.controller("indexController", function ($scope) {
    // Application logic goes here
});
```

モデルにをする

```
var indexController = myApp.controller("indexController", function ($scope) {
    // controller logic goes here
    $scope.message = "Hello Hacking World"
});
```

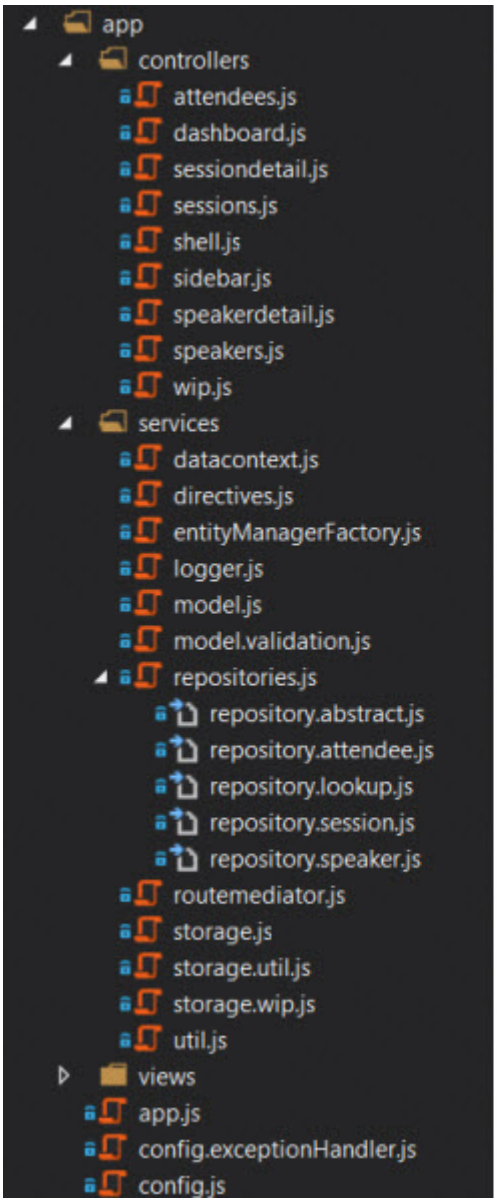
オンラインでMVCをむ https://riptutorial.com/ja/angularjs/topic/8667/mvc

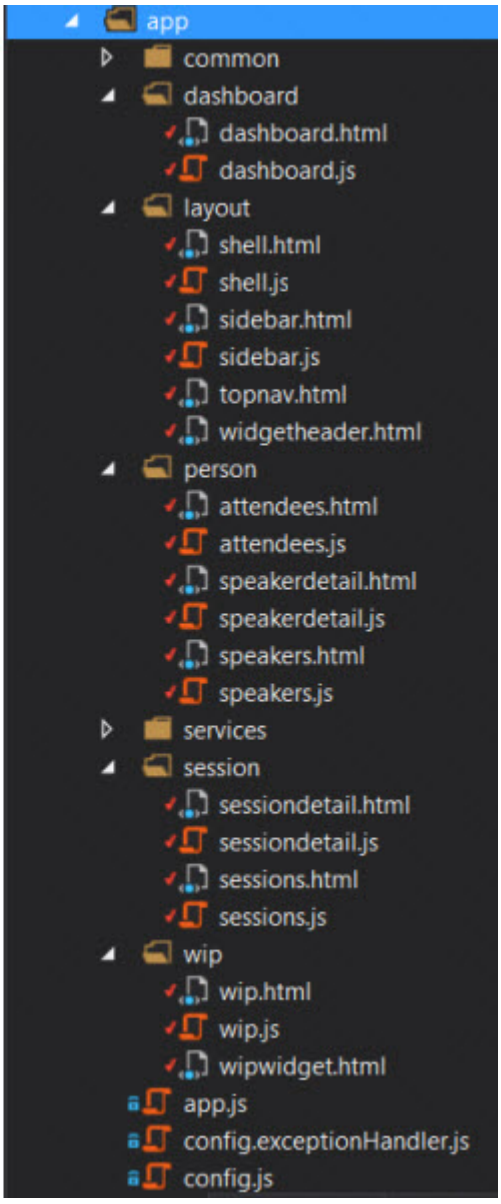
52: のあるプロジェクト - ディレクトリ

Examples

ディレクトリ

しいAngularプログラマのでよくある -「プロジェクトのはどうですか」れたは、スケーラブルなアプリケーションにちます。プロジェクトをするときには、「タイプによるべえ」と「フィーチャによるべえ」の2つのがあります。 2のがれています。になアプリケーションでは、プロジェクトのがににになります。





タイプにべえる

アプリケーションはファイルのにされています。

- アドバンテージ - さいアプリケーションの、Angularをしめたプログラマにとってはいで、2のにするのはです。
- - さなアプリケーションのでも、のファイルをつけるのがしくなります。たとえば、ビューとそのコントローラは2つの々のフォルダにあります。

にべえる

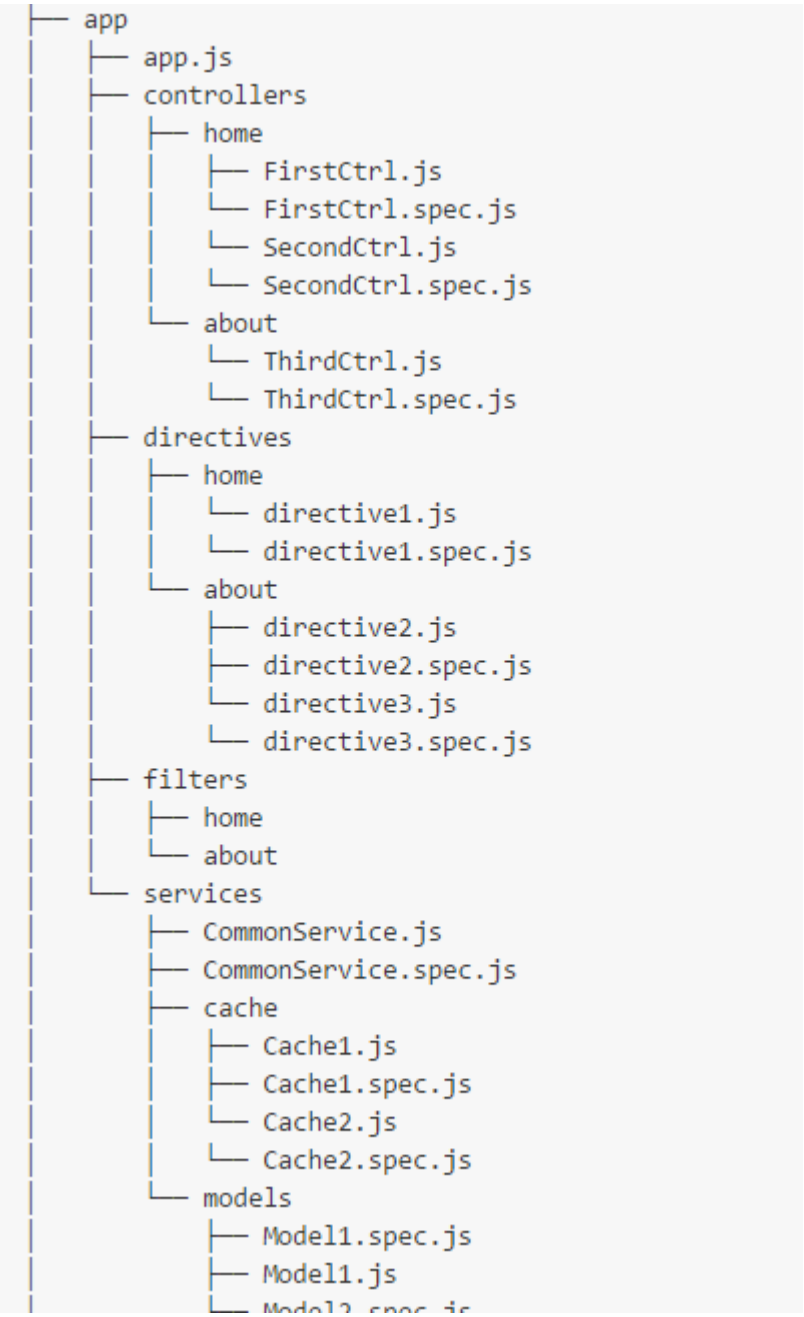
フィーチャがフィーチャのタイプにソートされているのされる。

レイアウトビューとコントローラはすべてレイアウトフォルダに、コンテンツはフォルダに、にきます。

- - のをするコードのセクションをしているとき、そのはすべて1つのフォルダにあります。
- - サービスはくのを「サービス」するため、しなります。

Angular Structureのためのリファクタリング

ののをみわせたファイル



クレジット Angular Style Guide

オンラインでのあるプロジェクト - ディレクトリをむ
https://riptutorial.com/ja/angularjs/topic/6148/のあるプロジェクト---ディレクトリ

クレジット

S. No		Contributors
1	AngularJSをいめる	Abhishek Pandey, After Class, Andrés Encarnación, AnonymousNotReally, badzilla, Charlie H, Chirag Bhatia - chirag64, Community, daniellmb, David G., Devid Farinelli, Eugene, fracz, Franck Dernoncourt, Gabriel Pires, Gourav Garg , H. Pauwelyn, Igor Raush, jengeb, Jeroen, John F., Léo Martin, Lotus91, LucyMarieJ, M. Junaid Salaat, Maaz.Musa, Matt, Mikko Viitala, Mistalis, Nemanja Trifunovic, Nhan, Nico, pathe.kiran, Patrick, Pushpendra, Richard Hamilton, Stepan Suvorov, Stephen Leppik, Sunil Lama, superluminary, Syed Priom, timbo, Ven, vincentvanjoe, Yasin Patel, Ze Rubeus, Артем Комаров
2	$ httpリクエスト	CENT1PEDE, jaredsk, Liron Ilayev
3	$ qサービスによるの	Alon Eitan, caiocpricci2, ganqqwerty, georgeawg, John F., Muli Yulzary, Praveen Poonia, Richard Hamilton, Rohit Jindal, svarog
4	AngularJSのとしとトラップ	Alon Eitan, Cosmin Ababei, doctorsherlock, Faruk Yazıcı, ngLover, Phil
5	AngularJSバインディングオプション `=`、 `@`、 `&`など	Alon Eitan, Lucas L, Makarov Sergey, Nico, zucker
6	AngularJsをしたSignalR	Maher
7	ES6をしたカスタムフィルタ	Bouraoui KACEM
8	ES6をしたコントローラ	Bouraoui KACEM
9	HTTPインターセプタ	G Akshay, Istvan Reiter, MeanMan, Mistalis, mnoronha
10	ng-classディレクティブ	Dr. Cool

11	ngModelControllerをするディレクティブ	Nikos Paraskevopoulos
12	ng-repeat	Divya Jain, Jim, Sender, zucker
13	ngRouteをしたルーティング	Alon Eitan, Alvaro Vazquez, camabeh, DotBot, sgarcia.dev, svarog
14	ng-style	Divya Jain, Jim
15	TypeScriptでAngularJSをする	Parv Sharma, Rohit Jindal
16	イベント	CodeWarrior, Nguyen Tran, Rohit Jindal, RyanDawkins, sgarcia.dev, shaN, Shashank Vivek
17	カスタムディレクティブ	Alon Eitan, br3w5, casraf, Cody Stott, Daniel, Everettss, Filipe Amaral, Gaara, Gavishiddappa Gadagi, Jinw, jkris, mnoronha, Pushpendra, Rahul Bhooteshwar, Sajal, sgarcia.dev, Stephan, theblindprophet, TheChetan, Yuri Blanc
18	カスタムフィルタ	doodhwala, Pat, Sylvain
19	コントローラ	Adam Harrison, Aeolingamenfel, Alon Eitan, badzilla, Bon Macalindong, Braiam, chatuur, DerekMT12, Dr. Cool, Florian, George Kagan, Grundy, Jared Hooper, Liron Ilayev, M. Junaid Salaat, Mark Cidade, Matthew Green, Mike, Nad Flores, Praveen Poonia, RamenChef, Sébastien Deprez, sgarcia.dev, thegreenpizza, timbo, Und3rTow, WMios
20	コントローラのまたはこの	It-Z, Jim
21	コンポーネント	Alon Eitan, Artem K., badzilla, BarakD, Hubert Grzeskowiak, John F., Juri, M. Junaid Salaat, Mansouri, Pankaj Parkar, Ravi Singh, sgarcia.dev, Syed Priom, Yogesh Mangaj
22	サービス	Abdellah Alaoui, Alvaro Vazquez, AnonDCX, DotBot, elliot-j, Flash, Gavishiddappa Gadagi, Hubert Grzeskowiak, Lex, Nishant123
23	サービスとファクトリの	Deepak Bansal
24	セッション	Rohit Jindal
25	ダイジェストループのウォークスルー	Alon Eitan, chris, MoLow, prit4fun

26	データの	elliot-j, Grundy, Lex, Mikko Viitala, Mistalis, Nix, prit4fun, Rohit Jindal, sgarcia.dev, Sunil Lama
27	データバインディングのしくみ	Lucas L, Sasank Sunkavalli, theblindprophet
28	データフィルタ、ページネーションなどのanglejs	Paresh Maghodiya
29	デコレータ	Mikko Viitala
30	デバッグ	Aman, AWolf, Vinay K
31	パフォーマンスプロファイリング	Deepak Bansal
32	ビルトインディレクティブ	Adam Harrison, Alon Eitan, Aron, AWolf, Ayan, Bon Macalindong, CENT1PEDE, Devid Farinelli, DillonChanis, Divya Jain, Dr. Cool, Eric Siebeneich, George Kagan, Grinn, gustavohenke, IncrediApp, kelvinelove, Krupesh Kotecha, Liron Ilayev, m.e.conroy, Maciej Gurban, Mansouri, Mikko Viitala, Mistalis, Mitul, MoLow, Naga2Raja, ngLover, Nishant123, Piet, redunderthebed, Richard Hamilton, svarog, tanmay, theblindprophet, timbo, Tomislav Stankovic, vincentvanjoe, Vishal Singh
33	フィルタ	Aeolingamenfel, developer033, Ed Hinchliffe, fracz, gustavohenke, Matthew Green, Nico
34	フォーム	Alon Eitan, fantarama, garyx, Mikko Viitala, Richard Hamilton, Rohit Jindal, shane, svarog, timbo
35	プロバイダー	Mikko Viitala
36	プロファイリングとパフォーマンス	Ajeet Lakhani, Alon Eitan, Andrew Piliser, Anfelipe, Anirudha, Ashwin Ramaswami, atul mishra, Braiam, bwoebi, chris, Dania, Daniel Molin, daniellmb, Deepak Bansal, Divya Jain, DotBot, Dr. Cool, Durgpal Singh, fracz, Gabriel Pires, George Kagan, Grundy, JanisP, Jared Hooper, jhampton, John Slegers, jusopi, M22an, Matthew Green, Mistalis, Mudassir Ali, Nhan, Psaniko, Richard Hamilton, RyanDawkins, sgarcia.dev, theblindprophet, user3632710, vincentvanjoe, Yasin Patel, Ze Rubeus
37	モジュール	Alon Eitan, Ankit, badzilla, Bon Macalindong, Matthew Green, Nad Flores, ojus kulkarni, sgarcia.dev, thegreenpizza
38	ユー·ルーター	George Kagan, H.T, Michael P. Bazos, Ryan Hamley, sgarcia.dev

39	レイジーローディング	Muli Yulzary
40		Andrea, badzilla, Gavishiddappa Gadagi, George Kagan, MoLow, Omri Aharon
41	テスト	daniellmb, elliot-j, fracz, Gabriel Pires, Nico, ronapelbaum
42		ziaulain
43		Sylvain
44	な	Mikko Viitala
45	の - グランツ	JanisP
46	め	Aayushi Jain, Manikandan Velayutham, Umesh Shende
47	みみディレクティブの	Gourav Garg
48	みみのヘルパー	MoLow, Pranav C Balan, svarog
49	$スコープ	Abhishek Maurya, elliot-j, Eugene, jaredsk, Liron Ilayev, MoLow, Prateek Gupta, RamenChef, ryansstack, Tony
50	2+への	ShinDarth
51	MVC	Ashok choudhary, Gavishiddappa Gadagi, Jim
52	のあるプロジェクト - ディレクトリ	jitender, Liron Ilayev