AngularJS

[JavaScript][AngularJS x Django][Bower]

・以下の本を試す

準備

・https://angularjs.org/

動作確認

・html タグに ng-app ディレクティブを追加
・実行すると、「Hello AngularJS!」と表示される

```
<html ng-app>
<head>
  <script     type="text/javascript"     src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.12
/angular.min.js"></script>
</head>
<body>
{{'Hello AngularJS!'}}
</body>
</html>
```

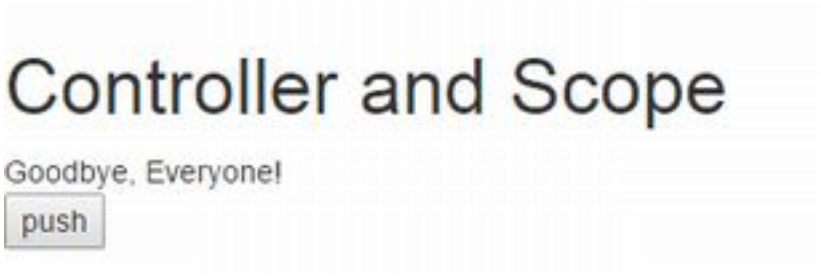
基本機能

・テンプレートエンジン
・ディレクティブ
・双方向データバインディング

```
<html ng-app>
<head>
<meta charset="UTF-8">
<title>AngularJS Test</title>
<script type="text/javascript" src="/static/js/angular.min.js"></script>
</head>
<body>
<h1>Literal</h1>
{{ 'Hello' + ' AngularJS!' }}
<h1>Expression</h1>
{{ 2 * 10 }}
<h1>Variable</h1>
<div ng-init="mynumber=123"/>
{{ mynumber }}
<h1>Directive</h1>
<h1>if</h1>
<div ng-init="showMessage=true"/>
<div ng-if="showMessage">
  Show Message is {{ showMessage }}.
</div>
<h2>repeat</h2>
<div ng-init="items=['orange','apple','pine','melon']"/>
<ul>
  <li ng-repeat="item in items">{{ item }}</li>
</ul>
<h1>Bi-directionally data binding</h1>
<input type="text" ng-model="message" /> {{ message }}
</body>
</html>
```

コントローラとスコープ

・MyController で、$scope オブジェクトにプロパティやメソッドを登録
・angular.module() でモジュール "app" を作成
・app.controller() でコントローラ myController を登録
・<html ng-app="app"> と記述し app モジュールを利用可能にする
・テンプレートからコントローラを利用するには、ng-controller ディレクティブを使いコントローラ名を指定
・$scope.message に指定した値が、{{message}} と記述した箇所に展開される



test_ag_basic2.html

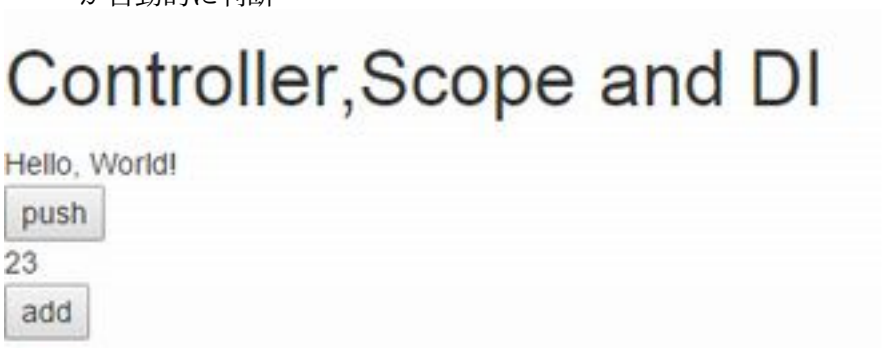
```
<!DOCTYPE html>
<html ng-app="app">
<head>
<meta charset="UTF-8">
<title>AngularJS Test</title>
<script type="text/javascript" src="/static/js/angular.min.js"></script>
<link rel="stylesheet" href="/static/css/bootstrap.min.css">
<script type="text/javascript" src="/static/test/js/test_ag_basic2.js"></script>
</head>
<body>
<h1>Controller and Scope</h1>
<div ng-controller="myController">
  <div>{{ message }}</div>
  <button ng-click="action()">push</button>
</div>
</body>
</html>
```

test_ag_basic2.js

```
var MyController = function ($scope) {
  $scope.message = "Hello, World!";
  $scope.action = function() {
    $scope.message = "Goodbye, Everyone!";
  };
};
var appModule = angular.module("app", []);
appModule.controller("myController", MyController);
```

DI

- DI コンテナにより、部品間の依存関係を管理
- angular.module().value() で関数を共有できるように指定
- 共有サービスを利用するコントローラーを定義
- MyService と MyController は、異なる関数スコープの中に定義されているため、本来は呼び出せない
- MyController の引数にサービス名を記述すると、引数名からどのインスタンスを渡すべきか自動的に判断



test_ag_basic2.html

```
<body>
<h1>Controller and Scope</h1>
<div ng-controller="myController">
  <div>{{ message }}</div>
  <button ng-click="action()">push</button>
  <div>{{ addvalue }}</div>
  <button ng-click="doAdd(11,12)">add</button>
</div>
```

test_ag_basic2.js

```
appModule = angular.module("app", []);
(function(){
  var MyService = function(a, b) {
    return a + b;
  };
  appModule.value("addService", MyService);
})();

(function() {
  var MyController = function ($scope, addService) {
    $scope.message = "Hello, World!";
    $scope.action = function() {
      $scope.message = "Goodbye, Everyone!";
    };
    $scope.addvalue = 0;
    $scope.add = addService;
    $scope.doAdd = function(x, y){
      $scope.addvalue = $scope.add(x, y);
    };
  };
  appModule.controller("myController", MyController);
})();
```

ビルトインディレクティブ

・AngularJS API リファレンス
  ・https://docs.angularjs.org/api

DOM 操作

ディレクティブ	概要
ng-bind	モデルをビューにバインド
ng-bind-html	モデルを HTML にバインド。ngSanitize モジュールが読み込まれている必要がある
ng-bind-template	テンプレートをビューにバインド
ng-non-bindable	バインディングしたくない場合に指定
ng-cloak	AngularJS の初期処理がかんりょうするまで、テンプレートの表示が点滅するように表示されるのを防ぐ
ng-style	style 属性を動的に操作する
ng-class	CSS を動的に操作する
ng-class-even	ng-repeat と組み合わせる。偶数行に対して振る舞いを指定するのに利用

ng-class-odd	ng-repeat と組み合わせる。奇数行に対して振る舞いを指定するのに利用
ng-show	値の評価が true であれば、内包する DOM を表示
ng-hide	値の評価が true であれば、内包する DOM を非表示
ng-open	定義した要素をクリックすると、内包する DOM の表示・非表示を切り替え
ng-pluraize	数値によって表示を切り替える
ng-if	値が false の場合、定義した DOM を DOM ツリーから削除
ng-switch	switch 文と同様の役割
ng-repeat	テンプレート上でループを行う
ng-messages/ng-message	メッセージを扱う。messages と message は親子関係

イベント

ディレクティブ	概要
ng-click	click イベントをリスナーに登録
ng-dblclick	ダブルクリックイベントをリスナーに登録
ng-mousedown	マウスダウンイベントをリスナーに登録
ng-mouseup	マウスアップイベントをリスナーに登録
ng-mouseenter	マウスエンターイベントをリスナーに登録
ng-mouseover	マウスオーバーイベントをリスナーに登録
ng-moucemove	マウスムーブイベントをリスナーに登録
ng-mouseleave	マウスリーブイベントをリスナーに登録
ng-focus	フォーカスイベントをリスナーに登録
ng-blur	ブラーイベントをリスナーに登録
ng-keydown	キーダウンイベントをリスナーに登録
ng-keypress	キープレスイベントをリスナーに登録
ng-keyup	キーアップイベントをリスナーに登録
ng-change	チェンジイベントをリスナーに登録
ng-checked	チェックイベントをリスナーに登録
ng-copy	コピーイベントをリスナーに登録、Windows では Ctrl+C を補足する

ng-cut	カットイベントをリスナーに登録、Windows では Ctrl+X を補足する
ng-paste	ペーストイベントをリスナーに登録、Windows で Ctrl+V を補足する
ng-submit	action 属性の有無により、デフォルト動作を実行するかどうかを判定

■ $event

・$event と呼ばれるイベントオブジェクトをラップしたオブジェクトがあります。

モジュールと DI

・AngularJS には、関数やオブジェクトをグループ化して管理するモジュールという仕組みが用意されている
・ディレクティブ、サービスやフィルター、コントローラーなどすべてモジュール機能によって管理されている
・サードパーティ製ライブラリもモジュールで管理されている

モジュールの作成と取得

・angular.module を利用する
・第 2 引数があると、モジュールを作成し、無いと作成済みモジュールを参照する

```
angular.module(name, [requires], [configFn]);
```

■ ng モジュール

・ng モジュールは、依存関係を明示しなくてもどのモジュールからでも利用可能
　・ng モジュールのサービスコンポーネント

モジュールに部品を登録する

■サービス実装の比較

メソッド	サービスの DI	プリミティブ型の登録	Module#config での利用	Module#decorator による書き換え	インスタンスの生成方法
Module#value	×	○	×	○	登録したオブジェクトそのまま
Module#constant	×	○	○	×	登録したオブジェクトそのまま
Module#factory	○	○	×	○	登録した関数の戻り値として渡す

Module#service	○	×	×	○	登録したコンストラクタ関数が new される
Module#provider	○	○	○	○	$get メソッドで返す

■判定

```
if (directive や provider のコンフィギュレーションで利用するパラメータを定義したい ) {
  Module#constant
} else if ( 他のサービスを利用する必要がない ) {
  Module#value
} else if (Mudule#config で設定を変更したい ) {
  Module#privider
} else if ( プリミティブ型を扱いたい、new でインスタンスを生成したくない ){
  Module#factory
} else {
  Module#service
}
```

■ Module#value

```
angular.Moduule#value(name,object);
```

・第 2 引数には、数値、文字列、配列、オブジェクト、関数などを登録できる。

■ Module#constant

```
angular.Module#constant(name,value);
```

・第 2 引数には、Module#value と同様に、数値、文字列、配列、関数などが登録できる。

Module#constant で登録したサービスは、利用できるタイミングが他よりも早くなっているため、Module#config やプロバイダのコンストラクタにインジェクトできる。

・Module#decorator で上書きできない
・値を書き換えることはできる

■ Module#factory

```
angular.Module#factory(name,getFn);
```

・第 2 引数として、サービスとして共有したいオブジェクトを返す関数を記述する。
・サービスを定義するには一般的に、Module#factory か Module#service が利用される。
・Module#value、Module#constant で登録したサービスは、DI を使用して他のサービスをインジェクトできない。

他のサービスを利用したサービスの定義

```
var app = angular.module('app');
app.constant('apiUrl','/api/products.json');
app.constant('apiKey','hogehoge');

app.factory('productsService',['$resource','apiUrl','apiKey',
  function($resource, apiUrl, apiKey){
```

```
    return $resource(apiUrl).query({apiKey: apiKey});
}]);
```

■ Module#service

```
angular.Module#service(name,constructor);
```

・Module#factory では共有したいオブジェクトを返す関数を登録するが、Module#service では共有したいオブジェクトのコンストラクタ関数を登録する。
・サービスを定義するには一般的に、Module#factory か Module#service が利用される。
・Module#value、Module#constant で登録したサービスは、DI を使用して他のサービスをインジェクトできない。

Module#facotry、Module#service ともに、作成されたインスタンスをシングルトンとして共有することに違いはない。

他のサービスを利用したサービスの定義

```
var app = angular.module('app');
app.constant('apiUrl','/api/products.json');
app.constant('apiKey','hogehoge');

app.factory('productsService',['$resource','apiUrl','apiKey',
  function($resource, apiUrl, apiKey){
    this.get = function() {
      return $resource(apiUrl).query({apiKey: apiKey});
    }
}]);
```

■ Module#provider

```
angular.Module#provider(name, provider);
```

・Module#value、Module#service、Module#factory は内部で Module#provider を使用している。
・サービスを登録するためには、$get メソッドを持ったオブジェクトを定義する必要がある。

Module#config でパラメータのセッティングが可能

```
angular.module('app')
  .provider('MyService', function(){
    this.$get = function() {
      var aService = {};
      aService.doService = function() {
        // do something
      };
      return aService;
    }
  });
```

■ Module#controller

■ Module#filter

■ Module#directive

■ Module#animation

サービス実装例

■シングルトン

・value、constant、service、factory、provider いずれもシングルトンとして扱われる。
・この性質を利用して、複数のコントローラー間でデータの共有も可能。

```
var app = angular.module('app');
app.service('ShareService', function(){
  this.values = {};
  this.setValue = function(key, value) {
    this.values[key] = value;
  };
  this.getValue = function(key) {
    return this.values[key];
  }
});
```

■新しいインスタンスを返す

・新しいインスタンスを生成して返すサービス

定義

```
var app = angular.module('app');
app.factory('NewInstanceFactory', function(){
  function NewInstance() {
  }
  NewInstance.prototype.doSomething = function() {
    // do something
  };
  function NewInstanceFactory() {
    return new NewInstance();
  }
  return NewInstanceFactory;
});
```

利用側

```
var app = angular.module('app',[]);
app.controller('HogeController', ['$scope','NewInstanceFactory',
  function($scope,NewInstanceFactory){
    var ni = NewInstanceFactory();

    $scope.doSomething = function() {
      return ni.doSomething();
    }
  }
]);
```

スコープとコントローラ

・テンプレートファイルと JavaScript コードを結びつけるためにスコープとコントローラを使用する。

利用方法

■ ng-controller ディレクティブを使用する

```
<div ng-controller="NormalController"></div>
```

■コントローラーを定義

・Module#controller を使用する

```
angular.module('app',[])
  .controller('NormalController',['$scope',function($scope) {
    $scope.message = "Hello";
  }
}]);
```

スコープ

・テンプレートに対して公開するデータや振る舞いを定義する。
・AngularJS には、DOM と JavaScript の間にスコープと呼ばれるオブジェクトが用意されている。
・スコープのプロパティは内容の変更が監視されており、値が変化すると DOM に反映される。
・DOM の値が変更されると、対応するスコープの値に変更が反映される。
・DOM イベントが発生した場合、対応するスコープのメソッドが呼び出される。
・DOM 操作は AngularJS が隠蔽するため、開発者はスコープオブジェクトを操作することで、メンテナンス性の高いコードで、画面の描画内容を変更できる。

コントローラの役割

・コントローラはスコープをセットアップする。
・DI によりインジェクトされたサービスを利用して $scope オブジェクトを組み立てる。

サーバーサイドの MVC パターンでのコントローラとは役割が異なるため、名称に惑わされないように注意
プレゼンテーションロジックはフィルターやディレクティブに、ビジネスロジックはサービスに分離し、コントローラはあくまでもスコープのセットアップ処理に徹するのが理想的

スコープの適用範囲

・ng-controller ディレクティブを使用してコントローラーを指定した場合、その要素より下位階層の要素でのみ、そのコントローラーでセットアップしたスコープオブジェクトを利用できる。
・コントローラーは適用するごとに新しいスコープのインスタンスが生成される。
・同一のコントローラーを複数個所に適用した場合、異なるインスタンスになる。

■階層化

・コントローラーの階層化も可能
・子コントローラーに渡されるスコープは親コントローラーのスコープの派生インスタンス

トップ階層にあるコントローラに渡されるスコープオブジェクトは、$rootScope から派生したものとなる

```
angular.module('app',[])
  .controller('ParentController',[$scope, function($scope){
    $scope.value = 'Parent';
  }])
    .controller('ChildController',[$scope, function($scope){
      $scope.value = 'Child';
    }]);
```

■ $rootScope

・$rootScope は通常の $scope 同様、コントローラにインジェクトして利用できる
・$rootScope のインスタンスは、アプリケーション内で必ず 1 つのため、複数のコントローラー間で同一のインスタンスを参照可能

■スコープとしてのコントローラー

・コントローラー関数をスコープオブジェクトとして利用する方法もある。
・ng-controller で　コントローラー as 別名 とする

```
<div ng-controller="AsSyntaxController as ctrl">
  <p>value: {{ctrl.vaue}}</p>
  <p>upperValue: {{ctrl.getUpperValue()}}</p>
</div>
```

・コントローラー定義では、引数に $scope を受け取らず、テンプレートに公開するプロパティやメソッドを自身 (this) に登録する

```
angular.module('app',[])
  .controller('AsSyntaxController', function(){
    this.value = "Hello";
    this.getUpperValue = function(){
      return angular.uppercase(this.value);
    };
  });
```

■スコープ間連携

大規模アプリケーションを開発する場合、再利用しやすい単位ごとにコントローラーを作成したり、画面領域ごとにコントローラを分割するのが一般的

・コントローラーを分割すると、コントローラー間でのデータとやり取りや、相互に通信する手段が必要になる
・$scope オブジェクトには、コントローラー間でイベントを通知する仕組みが用意されている。

$rootScope.Scope#$emit(name, ...args);

・派生元スコープに対してイベントを送信

$rootScopep.Scope#$broadcast(name ...args);

・派生先のスコープに対してイベントを送信ん

$rootScope.Scope#$on(name, listener);

・Scope#$emit、Scope#$broadcast で送信したイベントを受け取る

スコープの監視と反映

変更監視

・スコープオブジェクトにはデータの変化を監視するために、$watch、$watchGroup、$watchCollection が用意されている。

■ $rootScope.Scope#$watch

```
$rootScope.Scope#$watch(watchExpression, [listener],[objectEquality]);
```

・第 1 引数には、監視対象のプロパティを文字列で指定する方法と、監視対象の値を返す関数を登録する方法がある
・第 2 引数には、値が変化したときに呼び出されるリスナー関数を登録
・第 3 引数に true を指定すると、オブジェクトを比較するときに angular.equals を利用する。false(デフォルト) の場合、== での比較
・戻り値はリスナー関数を解除する関数。

```
angula.module('app')
  .controller('WatchController',['$scope',function($scope){
    $scope.message = 'Hello';
    $scope.result = '';
    $scope.$watch('message',function(newVal, oldVal, scope){
      if(angular.equlas(newVal,'success') {
        $scope.result = 'OK';
      }
    });
  }]);
```

■ $rootScope.Scope#watchGroup

```
$rootScope.Scope#watchGroup(watchExpressions, listener);
```

・複数のデータ値をまとめてチェックしたい場合

```
angular.module('app')
  .controller('WatchGroupController',['$scope',function($scope){
    $scope.greeting = 'Hello';
    $scope.message2 = 'Hello';
    $scope.message3 = 'Hello';
    $scope.result = '';
    $scope.watchGroup([
      function () { return $scope.greeting; },
      function () { return $scope.message2; },
      function () { return $scope.message3; }
    ],
    function(newVal, oldVal, scope) {
      scope.result = angular.toJson(newVal);
    }
    );
  }]);
```

ルーティング

HTML5 History API

・HTMLL5 で追加された History API history#pushState 関数 および popState イベントが利用されている。
・history.pushState() は、URL の表示をページ読み込みなしに変更出来る。

pushState

```
history.pushState('','',url);
```

popState

```
window.addEventListener('popstate', function(event){
});
```

AngularJS では、ng-route モジュールで簡単に利用出来る

ng-route の利用

■モジュールの読み込み

・CDN

```
<script type="text/javascript" src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.15/angular-route.min.js"></script>
```

Hash モードと html5 モード

・デフォルトは Hash モード。http://example.jp/#/angular ような URL が呼び出される。
・pushState による操作を実行するためには、$locationProvider.html5Mode(true) を呼び出し、html5 モードに変更する必要がある

IE9 以下では、pushState が実装されていないため、自動で Hash モードにフォールバックされる

ルーティングの設定

■ when 関数の第 2 引数

KEY	型	説明
controller	文字列 or 関数	コントローラーを使用
controllerAs	文字列	コントローラーンのエイリアスを使用可能
template	文字列 or 関数	文字列もしくは関数の戻り値でテンプレートを設定。ng-view に展開される。templateUrl が優先
templateUrl	文字列 or 関数	文字列もしくは関数の戻り値でテンプレートのパスを指定
reloadOnSearch	真偽値	$location#search、$location#hash で変更時に再読み込みされる route を指定。false で、URL が変更されたら、$routeScope の $routeUpdate イベントが呼ばれる
caseInsensitiveMatch	真偽値	大文字小文字が区別されずに評価。デフォルト false

■ app.js

```
var app = angular.module('app', ['ngRoute'])
  .config(['$routeProvider','$locationProvider', function( $routeProvider, $locationProvider){
    $routeProvider
      .when('/page/:name',{
        templateUrl: 'partials/page.html',
        controller: 'PageController',
        controllerAs: 'page'
      })
      .otherwise({
        redirectTo: '/'
      })
      ;
    $locationProvider.html5Mode(true);
  }]);
app.controller('PageController',['$routeParams',function($routeParams){
  this.name = 'PageController';
  this.params = $routeParams;
```

■ app.html

・https://docs.angularjs.org/error/$location/nobase

```
<html ng-app="app">
<head>
<base href="http://127.0.0.1/">
</head>
<body>
  <div ng-controller="PageController as page">
    <a href="page/page1">page1</a>
    <a href="page/page2">page2</a>
    <div ng-view></div>
  </div>
    :
</html>
```

■ partials/page.html

```
<div>
Page:{{ page.params.name }}
</div>
```

プロミス

・AngularJS では、ルーティングと呼ばれる技術を利用して、ページの部分更新やデータの取得動的な表示更新などを最小のコード量で簡単に実現できる。
・$http など通信処理を行う際にプロミスを知っていると便利。
・JavaScript の特徴であるノンブロッキング処理を適切に処理するために、コールバックとプロミスがある。

プロミスとは

・プロミスとは関数の結果を返すのではなく、プロミスオブジェクトを返す。
・同期的関数と同じように値を返すことが可能。
・ノンブロッキング処理内で成功と失敗に対するメソッドを呼び出せて、then メソッドにより次ぎ処理を呼び出せる。

コールバックを利用すると、ノンブロッキング処理内でコールバックが続くことが多くなり、「コールバック地獄」とも呼ばれる状態に陥り、1 つの関数が肥大化していくことが多々ある。これを解決するために用意されているのがプロミス。

$q

・プロミスは、Chorome 32、Firefox 29 の組込み関数として実装されているが、AngularJS では、$q と呼ばれるサービスが用意されている

プロミスの利用

・プロミスでは、deferred と呼ばれるインスタンスと、promise オブジェクトを一緒に利用出来る。

■ deferred 結果

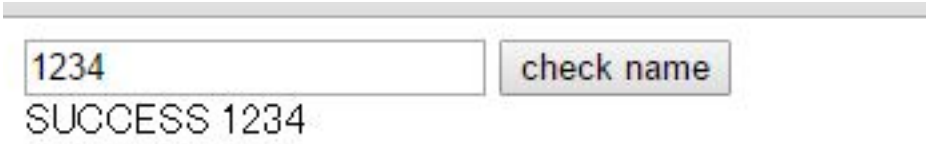
メソッド	結果
resolve	結果が成功の場合呼び出す
reject	結果が失敗の場合呼び出す
notify	resolve か reject が呼ばれるまでに何度でも呼び出すことが可能

■結果を待つメソッド

メソッド	引数	結果￥
then	successCallback,errorCallback,notifyCallback	成功失敗関係なく deferred で呼び出すメソッドの callback を指定
catch	errorCallback	失敗 (reject) を呼び出された場合のコールバックを指定
finally	callback	成功失敗関係なく最後に呼ばれるコールバックを指定

then はプロミスを返すのでメソッドチェーンが可能

■例




```
<!DOCTYPE html>
<html ng-app="app">
<head>
<script type="text/javascript" src="/angular/angular.min.js"></script>
<script type="text/javascript">
var app = angular.module('app',[]);
function checkNameLen(name){
    return name.length < 5;
}
app.controller('PromiseCtrl',['$scope','$q','$timeout',function($scope, $q, $timeout ){
    function checkAsync(name){
        var deferred = $q.defer();
        $timeout(function(){
            deferred.notify('Async Notify');
            if (checkNameLen(name)){
                deferred.resolve(name);
            } else {
```

```
                deferred.reject(name);
            }
        },500);
        return deferred.promise;
    }
    $scope.checkName = function(){
        checkAsync($scope.user_name)
            .then(
                function(msg){
                    $scope.msg = "SUCCESS " + msg;
                },
                function(msg){
                    $scope.msg = "FAIL " + msg;
                },
                function(msg){
                    $scope.msg = msg;
                });
    }
}]);
</script>
<title></title>
</head>
<body>
    <div ng-controller="PromiseCtrl">
        <input type="text" ng-model="user_name">
        <input type="button" value="check name" ng-click="checkName()">
        <div>{{msg}}</div>
    </div>
</body>
</html>
```

jqLite

・DOM を操作する API として、jQuery 互換の jqLite が用意されている
・angular.element で jqLite オブジェクトを取得できる
・https://docs.angularjs.org/api/ng/function/angular.element

■ jQuery の利用

・jqLite は、jQuery と比べると機能が大きく制限されている
・AngularJS を読み込む前に jQuery を読み込むと、angular.element で取得できるオブジェクトが jQuery のものに変更される (jQuery1.x 系のみ対応)

$http

・https://docs.angularjs.org/api/ng/service/$http

```
$http({
    method: 'POST',
    url: '/login',
    headers: {
        'X-CSRFToken': getCookie('csrftoken')
    },
    data: user,
}).success(function(user){
    alert(user);
});
```

$resource

・http://js.studio-kingdom.com/angularjs/ngresource_service/$resource
・angular-resource.js の読み込み、ngResource の依存設定が必要

angular-resource.js

```
<script type="text/javascript" src="js/angular-resource.js"></script>
```

ngResource

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

Tips

■ Django の CSRF トークンを送信するように config で設定する

- https://docs.angularjs.org/api/ng/provider/$httpProvider
- http://django-docs-ja.readthedocs.org/en/latest/ref/contrib/csrf.html

```
app.config(['$httpProvider', function ($httpProvider) {
    $httpProvider.defaults.xsrfHeaderName = 'X-CSRFToken';
    $httpProvider.defaults.xsrfCookieName = 'csrftoken';
}]);
```