Sommaire

[1. Apprentissage 2](#_Toc406498096)

[2. Structure d’une application AngularJS 2](#_Toc406498097)

[ng-app 2](#_Toc406498098)

[ng-controller 2](#_Toc406498099)

[{{}} 2](#_Toc406498100)

[ng-model 2](#_Toc406498101)

[$scope 3](#_Toc406498102)

[a) Example structure de code 3](#_Toc406498103)

[3. Les directives 3](#_Toc406498104)

[4. Les Services 4](#_Toc406498105)

[5. Les Factories ( !important) 5](#_Toc406498106)

[6. Les Filtres 5](#_Toc406498107)

[7. Binding de données à deux sens 6](#_Toc406498108)

[8. Appels XHR/Ajax/$http et binding JSON ( !important) 6](#_Toc406498109)

[9. Data-binding déclaratif ( !important) 7](#_Toc406498110)

[10. Liste des directives ( !important) 8](#_Toc406498111)

[11. Les fonctions de *scope* ( !important) 8](#_Toc406498112)

[12. Méthodes de DOM déclaratives 8](#_Toc406498113)

[13. Les expressions 9](#_Toc406498114)

[14. Routage et vues dynamiques ( !important) 9](#_Toc406498115)

[15. Données statiques globales ( !important) 10](#_Toc406498116)

[16. Minification (réduction du code) ( !important) 10](#_Toc406498117)

[17. Différences avec MVC et MVVM 11](#_Toc406498118)

[18. HTML5 Web Components 11](#_Toc406498119)

[19. Commentaires de scope 12](#_Toc406498120)

[20. Debugger AngularJS 12](#_Toc406498121)

[21. Test 12](#_Toc406498122)

# Apprentissage

<http://www.tinci.fr/blog/apprendre-angular-en-un-jour-le-guide-ultime/> \*\*\*\*\*\*\*

<http://www.angular-js.fr/angularjs-par-ou-je-commence/>

<http://blog.kaliop.com/blog/2014/01/13/angularjs-presentation/>

<https://egghead.io/lessons/angularjs-hijacking-existing-html-attributes-with-angular-directives>

<https://egghead.io/technologies/angularjs?order=ASC>

# Structure d’une application AngularJS

### ng-app

La **directive** ng-app permet de dire à AngularJs qu’il doit être actif sur cette section de la page. Dans notre cas, il s’agit de toute la page puisqu’elle est située sur la balise <html>, mais on pourrait très bien la mettre sur un <div> par exemple.

### ng-controller

Cette section de la page est gérée par le **contrôleur MainCtrl** (notez la majuscule). Les variables et fonctions déclarées dans le **scope** de ce contrôleur sont accessibles dans cette zone du html, et pas en dehors.

### {{}}

Tout ce qui est entre doubles accolades sera **interprété**. Les variables et fonctions du contrôleur sont utilisables ici. Si les données changent dans le contrôleur, les changements seront répercutés sur la vue. C’est du **one-way data binding**. Dans l’exemple ci-dessus, {{title}} affiche simplement le contenu de $scope.title.

### ng-model

C’est une des « killer features » d’AngularJs, c’est ce que l’on appelle du **two-way data binding**. Si les données sont mises à jour dans le contrôleur, les changements seront répercutés dans la vue, et si les données sont mises à jour dans la vue, les changements seront répercutés dans le contrôleur! Dans l’exemple ci-dessus, l’utilisateur tape du texte dans l’input, ce qui met à jour la variable $scope.name du contrôleur, et le changement est instantanement répércuté côté vue dans {{name}}.

### $scope

Le scope est ce qui fait le lien entre le contrôleur et la vue. Techniquement c’est un objet javascript, et les propriétés qu’on lui ajoute (variables et fonctions) sont accessibles dans la vue, elles sont en quelque sorte publiques. Mais il est également possible de créer des variables et des fonctions privées (pas accessibles dans la vue).

## Example structure de code

JS :

var MainCtrl = function ($scope) {

    $scope.title = "Ma première page"

    $scope.name = "";

};

Html :

<!doctype html>

<html lang="fr" ng-app>

<head>

<meta charset="UTF-8">

<title>Ma première page</title>

</head>

<body>

<div ng-controller="MainCtrl">

    <h1>{{title}}</h1>

    <input type="text" ng-model="name" placeholder="Votre nom">

    <p>Bonjour {{name}}</p>

</div>

<script src="<https://ajax.googleapis.com/ajax/libs/angularjs/1.0.8/angular.min.js>"></script>

<script src="app.js"></script>

# Les directives

Une directive ([lisez mon article sur les directives issues de scripts/plugins existants [en]](http://toddmotto.com/creating-an-angularjs-directive-from-one-of-your-existing-plugins-scripts)), dans sa forme la plus simple, est un petit morceau de template HTML, utilisé de préférence à plusieurs endroits de l'application. C'est un moyen facile d'injecter sans effort du DOM dans votre application ou d'effectuer des interactions particulières avec le DOM. Les directives ne sont pas simples pour autant et la courbe d'apprentissage pour les maitriser est assez importante. Ce qui suit devrait tout de même vous donner un bon point de départ.

À quoi servent donc les directives ? Beaucoup de choses dont la création de composants DOM (onglets ou éléments de navigation) - tout dépend de l'usage que votre application fait de l'interface utilisateur. Si vous avez, par exemple, joué un peu avec ng-show ou ng-hide, ce sont des directives (qui n'injectent pas de DOM).

<!-- 1: déclaration en tant qu'attribut -->

<a custom-button>Click me</a>

<!-- 2: en tant que nouvel élément -->

<custom-button>Click me</custom-button>

<!-- 3: en tant que classe (pour être compatible avec les vieux IE) -->

<a class="custom-button">Click me</a>

<!-- 4: en tant que commentaire (peu adapté à cette démo) -->

<!-- directive: custom-button -->

myApp.directive('customButton', function () {

return {

restrict: 'A', // method d’accès (A:attribute, E:element, C:classe, M commentaire)

replace: true, // balisage remplacé ou non

transclude: true, // dom original copié dans la directive ?

template: '<a href="" class="myawesomebutton" ng-transclude>' +

'<i class="icon-ok-sign"></i>' +

'</a>',

link: function (scope, element, attrs) {

// DOM manipulation/events here!

}

};

});

# Les Services

Les services sont souvent une notion un peu floue. D'après mes lectures et ma propre expérience, les services sont plus un design pattern de style qu'un réel apport de fonctionnalité. J'ai lu le code source d'Angular et a priori ils sont très proches des factories. Ils passent par le même compilateur et semble avoir de nombreuses fonctionnalités en commun. Il semble que les services soient préférables pour singleton et les factories pour les fonctions plus complexes comme les Object Literals ou d'autres cas plus compliqués.

myApp.service('Math', function () {

this.multiply = function (x, y) {

return x \* y;

};

});

myApp.controller('MainCtrl', ['$scope', function ($scope) {

var a = 12;

var b = 24;

// outputs 288

var result = Math.multiply(a, b);

}]);

# Les Factories ( !important)

Passer des services aux factories devrait être assez simple, on pourrait créer un Object Literal dans une factory ou simplement fournir des méthodes plus avancées :

// Plusieurs méthodes

myApp.factory('Server', function () {

return {

get: function(url) {

return $http.get(url);

},

post: function(url) {

return $http.post(url);

},

};

});

myApp.controller('MainCtrl', ['$scope', 'Server', function ($scope, Server) {

var jsonGet = 'http://myserver/getURL';

var jsonPost = 'http://myserver/postURL';

Server.get(jsonGet);

Server.post(jsonPost);

}]);

# Les Filtres

myApp.filter('reverse', function () {

return function (input, uppercase) {

var out = '';

for (var i = 0; i < input.length; i++) {

out = input.charAt(i) + out;

}

if (uppercase) {

out = out.toUpperCase();

}

return out;

}

});

<div ng-app="myApp">

<div ng-controller="MainCtrl">

<p>Sans filtre: {{ greeting }}</p>

<p>Reverse: {{ greeting | reverse }}</p>

</div>

</div>

// Pour les filtres dans le controller

<li ng-repeat="number in numbers | filter:greaterThanNum">

{{ number }}

</li>

# Binding de données à deux sens

La meilleure façon de le décrire est sous la forme d'un cercle de données synchronisées : si le modèle est mis à jour, la vue est mise à jour automatiquement ; si la vue est mise à jour, le modèle est automatiquement mis à jour. Cela veut dire que sans rien faire, la donnée est synchronisée. Si je bind un ng-model à un <input> et commence à taper dans ce dernier, cela crée un modèle (ou met à jour un modèle existant) en même temps.

<div ng-app="myApp">

<div ng-controller="MainCtrl">

<input type="text" ng-model="myModel" placeholder="Start typing..." />

<p>My model data: {{ myModel }}</p>

</div>

</div>

myApp.controller('MainCtrl', ['$scope', function ($scope) {

// On capture la donnée du modèle

// et/ou on l'initialise avec une chaîne de caractères vide

$scope.myModel = '';

}]);

# Appels XHR/Ajax/$http et binding JSON ( !important)

En développement local, vous utilisez probablement Java, ASP .NET, PHP ou une autre techno pour faire tourner l'application. Que vous contactiez une bonne de données locale ou que vous utilisiez ce serveur comme une API pour communiquer avec une autre ressource, la mise en place est globalement la même.

C'est ici que 'dollar http’ entre en scène. C'est dorénavant votre meilleur ami. La méthode $http d'Angular est un wrapper bien pratique pour accéder aux données du serveur et est d'une utilisation très simple. Voici un petit exemple pour une requête GET qui, comme vous l'aurez deviné, récupère des données depuis le serveur. Sa syntaxe est très proche de celle de jQuery, la transition est donc aisée :

myApp.controller('UserCtrl', ['$scope', '$http', function ($scope, $http) {

// Crée un Object user

$scope.user = {};

// Initialise le modèle avec une chaîne vide

$scope.user.username = '';

// Nous voulons effectuer la requête

// et obtenir le nom de l'utilisateur

$http({

method: 'GET',

url: '//localhost:9000/someUrlForGettingUsername'

})

.success(function (data, status, headers, config) {

// Ici nous assignons cet utilisateur à

// notre modèle existant !

$scope.user.username = data.user.name;

})

.error(function (data, status, headers, config) {

// Une erreur est survenue

});

}]);

<div ng-controller="UserCtrl">

<p>{{ user.username }}</p>

</div>

# Data-binding déclaratif ( !important)

C'est là que nous allons utiliser le binding déclaratif pour indiquer ce que va faire l'application - créer notre premier morceau de HTML dynamique. Nous allons utiliser la directive ng-repeat d'Angular qui va parcourir les données et générer un résultat sans que nous ayons à faire de callbacks ou de changements d'état, c'est gratuit :

myApp.controller('EmailsCtrl', ['$scope', function ($scope) {

// Crée un Object emails

$scope.emails = {};

// Nous écrivons ici en dur les données normalement

// reçues du serveur

$scope.emails.messages = [{

"from": "Steve Jobs",

"subject": "I think I'm holding my phone wrong :/",

"sent": "2013-10-01T08:05:59Z"

},{

"from": "Ellie Goulding",

"subject": "I've got Starry Eyes, lulz",

"sent": "2013-09-21T19:45:00Z"

},{

"from": "Michael Stipe",

"subject": "Everybody hurts, sometimes.",

"sent": "2013-09-12T11:38:30Z"

},{

"from": "Jeremy Clarkson",

"subject": "Think I've found the best car... In the world",

"sent": "2013-09-03T13:15:11Z"

}];

}]);

<ul>

<li ng-repeat="message in emails.messages">

<p>From: {{ message.from }}</p>

<p>Subject: {{ message.subject }}</p>

<p>{{ message.sent | date:'MMM d, y h:mm:ss a' }}</p>

</li>

</ul>

# Liste des directives ( !important)

https://docs.angularjs.org/api/ng/directive/

# Les fonctions de *scope* ( !important)

Après le binding de directives, les fonctions de scope sont la suite logique dans la création d'une application. Voici une fonction assez simple nous permettant de supprimer un email dans notre collection :

myApp.controller('MainCtrl', ['$scope', function ($scope) {

$scope.deleteEmail = function (index) {

$scope.emails.messages.splice(index, 1)

};

}]);

<a ng-click="deleteEmail($index)">Delete email</a>

# Méthodes de DOM déclaratives

Nous allons maintenant voir les méthodes de DOM. Elles sont comme les directives et permettent d'ajouter au DOM des fonctionnalités pour lesquelles vous auriez, en temps normal, beaucoup de code à écrire. Un très bon exemple serait une navigation déroulante. En utilisant ng-show et ng-click nous pouvons mettre cela en place très simplement :

<a href="" ng-click="toggle = !toggle">Afficher menu</a>

<ul ng-show="toggle">

<li>Lien 1</li>

<li>Lien 2</li>

<li>Lien 3</li>

</ul>

# Les expressions

<p>{{ data.length > 0 && 'My data' || 'No data' }}</p>

Ce morceau de DOM va être automatiquement mis à jour, sans besoin de callback, au fur et à mesure que votre application récupère les données. Il vous indiquera si des données sont présentes ou non. Il y a de très nombreux cas d'usage et Angular le gère automatiquement grâce au binding de données à deux sens.

# Routage et vues dynamiques ( !important)

La philosophie des applications côté client (et des applications web en général) est assez simple : Vous avez un header, un footer, une sidebar et le contenu au milieu injecte du contenu par magie en fonction de l'URL.

Angular facilite beaucoup la mise en place d'un tel mécanisme, ce qu'on appellerait les vues dynamiques. Les vues sont injectées dynamiquement au travers de $routeProvider, en fonction de l'URL.

myApp.config(['$routeProvider', function ($routeProvider) {

/\*\*

\* $routeProvider

\*/

$routeProvider

.when('/', {

templateUrl: 'views/main.html'

})

.when('/emails', {

templateUrl: 'views/emails.html'

})

.otherwise({

redirectTo: '/'

});

}]);

$routeProvider propose d'autres fonctionnalités qui valent la peine de fouiller un peu, mais avec ceci vous avez déjà de quoi faire des choses intéressantes. Il y a, par exemple, les intercepteurs $http qui émettent des événements lorsqu'une requête Ajax est en cours, ce qui permettrait d'afficher un spinner pendant que les données sont récupérées.

# Données statiques globales ( !important)

* Permet de transmettre les données en clair dans la page
* En gros, executer des requetes coté serveur et écrire le résultat dans un JSON de modèles de données

Gmail récupère une bonne partie de ses données en les écrivant en JSON dans la page (inspectez les sources de la page). Si vous voulez accéder instantanément à vos données dès le chargement de l'application et accélérer son lancement, cette technique est très pratique.

Lorsque je développe nos applications, des balises issues de Java sont insérées dans le DOM et, une fois l'application chargée, les données viennent du serveur. Je parle ici de Java mais vous pouvez utiliser n'importe quel langage côté serveur. Voici comment ajouter le JSON dans votre page et comment le charger ensuite depuis un contrôleur :

<!-- dans index.html (en bas de page) -->

<script>

window.globalData = {};

globalData.emails = <baliseJavaINTERPRETEEPARLESERVEURPourGenererLesMessages>;

</script>

Ma balise Java va être remplacée par les bonnes données au moment de l'affichage et Angular va affiche instantanément les emails. Récupérez simplement les données dans un contrôleur :

myApp.controller('EmailsCtrl', ['$scope', function ($scope) {

$scope.emails = {};

// Assigner les données initiales !

$scope.emails.messages = globalData.emails;

}]);

# Minification (réduction du code) ( !important)

Minifier le code AngularJS est simple, vous devez simplement spécifier les injections de dépendances dans un tableau, avant la fonction :

myApp.controller('MainCtrl',

['$scope', 'Dependency', 'Service', 'Factory',

function ($scope, Dependency, Service, Factory) {

// code

}]);

Une fois minifié :

myApp.controller('MainCtrl',

['$scope', 'Dependency', 'Service', 'Factory',

function (a,b,c,d) {

// a = $scope

// b = Dependency

// c = Service

// d = Factory

// $scope alias usage

a.someFunction = function () {...};

}]);

Rappelez-vous bien de garder la liste des injections dans le même ordre que les paramètres de la fonction, vous risqueriez un bon mal de crâne dans le cas contraire.

# Différences avec MVC et MVVM

Les différences avec MVC/MVVM dont AngularJS est fier :

* *MVC*: parle avec un contrôleur, Modèle-Vue-Contrôleur
* *MVVM*: encapsule une *binding* de données déclaratif qui, techniquement, se parle à lui-même. Modèle-Vue-Vue-Modèle. Le modèle parle à la vue et la vue peut parler au modèle. Le binding de données à deux sens d'Angular permet de garder cette synchronisation sans rien avoir à écrire. Cela vous permet d'écrire de la logique sans contrôleur.

Un exemple rapide, vous pouvez utiliser ng-repeat sans contrôleur en fournissant directement les données :

<li ng-repeat="number in [1,2,3,4,5,6,7,8,9]">

{{ number }}

</li>

# HTML5 Web Components

<myCustomElement></myCustomElement>

Cela permet d'aligner le web sur le futur de HTML5 qui introduit les web components et l'élément <template>. Angular nous permet de les utiliser dès aujourd'hui. Les web components mélangent éléments personnalisés et injection dynamique de JavaScript pour peupler dynamiquement la vue, c'est très excitant et déjà possible avec Angular ! Ils ont un cran d'avance et permettent de s'assurer que ce qui arrive fonctionne - chapeau bas.

# Commentaires de scope

<!-- scope: MainCtrl -->

<div class="content" ng-controller="MainCtrl">

</div>

<!-- /scope: MainCtrl -->

# Debugger AngularJS

https://chrome.google.com/webstore/detail/angularjs-batarang/ighdmehidhipcmcojjgiloacoafjmpfk

# Test

<http://jsfiddle.net/toddmotto/mN7QB/light/>