Directives

Les directives Angular JS permettent de modifier le DOM et son comportement. Elles peuvent être invoquées sur différents éléments (éléments - E, attributs - A, classe Css - C, commentaire - M) :

```
<my-dir></my-dir>
<span my-dir="exp"></span>
<!-- directive: my-dir exp -->
<span class="my-dir: exp;"></span>
```

Directives fréquemment utilisées

Directive	Portée	Exemple	Rôle
ng-app	A	`<html ng-app="myModule">...`	Désigne le root d'une application (module), se place généralement sur les balises html ou body de la page

ng-controller	A	`<div ng-controller="MyController">`
		`...`
		`{{uneExpressionDefinieDansLeScope}}`
		`</div>`
		
		`<div ng-controller="MyController as myCtrl">`
		`...`
		`{{myCtrl.uneVariable}}`
		`</div>`
			Associe une classe contrôleur dans une application (module) à la partie de la vue désignée et défini une nouvelle portée $scope

ng-bind	A C	`<span ng-bind="name"></span>`	Associe (bind) la partie HTML spécifiée à une expression
ng-model	A	`<input type="text" ng-model="name">`	Associe (bind) l'élément HTML spécifié (input, select, textarea) à une expression
ng-value	A	`<input type="radio" ng-value="name">`	Associe (bind) la valeur de l'élément HTML spécifié (input type="radio", option) à une expression

Directive	Portée	Exemple	Rôle
ng-include	E A C	`<div ng-include="expressionUrl">...</div>`	inclut à l'endroit spécifié le fichier html correspondant à l'expression

Directive	Portée	Exemple	Rôle

ng-switch	A	`<div ng-switch on="expression">`
		`  <div ng-switch-when="valeur1">...</div>`
		`  <div ng-switch-when="valeur2">...</div>`
		`  <div ng-switch-default>...</div>`
		`</div>`
			inclut **selon** la valeur de l'expression les éléments HTML aux endroits spécifiés

ng-repeat	A	`<div ng-repeat="element in elements">...</div>`	Répète une séquence HTML pour chacun des éléments d'un tableau ou d'une collection ng-init peut éventuellement servir à initialiser la collection

Directive	Portée	Exemple	Rôle
ng-click	A	`<div ng-click="jsExpression">...</div>`	Associe un comportement au click sur l'élément $event correspond à l'objet event généré
ng-show	A	`<div ng-show="booleanJsExpression">...</div>`	Affiche ou masque l'élément associé
ng-hide	A	`<div ng-hide="booleanJsExpression">...</div>`	Masque ou affiche l'élément associé

ng-click, ng-show,ng-hide, ng-class, ng-submit

Création de directives

Quels intérêts ?

- Etendre le HTML et lui attribuer une logique métier
- Factoriser le code, permettre la réutilisation

Directive simple & template

```
var app = angular.module('testDirectivesApp', [])
.controller('Controller', ['$scope', function($scope) {
  $scope.client = {
    nom: 'SMITH',
    prenom: 'John'
  };
}]);

app.directive('myClient', function() {
  return {
    template: 'Nom : {{client.nom}} Prénom : {{client.prenom}}'
  };
});
```

Utilisations possibles :

```
<html data-ng-app="testDirectivesApp" data-ng-controller="Controller">
...
    <my-client></my-client>
    <div my-client></div>
```

Résultat :

Nom : SMITH Prénom : John
Nom : SMITH Prénom : John

templateUrl

Excepté pour les petits templates, il est préférable d'utiliser un fichier template séparé en affectant la propriété **templateUrl** (notamment pour éviter les problèmes de quote/guillemets) :

```
app.directive('myClient', function() {
  return {
    templateUrl: 'my-client.html'
  };
});
```

```
Nom : {{client.nom}} Prénom : {{client.prenom}}
```

La propriété templateUrl peut-être définie par une fonction :

```
app.directive('myClient', function() {
  return {
    templateUrl: function(elem, attr){
      return 'client-'+attr.type+'.html';
    }
  };
});
```

Templates :

```
Nom : {{client.nom}} Prénom : {{client.prenom}}
```

```
Adresse: {{client.adresse}}<br>
CP: {{client.cp}} {{client.ville}}
```

Utilisation :

```
<html data-ng-app="testDirectivesApp" data-ng-controller="Controller">
...
    <div myclient type="nom"></div>
    <div my-client type="adresse"></div>
```

restrict

La propriété **restrict** permet de définir le type de directive à créer ; restrict peut prendre les valeurs suivantes :

- éléments - **E**
- attributs - **A**
- classe Css - **C**
- commentaire - **M**
- ou une combinaison de ces valeurs : **EA** par exemple, pour élément et attribut

Si la directive est définie sur 1 élément :

```
app.directive('myClient', function() {
  return {
    restrict: 'E',
    templateUrl: function(elem, attr){
      return 'client-'+attr.type+'.html';
    }
  };
});
```

Seule la première ligne sera compilée par Angular, la seconde sera ignorée :

```
<html data-ng-app="testDirectivesApp" data-ng-controller="Controller">
...
    <my-client type="nom"></my-client>
    <div my-client type="adresse"></div>
```

scope

Le scope défini la portée d'une directive. Sans précision, le scope d'une directive est le scope de son contrôleur parent. Ce qui explique que notre directive ait pu accéder à la variable **client** de **Controller**.

Isolation

La directive précédemment créée a un défaut : elle dépend du contrôleur dans laquelle elle a été définie, et si nous souhaitons afficher un autre client, il faut définir un autre contrôleur...
La définition de la variable scope de la directive permet de résoudre ce problème :

scope:{} Défini à {}, la directive a un scope qui lui est propre, différent de celui du controller auquel elle appartient :

```
app.directive('myClient', function() {
  return {
```

```
    restrict: 'AE',
    scope:{},
    templateUrl: function(elem, attr){
      return 'client-'+attr.type+'.html';
    }
  };
});
```

Avec ce scope isolé, la directive ne peut plus accéder à la variable client définie dans le scope du controller : tester la page tests.html

Nous allons ajouter une propriété permettant d'initialiser la directive, et lui affecter la personne à afficher :

```
<html data-ng-app="testDirectivesApp" data-ng-controller="Controller">
...
    <my-client type="nom" personne="client"></my-client>
    <my-client type="adresse" personne="client"></my-client>
```

```
app.directive('myClient', function() {
    return{
        restrict:'EA',
        scope:{client:"=personne"},
        templateUrl: function(elem, attr){
            return 'client-'+attr.type+'.html';
        }
    }
});
```

L'attribut **personne** “bind” vers la variable **client** utilisée dans le scope de la directive.

Lorsqu'un attribut a le même nom que la variable du scope auquel il est associé, on peut utiliser la notation :

```
scope : {attributeName: "="}
```

Pour résumer (rapidement...) :

Valeurs de scope	**Résultats**
false ou non définie	pas de scope
{}	scope isolé
{attributeName:“=”}	scope isolé, avec passage d'un attribut de type **expression valeur** du même nom
{attributeNameInDirectiveScope:“=attributeName”}	scope isolé, avec passage d'un attribut de type **expression valeur** de nom différent
{attributeName:“@”}	scope isolé, avec passage d'un attribut de type **texte** du même nom
{attributeNameInDirectiveScope:“@attributeName”}	scope isolé, avec passage d'un attribut de type **texte** de nom différent
{attributeName:“&”}	scope isolé, avec passage d'un attribut de type **expression action** du même nom

Valeurs de scope	**Résultats**
{attributeNameInDirectiveScope:“&attributeName”}	scope isolé, avec passage d'un attribut de type **expression action** de nom différent
true	scope non isolé

Exemple de passage d'une expression action (ng-click) :

```
    <my-client type="nom" personne="client" ng-click="client.adresse=''"></my-
client>
    <div my-client type="adresse" personne="client"></div>
```

Sur click du nom, l'adresse doit être vidée.

En l'état actuel du code, la directive ng-click n'est pas opérationnelle, il faut la passer à notre directive :

```
app.directive('myClient', function() {
    return{
        restrict:'EA',
        scope:{client:"=personne",onClick:"&ngClick"},
        templateUrl: function(elem, attr){
            return 'client-'+attr.type+'.html';
        }
    }
});
```

et la déclencher à nouveau dans le template :

```
<p ng-click="onClick">Nom : {{client.nom}} Prénom : {{client.prenom}}</p>
```

Link/compile

Lorsqu'il analyse une directive appelée dans une page et la traduit en un ensemble d'éléments DOM, AngularJS compile la directive en faisant appel aux fonctions suivantes, dans un ordre déterminé :

1. compile
2. controller
3. pre-link
4. post-link

Si une directive doit modifier ou créer le DOM d'un template défini dans une directive, elle doit donc définir certaines de ces fonctions.

Exemple

```
app.directive('soldeDir', function() {
    return {
        restrict : 'EA',
```

```
        scope:{montant:"="},
        compile : function(tElem, tAttrs) {
            if (tAttrs.bold == "true")
                tElem.css("font-weight","bold");
            else
                tElem.css("font-weight","normal");
            var linkFunction = function($scope, tElem, tAttrs) {
                var update=function(){
                    if($scope.montant<0){
                        tElem.html("Solde négatif : " + $scope.montant);
                        tElem.css("color", "red");
                    }else{
                        tElem.html($scope.montant);
                        tElem.css("color", "green");
                    }
                }
                update();
            }
            return linkFunction;
        }
    }
});
```

```
    <div ng-init="valeur=-5">
    <input type="text" ng-model="valeur">
    <solde-dir bold="true" montant="valeur"></solde-dir>
```