Controladores y templates

Anxo Fole
afole@plainconcepts.com

Controladores

Controladores

- Aumentan el estado inicial y el comportamiento del $scope

- No usar para:
  - Manipular el DOM -> usar directivas
  - Formatear inputs -> usar formularios y ng-model
  - Filtrar el output -> usar filtros
  - Compartir código entre controladores -> usar servicios

Controller

- Se crea una nueva instancia en cada solicitud
  - Son funciones constructoras

```
<div ng-controller="UserController">
    <p>{{name}}</p>
    <div ng-if="someCondition">
        <input ng-model="name">
    </div>
</div>
```

Creación de controladores

- Usar IIFE’s

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

    var controllerId = 'MyController';

    angular.module('app').controller(controllerId,
        ['$scope', MyController]);

    function MyController($scope) {
        $scope.title = 'MyController';
        $scope.activate = activate;

        function activate() { }
    }
})();
```

Controladores: $scope vs Controller As

$scope

```
(function () {
    'use strict';
    angular
        .module('app')
        .controller('controller2', controller2);

    controller2.$inject = ['$scope'];

    function controller2($scope) {
        $scope.title = 'controller2';
}
})();
```

Controller as

```
(function () {
    'use strict';
    angular
        .module('app')
        .controller('controller1', controller1);

    controller1.$inject = ['$location'];

    function controller1() {
        var vm = this;
        vm.title = 'controller1'; }
})();
```

Usar Controller As

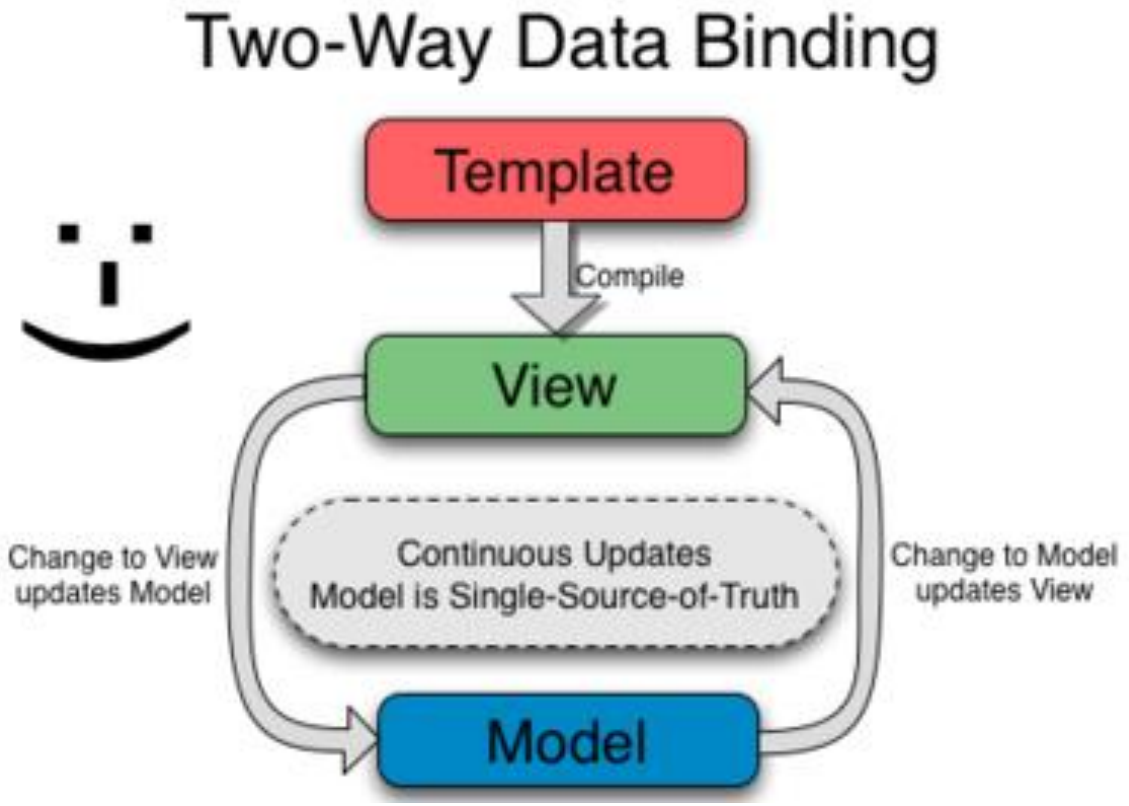
- Buenas prácticas
- Más legible
- Es más claro a que hacemos referencia cuado usamos $scopes anidados.
- Usamos “dot notation”. (e.g. customer.name instead of name),
  - Evitamos problemas con el binding.
  - Más contextual.
- Evitamos inyectar el $scope si no es necesario.

scope

Características

- Relaciona el controlador y las vistas exponiendo el modelo y el comportamiento expuesto por el controlador a la vista
  - Controlador y vista conocen el $scope, pero no se conocen entre ellos
- Es el contexto de ejecución de las expresiones
- Creados en estructura jerárquica desde el **$rootScope** (ng-app)
- Pueden observar ($watch) expresiones y propagar eventos

Data Binding



Scope

```
var User = function () {
    this.fullName = 'Miško Hevery';
    this.username = 'mhevery';
}
$scope = {};
$scope.user = new User();

<h1>{{user.fullName}}</h1>
<h1>{{user.username}}</h1>
```

Scopes anidados

Scope – Herencia de prototipo

```
function inherit(parent) {
    function C() { }
    C.prototype = parent;
    return new C();
}

parent = {};
child = inherit(parent);
expect(child.__proto__).toBe(parent);

parent.name = 'parent';
expect(child.name).toEqual('parent');

child.name = 'child';
expect(child.name).toEqual('child');
expect(parent.name).toEqual('parent');
```

La importancia del punto

Instancia compartida

```
scope.data = {message: 'Hola!'};


<div ng-app="">
    <input type="text" ng-model="data.message">
    <h1>{{ data.message }}</h1>
    <div ng-controller="FirstCtrl">
        <input type="text" ng-
model="data.message">
        <h1>{{ data.message }}</h1>
    </div>
    <div ng-controller="SecondCtrl">
        <input type="text" ng-
model="data.message">
        <h1>{{ data.message }}</h1>
    </div>
</div>
```

Variables diferentes

```
scope.message = 'Hola!';


<div ng-app="">
    <input type="text" ng-model="message">
    <h1>{{ message }}</h1>
    <div ng-controller="FirstCtrl">
        <input type="text" ng-model="message">
        <h1>{{ message }}</h1>
    </div>
    <div ng-controller="SecondCtrl">
        <input type="text" ng-model="message">
        <h1>{{ message }}</h1>
    </div>
</div>
```

$watch

```
scope.name = 'Miško Hevery';

scope.$watch('name', function (newValue, oldValue) {
    element.text(newValue);
});
```

Funcionamento del binding bidireccional

- Durante la fase de linkado de las templates, las directivas establecen $watch sobre el $scope.

- El $watch permite a la directiva ser notificada cuando hay cambios en el modelo
  - Lo que permite a la directiva renderizar el valor actualizado en la vista

$apply

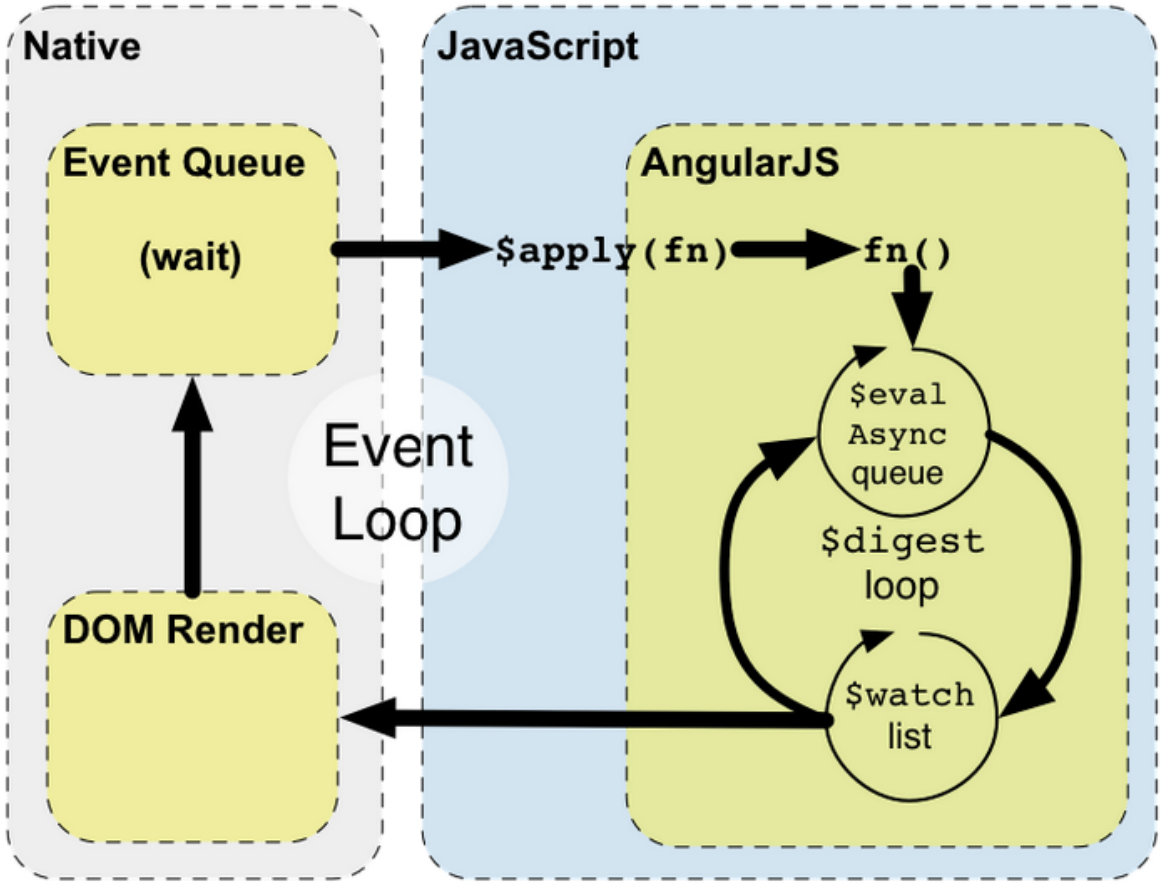
```
function Ctrl($scope) {
    $scope.message = "Waiting 2000ms for update";

    setTimeout(function () {
        $scope.message = "Timeout called!";
        // AngularJS unaware of update to $scope
    }, 2000);
}
```

```
function Ctrl($scope) {
    $scope.message = "Waiting 2000ms for update";

    setTimeout(function () {
        $scope.$apply(function () {
            $scope.message = "Timeout called!";
        });
    }, 2000);
}
```

Digest


Native
JavaScript
Event Queue
(wait)
AngularJS
$apply(fn)
fn()
$eval
Async
queue
Event
Loop
$digest
loop
DOM Render
$watch
list

Expresiones

Contexto de ejecución

- No tienen acceso a variables globales
- Se evalúan siempre contra las variables definidas en el $scope
- Servicios de Angular proven acceso a variables globales
  - $window
  - $document
  - $location

Características

- No lanza excepciones al acceder a propiedades sobre objetos “undefined”
  - Evalua a null o undefined
  - {{a.b.c}}

- One time binding -> {{::tc.user.name}}
  - Reduce el número de watches a evaluar en cada ciclo de dirty checking

Directivas

Incluidas en AngularJS

Directivas

- Componentes que encapsulan la interacción con el DOM

- Elemento, atributo, clase o comentario
- Normalizadas ***ngApp*** equivale a *ng-app, data-ng-app*, y otras…

- Angular ofrece un gran abanico de directivas OOTB
  - Hay un gran ecosistema de desarrollo de directivas de terceros

Directivas - Aplicación

Aplicación

- **ngApp**: autoregistra una aplicación de Angular
- **ngController**: agrega un controlador al elemento de la vista
- **ngInclude**: obtiene, compila e incluye un fragmento HTML externo
- **script**: carga el contenido del element en la cache de **templates**

```
<script type="text/ng-template" id="/tpl.html">
    Content of the template.
</script>
```

Directivas - Binding

Binding:

- **{{Expresiones}}**: expresiones evaluadas por Angular
- **ngCloak**: previene que se muestre la plantilla sin compilar
- **ngBind**: reemplaza el texto del element por el valor de una expresión
- **ngBindTemplate**: soporta varias expresiones simultáneas “{{a}}-{{b}}”
- **ngBindHtml**: require ngSanitize. Asigna el innerHtml de forma segura
- **ngNotBindable**: no compila el contenido del elemento

Directivas - Estado

- **ngClass, ngClassOdd, ngClassEven**: modifica las clases css que aplican a un elemento.
- **ngStyle**: modifica el estilo que aplica a un element, permite setear el estilo de un elemento condicionalmente.

- **ngDisabled**: vincula el atributo “disabled” con una expresión
- **ngSelected**: vincula el atributo “selected” con una expresión
- **ngOpen**: vincula el atributo “open” con una expression

Atributos booleanos: la especificación HTML no obliga a mantener el atributo cuando es false

- **ngShow**: muesta el element cuando es cierta una condición
- **ngHide**: oculta un element cuando es cierta una condición

Directivas - Origen

- **ngHref**: asigna el atributo “href” de un anchor con una expresión

NO <a href="http://www.gravatar.com/avatar/{{hash}}"/>

SI <a **ng-href**="http://www.gravatar.com/avatar/{{hash}}"/>

- **ngSrc**: asigna el atributo “src” de una imagen con una expresión

NO

SI

Directivas - Formularios

- **form:** ngFormController
- **ngForm:** formulario anidado
- **ngModel:** binding bidireccional
- **input:** ngModelController
- **textarea:** ngModelController
- **select:** Combos bindeados arrays de objetos
- **ngList:** Convierte un array de strings en una lista de separada por comas
- **ngValue:** bindea el valor (id) de botones de radio
- **ngChecked:** binding unidireccional de check

Directivas - Interacción

- ngChange
- ngClick
- ngDblclick
- ngMousedown
- ngMouseup
- ngMouseover
- ngMouseenter
- ngMouseleave
- ngMousemove
- ngKeydown
- ngKeyup
- ngKeypress
- ngSubmit (onsubmit)
- ngBlur
- ngCopy
- ngCut
- ngPaste

Directivas - Control

- **ngIf:** Añade element al DOM en base a una condición
- **ngRepeat:** Repite un elemento en base a los objetos que aparecen en un array. Importante usar un **track by** id cuando recuperamos objetos con claves primarias desde base de datos
- **ngSwitch:** Muestra de forma condicional un element. ng-switch on, ng-switch-when y ng-switch-default

Filtros

Incluidos en AngularJS

Filtros

- **currency:** formato de monedas
- **lowercase:** convierte a minúsculas
- **uppercase:** convierte a mayúsculas
- **number:** formato de número
- **json:** formatea un objeto como json
- **date:** format de fecha
- **filter:** fitrado de un array
- **limitTo:** crea un Nuevo array solo con el número de elementos especificdo
- **orderBy:** ordena un array por la expression o predicado especificado

Componentes

Incluidos en AngularJS

Componentes

- Un componente es una especia de directiva con una configuración más simple, más adecuada para una aplicación con estructura de componentes.

- Ventajas:
  - Configuración simple
  - Buenas prácticas
  - Optimizada para arquitecturas basadas en componentes
  - Más fácil de actualizar Angular

Componentes

- No usar componentes para:
  - Directivas con necesidades avanzadas de definición, como prioridad, multielemento.
  - Directivas que son de tipo atributo o clase CSS.

Componentes

```
angular.module('heroApp').component('heroDetail', {
templateUrl: 'heroDetail.html',
 bindings: {
        hero: '='
        }
});
```

```
 <span> Name: {{$ctrl.hero.name}} </span>
```

Arquitectura basada en componentes

- Un componente solo controla su vista y sus datos.
  - Isolate Scope, solo tiene acceso a sus datos.
  - Un componente nunca debería modificar datos del DOM que no están en su scope.
- Input
  - Solo deberíamos usar one-way binding (<) o strings (@). Nunca two-way (=).
    - Si pasamos una propiedad al componente, y esta cambia, el padre no se actualiza.
- Output
  - Se realizan con callbacks (& binding) al padre.
  - El padre decide que hacer con el.

Ciclo de vida de los componentes

- **$onInit():** Se llama en cada componente despues de que todos los controles de un element han sido construidos y sus bindings inicializados.
- **$onChanges(changesObj):** Se llama cuando algunos de los bindings (one-way) han sido actualizados. changesObj {currentValue, previousValue, isFirstChange}
- **$doCheck:** Se llama en cada ciclo de $digest, sirve para detector cambios que no detecta $onChanges. Deep Equality Check.
- **$onDestroy:** Se llama cuando se destrute el scope del componente.
- **$postLink:** Se llama despues de que los controles del element y sus hijos han sido linkados.

Componentes como templates de ruta

```
myMod.config(function($routeProvider) {
$routeProvider.when('/', {
template: '<home></home>’
}); });
```