****

A **JavaScript Framework**

**History**

AngularJS development was started by two developers, Misko Hevery and Adam Abrons at Brat Tech LLC in 2009. Later AngularJS version 1.0 released in 2012 and officially powered by Google.

**What AngularJS?**

It’s a **JavaScript based *open-source framework*** for Dynamic web Application. AngularJS extends HTML attributes with Directives and Data Binding to HTML with Expression. It’s well suited for creating SPAs (*Single Page Application*), but it can also be used for multipage based dynamic web applications.

**“Two Way Data Binding”** the most notable feature of the framework.

**$watch:**

An angular method used for **dirty checking**. Any variable or expression assigned in scope automatically sets up a $watchExpression in angular.

$watch is used to **detect the change in scope** variable. If it finds any change, then it updates the scope variable.

**$digest**:

It’s a loop or cycle. Inside that we have $watch list.

**Dirty checking:**

It’s a process which walks through the $watch list.

In details it compares the new value with old value of each watcher in the $watch list. If a value changes, it makes a record of it and continues to the next watcher.

**$apply:**

A method which internally invokes **$digest**.

Or a method to start dirty checking manually (by executing all $watches).

**$destroy:**

A method which removes a scope and all its children from dirty checking.

**Extension to HTML:**

1. ng-app
2. ng-model
3. ng-bind
4. ng-init
5. ng-repeat
6. ng-click
7. ng-change
8. ng-if
9. ng-show
10. ng-hide
11. ng-switch

**Scope:**

* A scope is an object that refers to the application model. It is used to specify the binding part between the HTML and Controller.
* It is accessible in both controller and views.
* Scopes can watch bindings and propagates events.
* Scopes in AngularJS support **Hierarchies or Inheritance**.
* In AngularJS the nested scopes are either **child scope** or **isolated scope**. The child scope is inheriting properties from the base/parent scope where as an isolated scope does not prototypically inherit from its parent scope.

Usage:

$scope.message = "Welcome to AngularJS World.";

**Child Scopes**:

* AngularJS supports **prototypical** inheritance.
* Each Angular Application may have **several child scopes**, but it has exactly **one** **root scope**.
* At the point a new scope is created, they include as the children of their parent scope.

**Isolated Scopes**:

* A new isolate scope is created for the directive element. The scope option is used to create an isolate scope.
* Isolate scope does not prototypically inherit from its parent scope.

**Scope Event Propagation**:

* It provides an effective way to exchange messages from different scopes at different hierarchy level.
* **$emit**: Propagates events **upwards** to scope hierarchy. Thereafter the event traverses upward towards the root scope and calls every registered listener along the way. If one of the user cancels the event, then emit will stop propagating.
* **$broadcast**: Propagates events **downwards** to every child scopes hierarchy. The event can’t be canceled.

**Expression:**

AngularJS Expressions are much like JavaScript expression and used to bind data in HTML. It executes expression and return the values. It can be used as **{{expression}}** or **ng-bind=”expression”.**

**Interpolation:**

* Interpolation markup with embedded expressions will provide data binding to text nodes and attribute values.
* The default symbol of expression is {{ and }}.(curly braces)
* To modify the symbol, we can set the *startSymbol()* and *endSymbol()* method.

Usage:

<a ng-href="img/{{username}}.jpg">Hello {{username}}</a>

Note:

It is preferable to use ng-bind instead of {{expression}} if a template is momentarily displayed by the browser in its raw state before angular complies/renders it.

**Module:**

* AngularJS supports modular approach. The module is basically a container which defines an application.
* *angular.module* function is used to create a module.

Usage:

angular.module("applicationModuleName", []);

**Controller:**

* AngularJS controllers are regular **JavaScript Objects** which **controls the data** in the applications.
* It is defined by a **JavaScript Constructor Function**.
* By using specific constructor function, the library will initiate a **new controller object** and a new **child scope** will be available to the controller as injectable parameter.
* The ng-controller directive is used to add a controller to module.

Usage:

app.controller("contrlName", function($scope) { });

**Filter:**

* **currency –** format a number to a currency format
* **date** – format a date to a specified format
* **filter** – select a subset of items from an array
* **Json** – format an object to a JSON string.
* **limitTo** – limits an array/string to a specified number of elements/characters.
* **lowercase** – format a string to lowercase.
* **number** – format a number to a string.
* **orderBy** – orders an array by an expression.
* **uppercase** – format a string to uppercase.

Usage:

module.service('filterName', function (){});

module.filter("filterName", function () {

return function(item, optParams) {

return item.toUpperCase();

};

});

**Directive:**

* AngularJS directives are extended HTML attributes with prefix ng.
* It has a set of built-in directives like ng-app, ng-init, ng-model etc as well as it does allow to create your own directives (custom directive).

Properties:

1. restrict:

‘A’- Only matches attribute name

‘E’- Only matches element name

‘C’- Only matches class name

‘M’- Only matches comment

These restrictions can all be combined as needed like ‘AEC’- matches either attribute, element or class name

1. templateURL:
2. scope:
3. link
4. require
5. transclude

4 Functions of Directive Life Cycle:

1. Compile: It allows the directive to manipulate the DOM before it is compiled.
2. Controller: It facilitates directive communication.
3. Pre-link: It allows for private $scope manipulation before the post-link process begins.
4. Post-link: It is the primary method of the directive.

**Component:**

Added in version 1.5.x later.

**Service:**

* A Service is a function or method in our module that takes a name and a function that defines the service.
* A Service is a constructor function.
* AngularJS has about 30 built-in services(refer <https://docs.angularjs.org/api/ng/service> ). Few of them are listed below

1. $location
2. $http
3. $timeout
4. $interval
5. $exceptionHandler

Usage:

module.service('serviceName', function (){});

When declaring serviceName as an injectable argument you will be provided with an instance of a function. In other words, ‘new FunctionPassedToService()’.

Custom Service creation:

app.service("mySampleService", function() {

this.joinStrings = function (str1, str2) {

return str1.concat(str2);

};

});

**Factory:**

* Factory allows us to add some logic before creating the object we require.
* It differs from service in a way where it allows us to pass a function which factory then invokes and returns the result.

Syntax:

module.factory('factoryName', function (){});

**Provider:**

* Providers are the only service you can pass into your .config() function.
* Factory and Service are all special type of providers.
* Providers only executed once. i.e. the moment when its injected to the controller.
* All providers in AngularJS (value, constant, services, factories) are singletons!

Usage:

module.provider('providerName', function (){});

**Data Binding**

There are 2 ways of data binding in AngularJS.

1. Classic Template System (***One Way Data Binding***):

* This combines the data in single direction (i.e. from Controller to View). The scope data gets propagated to view. If there any changes happened in the scope it will be immediately propagated to view.
* It Usually performed by **ng-bind** or evaluation expression.

1. Angular Templates (***Two Way Data Binding***):

* This combines the data in both the direction (i.e. Controller to View and Vice versa). This compilation basically produces a live view. Changes done in the view will immediately reflected in model, similarly any such changes in the model will be propagated in the view as well.
* It usually performed by **ng-model**.

**Why AngularJS?**

* Front-end framework provided by Google as an Open-source.
* Easy to understand since the library is written in JavaScript.
* Provides Two-way data-bindings.
* Well defined for SPAs & multipage dynamic web applications.
* Controls the entire application and show/hide everything in HTML DOM.
* Enhance the form and validation.
* Provides extended features to create CRUD Application.
* Provides Routing which allow us to handle the URL hash.