<http://tutlane.com/tutorial/angularjs/angularjs-data-bindings-one-way-two-way-with-examples>

Difference between ng-if and ng-show

**ngIf**

The ngIf directive **removes or recreates** a portion of the DOM tree based on an expression. If the expression assigned to ngIf evaluates to a false value then the element is removed from the DOM, otherwise a clone of the element is reinserted into the DOM.

<!-- when $scope.myValue is truthy (element is restored) -->

<div ng-if="1"></div>

<!-- when $scope.myValue is falsy (element is removed) -->

<div ng-if="0"></div>

When an element is removed using ngIf its scope is destroyed and a new scope is created when the element is restored. The scope created within ngIf inherits from its parent scope using prototypal inheritance.

If ngModel is used within ngIf to bind to a JavaScript primitive defined in the parent scope, any modifications made to the variable within the child scope will not affect the value in the parent scope, e.g.

<input type="text" ng-model="data">

<div ng-if="true">

<input type="text" ng-model="data">

</div>

To get around this situation and update the model in the parent scope from inside the child scope, use an object:

<input type="text" ng-model="data.input">

<div ng-if="true">

<input type="text" ng-model="data.input">

</div>

Or, $parent variable to reference the parent scope object:

<input type="text" ng-model="data">

<div ng-if="true">

<input type="text" ng-model="$parent.data">

</div>

**ngShow**

The ngShow directive **shows or hides** the given HTML element based on the expression provided to the ngShow attribute. The element is shown or hidden by removing or adding the ng-hide CSS class onto the element. The .ng-hide CSS class is predefined in AngularJS and sets the display style to none (using an !important flag).

<!-- when $scope.myValue is truthy (element is visible) -->

<div ng-show="1"></div>

<!-- when $scope.myValue is falsy (element is hidden) -->

<div ng-show="0" class="ng-hide"></div>

When the ngShow expression evaluates to false then the ng-hide CSS class is added to the class attribute on the element causing it to become hidden. When true, the ng-hide CSS class is removed from the element causing the element not to appear hidden.

Note:- I understand that ng-show and ng-hide affect the class set on an element and that ng-if controls whether an element is rendered as part of the DOM.

1. ng-if will remove elements from DOM. This means that all your handlers or anything else attached to those elements will be lost. For example, if you bound a click handler to one of child elements, when ng-if evaluates to false, that element will be removed from DOM and your click handler will not work any more, even after ng-if later evaluates to true and displays the element. You will need to reattach the handler.
2. ng-show/ng-hide does not remove the elements from DOM. It uses CSS styles to hide/show elements (note: you might need to add your own classes). This way your handlers that were attached to children will not be lost.
3. ng-if creates a child scope while ng-show/ng-hide does not

Service:- You can use services to organize and share code across your app.it is like as singleton. Angular services are:

* Lazily instantiated – Angular only instantiates a service when an application component depends on it.
* Singletons – Each component dependent on a service gets a reference to the single instance generated by the service factory.

Angular offers several useful services (like [$http](https://docs.angularjs.org/api/ng/service/$http)), but for most applications you'll also want to [create your own](https://docs.angularjs.org/guide/services#creating-services).

**Note:** Like other core Angular identifiers, built-in services always start with $ (e.g. $http).

Difference between service,factory,provide.

app.factory('a', fn);

app.service('b', fn);

app.provider('c', fn);

The difference between the three is that:

1. a's stored value comes from running fn
2. b’s stored value comes from newing fn
3. c’s stored value comes from first getting an instance by newing fn, and then running a $getmethod of the instance

which means, there’s something like a cache object inside angular, whose value of each injection is only assigned once, when they've been injected the first time, and where:

cache.a = fn()

cache.b = new fn()

cache.c = (new fn()).$get()

This is why we use this in services, and define a this.$get in providers.

Reference link:- <http://stackoverflow.com/questions/13762228/confused-about-service-vs-factory>

https://toddmotto.com/factory-versus-service

Example:-

var myApp = angular.module('myApp', []);

//service style, probably the simplest one

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

this.sayHello = function() {

return "Hello, World!"

};

});

//factory style, more involved but more sophisticated

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

return {

sayHello: function() {

return "Hello, World!"

}

};

});

//provider style, full blown, configurable version

myApp.provider('helloWorld', function() {

// In the provider function, you cannot inject any

// service or factory. This can only be done at the

// "$get" method.

this.name = 'Default';

this.$get = function() {

var name = this.name;

return {

sayHello: function() {

return "Hello, " + name + "!"

}

}

};

this.setName = function(name) {

this.name = name;

};

});

//hey, we can configure a provider!

myApp.config(function(helloWorldProvider){

helloWorldProvider.setName('World');

});

function MyCtrl($scope, helloWorld, helloWorldFromFactory, helloWorldFromService) {

$scope.hellos = [

helloWorld.sayHello(),

helloWorldFromFactory.sayHello(),

helloWorldFromService.sayHello()];

}​

Syntax: module.service( 'serviceName', function );

Result: When declaring serviceName as an injectable argument you will be provided with the**instance of a function** passed to module.service.

Usage: Could be useful for **sharing utility functions** that are useful to invoke by simply appending () to the injected function reference. Could also be run with injectedArg.call( this ) or similar.

Factories

Syntax: module.factory( 'factoryName', function );

Result: When declaring factoryName as an injectable argument you will be provided the **value that is returned by invoking the function reference** passed to module.factory.

Usage: Could be useful for returning a **'class'** function that can then be new'ed to create instances

service - registered function will be invoked as a constructor (aka 'newed')

factory - registered function will be invoked as a simple function

**Service** : returns the actual function: Useful for sharing utility functions that are useful to invoke by simply appending () to the injected function reference.

**Factory** : returns the function's return value: instantiate an object like new Object() in java.

**Provider** : returns the output of the function's $get function: Configurable.

* *Singleton* - called only once, stored, and pass the same object.

NOTE: If you use factory with <constructor function> or service with <function with a return value>, it will not work.

**Services**

**Syntax**: module.service( 'serviceName', function );

**Result**: When declaring serviceName as an injectable argument you will be provided the actual function reference passed to module.service.

**Usage**: Could be useful for sharing utility functions that are useful to invoke by simply appending () to the injected function reference. Could also be run with injectedArg.call( this ) or similar.

**Factories**

**Syntax**: module.factory( 'factoryName', function );

**Result**: When declaring factoryName as an injectable argument you will be provided the value that is returned by invoking the function reference passed to module.factory.

**Usage**: Could be useful for returning a 'class' function that can then be new'ed to create instances.

**Providers**

**Syntax**: module.provider( 'providerName', function );

**Result**: When declaring providerName as an injectable argument you will be provided the value that is returned by invoking the $get method of the function reference passed to module.provider.

**Usage**: Could be useful for returning a 'class' function that can then be new'ed to create instances but that requires some sort of configuration before being injected. Perhaps useful for classes that are reusable across projects? Still kind of hazy on this one." Ben

What are benefits of angular.js?

1.MVC pattern(code modularising)

2.less dom manipulation compare to jquery

3.handle normal business logic in client side (increase performance)

4.handle filter operation in clientside

5.faster

6.data-binding(which means it will bind data html(view) to model and refreshing page it will load data).

7.page-posting(typically in previous web development application client was requesting to server to specific url page and server returns that page template with content. But in angular it is not like that, it is single page application in which page is loading and redirecting based on routing and request goto server only for specific div ir content.it won’t load complete page. )

Mvvm-model-view-view-model

Mvc -model\_view\_conroller

Model-it helps to carry data, business logic,set of rules.

View-it is visible like html contents.

Controller-controls over all application.

What is directive ?-it is nothing but marker in dom(document object module).it tell the compiler what operation and behaviour to do.it extends the html tag properties in angular.js.

Types of directives:-element,class,attribute,comment

Benefits of angular js 2.0 ?

Fast and support in latest browser.

Simplify angular.js app development.

Mobility driven-approach like as low memory ,performance tunning,touch support.

Fully encapsulated custom element development.

Scalable,extensible,Strong and flexible routing.

Revamped dependency injection.

Using es6

No two-way data-binding

Typescript default language

Decorator syntax

No conrollers,now we have components

Why you moved to angular.js even already many libraries are available in market?

As a developer earlier I used native javascript and,jquery. with help of this library developer resolves problem with some limited extent not completely.Even you can’t get more support for this technology and as well as can’t get frequent releases from group or developers. So after developing something after sometimes(1-2 years) u will get stuck.

So I moved to angular.js bcz it is developed and maintain by google dedicated enginners which give me more confidence.it provides complete package like jquery ,testing framework like gasmine ,karma.

What about jquery was popular so much earlier?

Yes jquery was popular ,but angular.js framework maintain jqlite engine.so as a developer or users no need to worry about jquery library and all those aspeects.

Is it possible to use together javascript framework like as backbone.js,ext.js.angular.js?

No,every framework have some specific limitation and uses and subscription. Based on specific requirement its uses. If u want to integrate bootstrap kind of thing then angular.js is preferable.

If yes, then there is no straight forward,there wiil be more complexity.

Which kind of yours application? single page application or multipage application.

If spa then it’s ok.if multile then why u r using angular.js.

How to achieve one-way binding, two-way binding?

Form validation?

What is AOT(Ahead-of-Time (AOT) Compiler)?

<https://angular.io/guide/aot-compiler>

key features of angular.js?

scope,controller,filters,services,testable,directives,model,view

Injecter-it is service locator used to retrieve object instances.

Data-binding-it is synchronization of data between model and view components.

When ng-init called?-when page will render or when take turns to back .

What is difference between $rootscope and $scope>

1. Scopes provide APIs**($watch)** to observe model mutations.
2. Scopes provide APIs **($apply)** to propagate any model changes through the system into the view from outside of the “Angular realm” (controllers, services, Angular event handlers).
3. Scopes provide context against which expressions are evaluated.
4. For example {{username}} expression is meaningless, unless it is evaluated against a specific scope which defines the username property.Scope is the glue between application controller and the view

**Scope characteristics of $rootscope**

1. Share data between controllers in AngularJS.
2. This method can be useful when you have multiple controllers on a page, some of which need to know when data managed by another controller has been changed.
3. This gives us access to it in the controller, just like a standard AngularJS service (such as $scope, $location, etc).

$**scope** is an object that is accessible from current component e.g Controller, Service only. $**rootScope** refers to an object which is accessible from everywhere of the application we can say it is a global scope of variable.

You can think $**rootScope** as global variable and $scope as local variables.

This image explains difference of $**rootScope** and $**scope** in Angularjs. so $**rootScope** is available globally, no matter what controller you are in, whereas $**scope** is only available to the current controller and it's children.

$**rootScope** is a parent object of all whereas $**scope** angular objects created in a web page.

$**scope** is created with ng-controller while $**rootscope** is created with ng-app.

how to share information between controllers in angularjs?

<https://stackoverflow.com/questions/21919962/share-data-between-angularjs-controllers>

(a) sending data from child to parent controller

$scope.Save(Obj,function(data) {

$scope.$emit('savedata',data);

//pass the data as the second parameter

});

$scope.$on('savedata',function(event,data) {

//receive the data as second parameter

});

(b) sending data from parent to child controller

$scope.SaveDB(Obj,function(data){

$scope.$broadcast('savedata',data);

});

$scope.SaveDB(Obj,function(data){`enter code here`

$rootScope.$broadcast('saveCallback',data);

});

Config() and run():-

Run blocks are the closest thing in AngularJS to the main method. A run block is the code which needs to run to kickstart the application. It is executed after all of the services have been configured and the injector has been created. Run blocks typically contain code which is hard to unit-test, and for this reason should be declared in isolated modules, so that they can be ignored in the unit-tests.

<http://firstcrazydeveloper.com/Blogs/BlogView.html/46/importance-of-config-and-run-blocks-in-angularjs>

how watch is working.

how to stop and deactivate watch in angularjs?

var unbindWatch = $scope.$watch("myvariable", function() {

//...

});

setTimeout(function() {

unbindWatch();

}, 1000);

var listener = $scope.$watch(someVariableToWatch, function(....));

$scope.$on('$destroy', function() {

listener(); // call the de-register function on scope destroy

});

# Understanding Angular’s $apply() and $digest()

<https://www.sitepoint.com/understanding-angulars-apply-digest/>

<http://angulartestingquickstart.com/#unit-testing>

<https://ciphertrick.com/2016/06/26/unit-testing-controllers-angularjs-karma-jasmine/>