# Essentials

Inside <head> tag : <script src="//code.angularjs.org/snapshot/angular.min.js"></script>

Inside <body> tag : <script> </script>

Example:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

| <!DOCTYPE html>

| <html>

| <head>

| <script src="//code.angularjs.org/snapshot/angular.min.js" </script>

|</head>

|

|<body ng-app="myApp">

| <script>

| angular.module('myApp', [])

| .controller('myCtrl', ['$scope', function($scope) {

| $scope.list= [];

| $scope.text = 'hello';

| $scope.submit = function() {

| if($scope.text) {

| $scope.list.push($scope.text)

| }

| };

| }]);

| </script>

|

| <form ng-submit="submit()" ng-controller="myCtrl" >

| Enter text and hit enter:

| <input type="text" ng-model="text" name="text">

| <input type="submit" value="Submit" id="submit">

| <pre> list= {{list}}</pre>

| </form>

|</body>

|</html>

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Enter text and hit enter: |\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |Submit|

list: ["hello"]

How does the digest phase work?

In a nutshel, on every digest cycle all scope models are compares against their previous values. This is dirty checking. If change is detected, the watches set on that model are fired. Then another digest cycle executes and so on untill all models are stable.

It is probably important to mention that there is no “.$digest()” polling. That means that every time it is being called deliberately. As long as core directives are used, we don’t need to worry, but when external code changes models the digest cycle needs to be called manually. Usually to do that, “.$apply()” or similar is used, and not “.$digest()” directly.

What is $rootScope and how does it relate to $scope?

$rootScope is the parent object of all $scope Angular objects created in a web page.

What are the DOM and the BOM?

The DOM is the Document Object Model. It’s the view part of the UI. Whatever we are changing in page elements is reflected in the DOM.

BOM is the Browser Object Model, which specificies the global browser objects like window, localstorage, and console.

What Is the Difference Between $Scope And Scope?

Scopes provide APIs ($watch) to observe model mutations.

Scopes provide APIs ($apply) to propagate any model changes through the system into the view from outside of the "AngularJS realm" (controllers, services, AngularJS event handlers).

Scopes can be nested to limit access to the properties of application components while providing access to shared model properties. Nested scopes are either "child scopes" or "isolate scopes". A "child scope" (prototypically) inherits properties from its parent scope. An "isolate scope" does not. See isolated scopes for more information.

$scope : is the glue between application controller and the view.

**---------------**

**scope property :**

---------------

The scope property can be false, true, or an object:

1> false (default): No scope will be created for the directive. The directive will use its parent's scope.

2> true: A new child scope that prototypically inherits from its parent will be created for the directive's element. If multiple directives on the same element request a new scope, only one new scope is created.

3> {...} (an object hash): A new "isolate" scope is created for the directive's template. The 'isolate' scope differs from normal scope in that it does not prototypically inherit from its parent scope. This is useful when creating reusable components, which should not accidentally read or modify data in the parent scope. Note that an isolate scope directive without a template or templateUrl will not apply the isolate scope to its children elements.

------------------------------------------------------------------

!!!!!!!!!!!!!!!!!!! example for scope property !!!!!!!!!!!!!!!!!!!

------------------------------------------------------------------

1- index.html

2- script.js

3- my-customer-ios.html

index.html

==========

<div ng-controller = 'myController' >

<my-customer info='naomi' > </my-customer>

<my-customer info='igor' > </my-customer>

</div>

script.js

=========

angular.module('myApp', [])

.controller ('myController', [ '$scope', function ( $scope ) {

$scope.naomi = { name: 'Naomi', address: '1600 Amphitheater'};

$scope.igor = { name: 'Igor', address: '1800 Amphitheater'};

}])

.directive ('myCustomer', function () {

return {

restrict: 'E',

scope: { customerInfo: '=info'}, //this is isolation scope definition

templateUrl: 'my-cutomer-iso.html',

// template: 'Name: {{customer.name}} Address: {{customer.address}}' // provideing template inline

};

});

my-customer-iso.html

====================

Name: {{ customerInfo.name }} Address: {{ customerInfo.address }}

How this (above isolation scope definition) works ???

= or =attr - set up a bidirectional binding between a local scope property and an expression passed via the attribute attr. The expression is evaluated in the context of the parent scope.

Given <my-component my-attr="parentModel"> and the isolate scope definition scope: { localModel: '=myAttr' }, the property localModel on the directive's scope will reflect the value of parentModel on the parent scope. Changes to parentModel will be reflected in localModel and vice versa. If the binding expression is non-assignable, or if the attribute isn't optional and doesn't exist, an exception ($compile:nonassign) will be thrown upon discovering changes to the local value, since it will be impossible to sync them back to the parent scope.

What are Directives?

At a high level, directives are markers on a DOM element (such as an attribute, element name, comment or CSS class) that tell AngularJS's HTML compiler ($compile) to attach a specified behavior to that DOM element (e.g. via event listeners), or even to transform the DOM element and its children.

AngularJS comes with a set of these directives built-in, like ngBind, ngModel, and ngClass. Much like you create controllers and services, you can create your own directives for AngularJS to use. When AngularJS bootstraps your application, the HTML compiler traverses the DOM matching directives against the DOM elements.

What does it mean to "compile" an HTML template? For AngularJS, "compilation" means attaching directives to the HTML to make it interactive. The reason we use the term "compile" is that the recursive process of attaching directives mirrors the process of compiling source code in compiled programming languages.

What Are Different Ways To Invoke A Directive?

There are four different ways to invoke a directive in an angular application. They are as follows.

1) As an attribute:

<span my-directive></span>

2) As a class:

<span class="my-directive: expression;"></span>

3) As an element:

<my-directive></my-directive>

4) As a comment:

<!-- directive: my-directive expression -->

Creating Directives

================

First let's talk about the API for registering directives. Much like controllers, directives are registered on modules. To register a directive, you use the module.directive API. module.directive takes the normalized directive name followed by a factory function. This factory function should return an object with the different options to tell $compile how the directive should behave when matched.

The factory function is invoked only once when the compiler matches the directive for the first time. You can perform any initialization work here. The function is invoked using $injector.invoke which makes it injectable just like a controller.

angular.module('myApp', [])

.controlller('myController', [$scope, function($scope) {

$scope.customer = {

name: "Jone",

address: "1600 downtown"

};

}])

.directive('myCustomer', function() {

return {

restrict: 'E'

template: 'Name: {{customer.name}} Address: {{customer.address}}

};

});

<div ng-controller="myController">

<my-customer></my-customer>

</div>

output:

======

Name: Jone Address: 1600 downtown

------------------------------------------------------------------------------

FILTERS:

\*\*\*\*\*\*\*

Can be used in:

1. View template

2. Controllers, Services and Directives

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

In VIew Template

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Filters can be applied to expressions: {{ expression | filter }}

e.g. {{ 2 | currency }} => $2.00 ==> formats the number 12 as a currency

{{ 1234 | number:2 }} => 1234.00 ==> formats the number 1234 with 2 decimal points

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

In Controllers, Services and Directives

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

For this, inject a dependency with the name <filterName>Filter into your controller/service/directive. E.g. a filter called number is injected by using the dependency numberFilter. The injected argument is a function that takes the value to format as first argument, and filter parameters starting with the second argument.

script.js

======

angular.module('myApp',[])

.controller ('myController', [ 'filterFilter' function ( filterFilter ) {

this.array = [

{ name : 'John'},

{ name : 'Jane'},

{ name : 'Merry'},

{ name : 'James'},

{ name : 'Jeff'}

];

this.filteredArray = filterFilter (this.array , 'a' );

}]);

index.html

==========

<div ng-controller = "myController as ctrl">

<div>

All entries :

<span ng-repeat ="entry in ctrl.array" > {{ entry.name }} </span>

<div>

<div>

Entries that contain an 'a' :

<span ng-repeat="entry in ctrl.filteredArray > {{entry.name }} </span>

<div>

</div>

Output:

=======

All entries: Tobias Jeff Brian Igor James Brad

Entries that contain an "a": Tobias Brian James Brad

Creating Filter

============

Writing your own filter is very easy:

1> just register a new filter factory function with your module. Internally, this uses the filterProvider.

2> This factory function should return a new filter function which takes the input value as the first argument. Any filter arguments are passed in as additional arguments to the filter function.

filter is someting like,

|=========fn===========|

----input----> | [ process the input ]|---output---->.

|======================|

Therefore, filte is nothing but a function which takes some input and returns output.

Difference from normal function is that ===>

1> it is registered with the module ===> with

2> to be used, it has to be injected into a controller ===> by adding suffix (Filter)

script.js

\*\*\*\*\*\*\*

angular.module('myApp',[])

.filter ('reverse', function() {

return function (input, uppercase) {

input = input || ' ';

var out =' ';

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

out = input.charAt(i) + out;

}

if(uppercase){

out = out.toUpperCase();

}

return out;

};

})

.controller('myController', [$scope, 'reverseFilter', function ( $scope, reverseFilter) {

$scope.greeting = 'hello';

$scope.filteredgreeting = reverseFilter(greeting);

}]);

index.html

\*\*\*\*\*\*\*\*\*

<div ng-controller='myController'>

<input ng-model = 'greeting' type ='text' > <br>

No filter : {{ greeting }} <br>

Reverse : {{ greeting | reverse }} <br>

Reverse + uppercase : {{ greeting | reverse : true }} <br>

Reverse, filtered in controller: {{filererdgreeting }} <br>

</div>

output

\*\*\*\*\*\*

No filter: hello

Reverse: olleh

Reverse + uppercase: OLLEH

Reverse, filtered in controller: olleh

Normalization

============

AngularJS normalizes an element's tag and attribute name to determine which elements match which directives. We typically refer to directives by their case-sensitive camelCase normalized name (e.g. ngModel). However, since HTML is case-insensitive, we refer to directives in the DOM by lower-case forms, typically using dash-delimited attributes on DOM elements (e.g. ng-model).

The normalization process is as follows:

Strip x- and data- from the front of the element/attributes.

Convert the :, -, or \_-delimited name to camelCase.

For example, the following forms are all equivalent and match the ngBind directive:

script.js

----------

angular.module('docsBindExample', [])

.controller('Controller', ['$scope', function($scope) {

$scope.name = 'Max Karl Ernst Ludwig Planck (April 23, 1858 – October 4, 1947)';

}]);

index.html

--------------

<div ng-controller="Controller">

Hello <input ng-model='name'> <hr/>

<span ng-bind="name"></span> <br/>

<span ng:bind="name"></span> <br/>

<span ng\_bind="name"></span> <br/>

<span data-ng-bind="name"></span> <br/>

<span x-ng-bind="name"></span> <br/>

</div>

output:

--------

Hello |Max Karl st Ludwig Planck (April 23, 1858 – October 4, 1947)|

Max Karl st Ludwig Planck (April 23, 1858 – October 4, 1947)

Max Karl st Ludwig Planck (April 23, 1858 – October 4, 1947)

Max Karl st Ludwig Planck (April 23, 1858 – October 4, 1947)

Max Karl st Ludwig Planck (April 23, 1858 – October 4, 1947)

Max Karl st Ludwig Planck (April 23, 1858 – October 4, 1947)

Max Karl st Ludwig Planck (April 23, 1858 – October 4, 1947)

ngModel.NgModelController

**Overview**

NgModelController provides API for the ngModel directive. The controller contains services for:

data-binding,

validation,

CSS updates,

value formatting,

parsing.

It purposefully does not contain any logic which deals with DOM rendering or listening to DOM events. Such DOM related logic should be provided by other directives which make use of NgModelController for data-binding to control elements. AngularJS provides this DOM logic for most input elements.

**Methods**

$render(); $isEmpty(value); $setPristine(); $setDirty(); $setUntouched();

$setTouched();$rollbackViewValue()$validate();$commitViewValue();$setViewValue(value, trigger);$overrideModelOption(options);$processModelValue();$setValidity(validationErrorKey, isValid);

**Properties**

$viewValue, $modelValue, $parsers, $formatters, $validators, $asyncValidators, $viewChangeListeners, $error, $pending, $untouched, $touched, $pristine, $dirty, $valid, $invalid, $name

**Form Validation in AngularJS**

Client side validation in AngularJS

note: novalidate is used to disable browser's native validation

----

1- Using CSS Classes

2- Validation tokens

3- Directives

**Using CSS Classes**

To allow styling of form as well as controls, ngModel adds these CSS classes:

1> ng-valid : is set if the form is valid.

2> ng-invalid : is set if the form is invalid.

3> ng-prestine:control hasn't been interacted been yet

4> ng-dirty : the control has been interacted with

5> ng-touched

6> ng-untouched

7> ng-pending : $asyncValidatiors are unfulfilled

**Built-in validation tokens:**

1. email
2. max
3. maxlength
4. min
5. minlength
6. number
7. pattern
8. required
9. url
10. date
11. datetimelocal
12. time
13. week
14. month

**Form**

**FormController:** it keeps track of all it's controls and nested forms as well as the state of them, such as being valid/invalid or dirty/prestine.

Each form directive <ng-form> creates an instance of FormCOntroller.

**Methods**

1. $rollbackViewValue();
2. $commitViewValue();
3. $addControl(control);
4. $getControls();
5. $removeControl(control);
6. $setDirty();
7. $setPristine();
8. $setUntouched();
9. $setSubmitted();
10. $setValidity(validationErrorKey, isValid, controller);

**Properties**

1. $pristine
2. $dirty
3. $valid
4. $invalid
5. $submitted
6. $pending
7. $error

**Explanation of Methods**

**$setDirty():** Sets the form to a dirty state. This method CAN BE called to add the 'ng-dirty' class and set the form to a dirty state (ng-dirty class). This method will also propagate to parent forms.

**$setPristine():** Sets the form to its pristine state. This method sets the form's $pristine state to true, the $dirty state to false, removes the ng-dirty class and adds the ng-pristine class. Additionally, it sets the $submitted state to false.

This method will also propagate to all the controls contained in this form.

Setting a form back to a pristine state is often useful when we want to 'reuse' a form after saving or resetting it.

**$setUntouched():** Sets the form to its untouched state. This method CAN BE called to remove the 'ng-touched' class and set the form controls to their untouched state (ng-untouched class).

Setting a form controls back to their untouched state is often useful when setting the form back to its pristine state.

**$setSubmitted():** Sets the form to its $submitted state. This will also set $submitted on all child and parent forms of the form.

**form directives**

It is directive that instantiates Formcontroller.If name attribute of this directive is specified then the form controller is publisdhes onto the current scope under this name.

<form

[name="string"]>

...

</form>

Note: it can be used as an element

Alias: ngForm

In angularJS form can be nested, which means the outer form is valid when all inner chils forms are valid as well.

However, browser does not allow nesting of <form> element, and therefore AngularJS provides ngForm directive, which behaves identically to form, but can be nested

You can use one of the following ways to specify what javascript method shoud be called when a form is submitted:

* 1. ngSubmit - this directive is used on the form element.
  2. ngClick - this directive used in the first button or input field of type submit (input[type=submit])

\*\*To prevent double execution of the handler, use only one of the ngSubmit or ngClick directives. This is because of the following form submission rules in the HTML specification:

Form has:

(i) 1 input field -----then-----> hitting enter in this field -----triggers----> ngSubmit

(ii) 1 or more input field + 1 or more buttons -----then----->

hitting enter in input fields -----triggers---->

click handler(ngClick) on the first button [OR] submit handler(ngSubmit) on the enclosing form

(iii) 2 or more input field only -----then-----> hitting enter in this field -----triggers----> NOTHING!!!

**ng-form directive**

Helper directive that makes it possible to create control groups inside a form directive. These "child forms" can be used, for example, to determine the validity of a sub-group of controls.

Note: ngForm cannot be used as a replacement for <form>, because it lacks its built-in HTML functionality. Specifically, you cannot submit ngForm like a <form> tag. That means, you cannot send data to the server with ngForm, or integrate it with ngSubmit.

Can be used as:

element : <ng-form name:"string"></ng-form>

attribute : <ANY ng-form="string"></ANY>

CSS class : <ANY class="ng-form:string"></ANY>

**AngularJS form validation**

------------------------------------------------------------------------------------------------------------------------

1. CSS Classes [added by ngModel]
2. Directives for validation (required, pattern, minlength, maxlength, min, max)
3. Properties ($prestine, $dirty, $valid, $invalid, $submitted, $error)
4. AngularJS directives: ng-requires, ng-pattern
5. **CSS classes**
   1. ng-valid is set if the form is valid.
   2. ng-invalid is set if the form is invalid.
   3. ng-pending is set if the form is pending.
   4. ng-pristine is set if the form is pristine.
   5. ng-dirty is set if the form is dirty.
   6. ng-submitted is set if the form was submitted.

Keep in mind that ngAnimate can detect each of these classes when added and removed.

Lets see how we can use them. Usage:

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

**In Form:** myForm.$dirty

**For Field:**  myForm.firldName.$dirty

**In CSS:** .ng-dirty{

background-color: yellow;

}

**AngularJS directives:** ng-required, ng-pattern

**ng-required**

> It adds \*required\* "validator" to mgModel.

> This directive sets 'required' attribute on the element if AngularJS expression inside ng-required evaluates to true.

> The validator will set the required error key to true if the required attribute is set and calling NgModelController.$isEmpty [with the ngModel.$viewValue] returns true

**ng-pattern**

> It adds \*pattern\* "validator" to mgModel.

> Often used for text-based input controls.

> The validator sets the pattern error key if the ngModel.$viewValue does not match a RegExp which is obtained from the ngPattern attribute value.

the value is an AngularJS expression:

-> If the expression evaluates to a RegExp object, then this is used directly.

-> If the expression evaluates to a string, then it will be converted to a RegExp after wrapping it in ^ and $ characters. For instance, "abc" will be converted to new RegExp('^abc$').

-> If the value is a RegExp literal, e.g. ngPattern="/^\d+$/", it is used directly.

**Usage:**

<ANY ... ng-pattern="x"></ANY>

x ==> Expression or regex , AngularJS expression that must evaluate to a RegExp or a String parsable into a RegExp, or a RegExp literal.

============================Self Understanding==================================

bolean type: $prestine, $dirty, $valid, $invalid, $submitted

object type: $error

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

| <!DOCTYPE html>

| <html>

| <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>

| <body ng-app="">

|

|

| <p> Enter in input field </p>

| <form name ="myForm">

| <input type="text" name="myInput" ng-model="myInput" ng-required=true>

| </form>

| <p> Error Object : <span style="color:red"> {{ myForm.myInput.$error }}</span></p>

| <p> $error.required : <span style="color:red"> {{ myForm.myInput.$error.required }}</span></p>

| <p> $valid state of input field is : <span style="color:red"> {{ myForm.myInput.$valid }}</span></p>

|

| </body>

| </html>

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**OUTPUT [no input]**

------------------

Enter in input field

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

Error Object : {"required":true}

$error.required : true

$valid state of input field is : false

**OUTPUT [with "asdf" input]**

------------------

Enter in input field

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|asdf\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

Error Object : {} //===> this indicates that error object is set to NULL for valid input

$error.required :

$valid state of input field is : true

How to use ng-options?

It needs:

1) to be enclosed within select tag

2) add ng-model

<script>

angular.module('myApp', [])

.controller('myCtrl', ['$scope', function ($scope) {

$scope.items = ['---select---', 'option1','option2'];

$scope.selectedOption = '---select---';

</script>

</head>

<body ng-app="myApp">

<div ng-controller="myCtrl">

Select a name: <select ng-model="selectedOption" ng-options="item for item in items">

</div>

</body>

//this will create a dropdown with options as defined in items array and selected value will be assigned to selectedOption

How to use ng-include?

ng-include directive will compile and include one html page in another page.

<div ng-include="'myTable.htm'"></div>

How to Redirect to another HTML page in Angular JS ?

var path = "/afterLogin.html";

window.location.href = path;

AngularJS Expression

AngularJS expression is like JavaScript expression surrounded with braces - {{ expression }}. AngularJS evaluates the specified expression and binds the result data to HTML.

AngularJS expression can contain literals, operators and variables like JavaScript expression. For example, an expression {{2/2}} will produce the result 1 and will be bound to HTML.

Example: Expression

<!DOCTYPE html>

<html >

<head>

<script src="~/Scripts/angular.js"></script>

</head>

<body >

<h1>AngularJS Expression Demo:</h1>

<div ng-app>

2 + 2 = {{2 + 2}} <br />

2 - 2 = {{2 - 2}} <br />

2 \* 2 = {{2 \* 2}} <br />

2 / 2 = {{2 / 2}}

</div>

</body>

</html>

Result:

2 + 2 = 4

2 - 2 = 0

2 \* 2 = 4

2 / 2 = 1

**AngularJS expression is like JavaScript code expression except for the following differences:**

1. AngularJS expression cannot contain conditions, loops, exceptions or regular expressions e.g. if-else, ternary, for loop, while loop etc.
2. AngularJS expression cannot declare functions.
3. AngularJS expression cannot contain comma or void.
4. AngularJS expression cannot contain return keyword.

**Ng-click vs onclick**

ng-click holds an angular expression. Angular expressions are evaluated in the context of an Angular [scope](http://docs.angularjs.org/guide/scope), which is bound to the element having the ng-click attribute or an ancestor of that element.

The Angular expression language doesn't include flow control statements and can't declare variables or define functions. These limitations mean templates can only access variables and run functions made available by a controller or directive.

**onclick vs addEventListener**

**What’s the difference between these two lines of code?**

element.onclick = function() { /\* do stuff \*/ }

element.addEventListener('click', function(){ /\* do stuff \*/ }, false);

They apparently do the same thing: listen for the click event and execute a callback function. Nevertheless, they’re not equivalent. If you ever need to choose between the two, this could help you to figure out which one is the best for you.

The main difference is that **onclick is just a property**, and like all object properties, if you write on more than once, it will be **overwritten**. With **addEventListener**() instead, we can simply bind an event handler to the element, and we can call it each time we need it without being worried of any overwritten properties.

In first place I was tempted to keep using onclick, because it’s shorter and looks simpler… and in fact it is. But I don’t recommend using it anymore. It’s just like using inline JavaScript. Using something like <button onclick="doSomething()"> – **that’s inline JavaScript** – is highly discouraged nowadays (inline CSS is discouraged too, but that’s another topic).

However, the addEventListener() function, despite it’s the standard, just doesn’t work in old browsers (Internet Explorer below version 9), and this is another big difference. If you need to support these ancient browsers, you should follow the onclick way. But you could also use jQuery (or one of its alternatives): it basically simplifies your work and reduces the differences between browsers, therefore can save you a lot of time.

**The difference you could see if you had another couple of functions:**

var h = document.getElementById('a');

h.onclick = doThing\_1;

h.onclick = doThing\_2;

h.addEventListener('click', doThing\_3);

h.addEventListener('click', doThing\_4);

Functions 2, 3 and 4 work, but 1 does not.  1 gets overridden by 2 and is never called. This is because addEventListener does not overwrite existing event handlers, whereas onclick overrides any existing onclick = fn event handlers.

The other significant difference, of course, is that onclick will always work, whereas addEventListener does not work in Internet Explorer before version 9. You can use the analogous attachEvent (which has slightly different syntax) in IE <9.

So if I need multiple functions for one event, I am stuck with addEventListener, and I have to write more code for attachEvent just to accomodate IE.

**AngularJS Events**

AngularJS includes certain directives which can be used to provide custom behavior on various DOM events, such as click, dblclick, mouseenter etc. The following table lists AngularJS event directives.

| Event Directive |
| --- |
| ng-blur |
| ng-change |
| ng-click |
| ng-dblclick |
| ng-focus |
| ng-keydown |
| ng-keyup |
| ng-keypress |
| ng-mousedown |
| ng-mouseenter |
| ng-mouseleave |
| ng-mousemove |
| ng-mouseover |
| ng-mouseup |

Let's take a look at some of the important event directives.

## **ng-click**

The ng-click directive is used to provide event handler for click event.

Example: ng-click

<!DOCTYPE html>

<html >

<head>

<script src="~/Scripts/angular.js"></script>

</head>

<body ng-app="myApp">

<div ng-controller="myController">

Enter Password: <input type="password" ng-model="password" /> <br />

<button ng-click="DisplayMessage(password)">Show Password</button

</div>

<script>

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

myApp.controller("myController", function ($scope, $window) {

$scope.DisplayMessage = function (value) {

$window.alert(value)

}

});

</script>

</body>

</html>

In the above example, ng-click directive is used to call a DisplayMessage() function with the 'password' parameter when a user clicks a button. A 'password' is a model property defined using ng-model directive in the input box. The DisplayMessage() function is attached to a $scope object in myController, so it will be accessible from button click as button comes under myController. The [$window](https://www.tutorialsteacher.com/angularjs/angularjs-window-service) service is used to display an alert.

## **Mouse Events**

The following example demonstrates important mouse event directives - ng-mouseenter and ng-mouseleave.

Example: Mouse Events

<!DOCTYPE html>

<html>

<head>

<script src="~/Scripts/angular.js"></script>

<style>

.redDiv {

width: 100px;

height: 100px;

background-color: red;

padding:2px 2px 2px 2px;

}

.yellowDiv {

width: 100px;

height: 100px;

background-color: yellow;

padding:2px 2px 2px 2px;

}

</style>

</head>

<body ng-app>

<div ng-class="{redDiv: enter, yellowDiv: leave}" ng-mouseenter="enter=true;leave=false;" ng-mouseleave="leave=true;enter=false">

Mouse <span ng-show="enter">Enter</span> <span ng-show="leave">Leave</span>

</div>

</body>

</html>

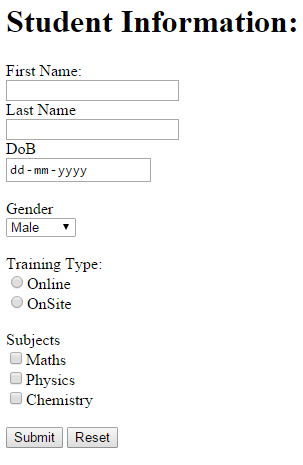
In the above example, the ng-class directive includes map of CSS classes, so redDiv will be applied if enter=true and yellowDiv will be applied if leave=true. The ng-mouseenter directive sets 'enter' to true, which will apply redDiv class to the <div> element. In the same way, ng-mouseleave will set leave to true, which will apply yellowDiv class.

AngularJS Forms

The HTML form is a collection of input controls where user can enter the data. Here, you will learn how to display AngularJS form and submit the data.

## **An AngularJS Form Example**

We will create following Student Information form with submit and reset functionality.

[](https://www.tutorialsteacher.com/Content/images/ng/angular-form.png)

Sample AngularJS Form

**The following is the code of the above form**.

<!DOCTYPE html>

<html ng-app="studentApp">

<head>

<script src="~/Scripts/angular.js"></script>

</head>

<body ng-controller="studentController">

<h1>Student Information:</h1>

<form **ng-submit="submitStudnetForm()"** >

<label for="firstName" >First Name: </label><br />

<input type="text" id="firstName" ng-model="student.firstName" /> <br />

<label for="lastName">Last Name</label><br />

<input type="text" id="lastName" ng-model="student.lastName" /> <br />

<label for="dob" >DoB</label><br />

<input type="date" id="dob" ng-model="student.DoB" /> <br /><br />

<label for="gender" >Gender</label> <br />

<select id="gender" ng-model="student.gender">

<option value="male">Male</option>

<option value="female">Female</option>

</select><br /> <br />

<span>Training Type:</span><br />

<label><input value="online" type="radio" name="training" ng- model="student.trainingType" />Online</label><br />

<label><input value="onsite" type="radio" name="training" ng-model="student.trainingType" />OnSite</label> <br /><br />

<span>Subjects</span><br />

<label><input type="checkbox" ng-model="student.maths" />Maths</label> <br />

<label><input type="checkbox" ng-model="student.physics"/>Physics</label><br />

<label><input type="checkbox" ng-model="student.chemistry" />Chemistry</label><br /><br />

<input type="submit" value="Submit" />

<input type="reset" ng-click="resetForm()" value="Reset" />

</form>

<script>

//1. create app module

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

//2. create controller

studentApp.controller("studentController", function ($scope, $http) {

//3. attach originalStudent model object

$scope.originalStudent = {

firstName: 'James',

lastName: 'Bond',

DoB: new Date('01/31/1980'),

gender: 'male',

trainingType: 'online',

maths: false,

physics: true,

chemistry: true

};

//4. copy originalStudent to student. student will be bind to a form

$scope.student = angular.copy($scope.originalStudent);

//5. create submitStudentForm() function. This will be called when user submits the form

$scope.submitStudnetForm = function () {

var onSuccess = function (data, status, headers, config) {

alert('Student saved successfully.');

};

var onError = function (data, status, headers, config) {

alert('Error occured.');

}

$http.post('/student/submitData', { student:$scope.student })

.success(onSuccess)

.error(onError);

};

//6. create resetForm() function. This will be called on Reset button click.

$scope.resetForm = function () {

$scope.student = angular.copy($scope.OriginalStudent);

};

});

</script>

</body>

</html>

The following is a step by step explanation of the above example:

1. Create an HTML page and wrap all the necessary input controlls into <form> tag.
2. Create the AngularJS application module in the <script> tag.
3. Create studentController in application module.
4. Create originalStudent object and attach to the $scope with required properties. This will stay unchanged during entire life cycle.
5. Create new student object and attach to the $scope and copy all the properties and values from originalStudent. This student object will be bound to the form using ng-model directive. Therefore, if user changes form values then the student object will also get changed.
6. Create submitStudnetForm function which will get called when user submits the form using Submit button. Here, send http POST request to the remote server to submit the data using [$http service](https://www.tutorialsteacher.com/angularjs/angularjs-service-http).
7. Create resetForm() function, which will reset the form values to the originalStudent values by copying it to student object.
8. Apply ng-app, ng-controller directives.
9. Apply ng-model directives to each HTML input element to bind appropriate properties of student object.
10. Apply ng-submit directive to form which will call submitStudentForm() on the form submit event.
11. Apply ng-click directive to reset button which will call resetForm() on the button click event.

An AngularJS form can be submitted using either ng-submit or ng-click directive but not both.

**Ng-submit:** Binds angular expression to onsubmit event when form does not include action attribute.

**Ng-click:** Binds angular expression to onclick event.

Note : The angular form can be submitted using ng-submit directive on the form tag or using ng-click directive on <input type="submit" /> element. Use either ng-submit or ng-click directive but not both to submit the form. The form will be submitted twice if both ng-submit and ng-click directives are used.

*What is Novalidate form tag?*

**novalidate** attribute is used to disable browser's native form validation. You can use it when you need do your own AngularJS custom validation.

You can use the same ones used by the HTML 5 specification in Angular,so you can add the **novalidate** attribute to the form element, which tells the browser **not** to use its **native validation**. Because different browsers have different implementation validations. Since Angular get validation itself, the browser don't need to do validation implementation.

**What is the difference between novalidate and formnovalidate attributes?**

The novalidate and formnovalidate attributes are used to bypass validation. The novalidate attribute is applied to a form and prevents it from validation. The **formnovalidate is applied to input type submit button**, which overrides the novalidate. It submits the form without validating.

The novalidate attribute is also a Boolean attribute, but using it won’t validate the form of submission. The formnovalidate attribute in HTML is useful when you have a form with more than one submit button.

You can try to run the following code to learn how to use novalidate attribute in HTML. In the following example, if you will add text in the <input type=”number” > field, then it won’t show an error.

<!DOCTYPE html>

<html>

   <head>

      <title>HTML novalidate attribute</title>

   </head>

 <body>

      <form action = "" method = "get" novalidate>

         Team Name<br><input type = "name" name = "tname"><br>

         Team Rank<br><input type = "number" name = "trank"><br>

         <input type = "submit" value = "Submit">

      </form>

   </body>

</html>

You can try to run the following code to learn how to use the formnovalidate attribute in HTML. If you will select the submit button with no validation, then the form won’t get validate.

<!DOCTYPE html>

<html>

   <head>

      <title>HTML formnovalidate attribute</title>

   </head>

 <body>

      <form action = "" method = "get">

         Rank <input type="number" name="rank"><br>

         <input type="submit" value="Submit"><br>

         <input type="submit" formnovalidate="formnovalidate"

            value="Submit with no validation”>

      </form>

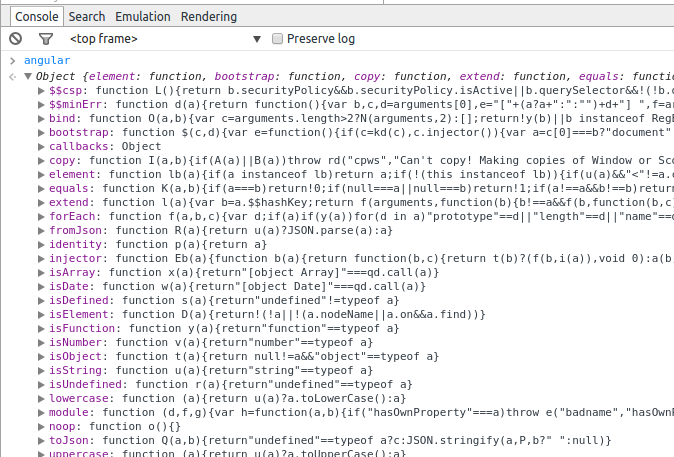
   </body>

</html>

[How can I tell whether a web app was built using Angular (or other technologies)?](https://stackoverflow.com/questions/29950213/how-can-i-tell-whether-a-web-app-was-built-using-angular-or-other-technologies)

* Application declared using ng-app directive
* very simple controller and directive
* check for ng-model, ng-repeater attributes in the code. All these attibutes are written in small letters.
* Also you can check by typing in the console(ctrl + shift + i) and navigate to console tab. There type in window.angular.version --> it displays the version of the site your are currently inspecting.

=====&====

The best way to check is to write "angular" on browser console. If you get any object [With child objects as "bind","bootstrap","callbacks","module" etc.] then its an angular web app.

**Exception Handling in AngularJS**

Every application needs proper exception handling mechanism. You can use try, catch, and finally block of JavaScript to handle exceptions in AngularJS modules.

tip$exceptionHandler does not handle syntax errors.

AngularJS also includes built-in $exceptionHandler service, which handles uncaught exceptions in the application.

The default implementation of $exceptionHandler service logs the exception into the browser console. You can override this service as per your requirement.

The following example demonstrates uncaught exception handling using $exceptionHandler service.

Example: $exceptionHandler

<!DOCTYPE html>

<html ng-app="studentApp">

<head>

<script src="~/Scripts/angular.js"></script>

</head>

<body class="container" ng-controller="studentController">

Status: {{status}} <br />

Data: {{data}} <br />

<input type="button" value="Get Data" ng-click="getStudent()" />

<script>

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

app.config(function ($provide) {

$provide.decorator('$exceptionHandler', function ($delegate) {

return function (exception, cause) {

$delegate(exception, cause);

alert('Error occurred! Please contact admin.');

};

});

});

app.controller("studentController", function ($scope) {

var onSuccess = function (response) {

$scope.status = response.status;

$scope.data = response.data;

};

var onError = function (response) {

$scope.status = response.status;

$scope.data = response.data;

}

$scope.getStudent = function () {

$http.get("/getdata").then(onSuccess, onError);

};

});

</script>

</body>

</html>

In the above example, we override the $provide service's default behavior using $provide.decorate() method in the app.config() method. The decorate method allow us to override or modify the behavior of the service. So, in the decorate method, we display custom error messages along with logging exception messages to the browser console.

Note that we have used $http service in the studentController. However, we have not included $http service as a parameter in the controller function to raise an exception for demo purpose. Now, the exception will be handled by $exceptionHandler and displays an alert message.

<https://stackoverflow.com/questions/28621958/how-does-the-browser-recognize-angularjs-tags-in-html>

We use ng-app directive to bootstrap the application as an angular application.Once you have this directive (ng-app="myApp") in your HTML root element all the nested ng elements are identified and are compiled .

**For example**

angular.module('myApp', []);//module definition in JS File

<body ng-app="myApp">

<div ng-controller="myCntrl">

<input type="text" ng-model="User"/> //this will be recognized and compiled

</div>

I hope this is what you were looking for.