AngularJS

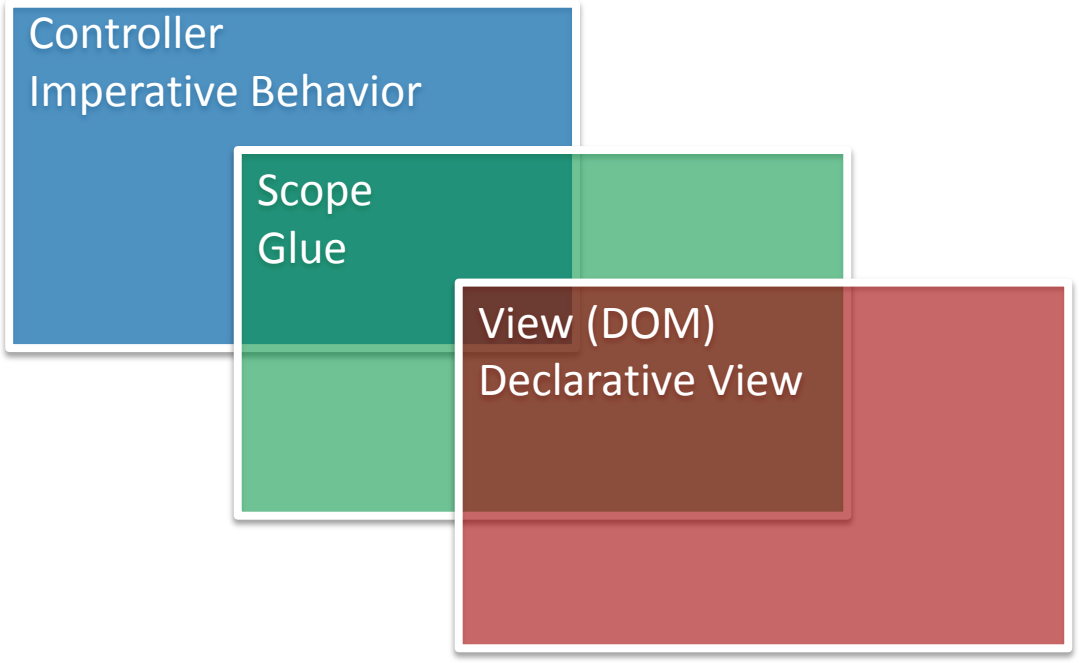
AngularJS Directives

Lesson Objectives

- **Directives**
- **Built-In Directives**
- **Creating Custom Directives**

Controllers

- **Controller is used to provide the business logic behind the view and construct and value the model.**
- **The goal is to not manipulate the DOM at all in the controller, which we have done using other frameworks.**



Controllers – Best practices

- **Controllers should not know anything about the view they control.**
- **Controllers should be small and focused.**
- **Controllers should not talk to other controllers**
- **Controllers should not own the domain model.**

Directives

- Directives are ways to transform the DOM through extending HTML and provides new functionality to it.
- As a good practice DOM manipulations need to be done in directives.
- Directive is simply a function that we run on a particular 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 or even transform the DOM element and its children.
- Directives are actually defined with camelCase in the JavaScript, but applied with a dash to the HTML.
- Angular comes with a set of built-in directives along with that we can create our own directives.

Directives

- **Angular directive can be specified in 3 ways**
- **As a tag**
  - <ng-form />
- **As an attribute**
  - <div ng-form />
- **As a class**
  -
- **All the in-built directives in angular cannot be specified with all the 3 ways, some of them can be specified in 1 or 2 ways only.**

Built-In Directives

- **Angular provides a suite of built-in directives.**
- **ngApp**
  - Placing ng-app on any DOM element marks that element as the beginning of the $rootScope.
  - $rootScope is the beginning of the scope chain, and all directives nested under the ng-app in your HTML inherit from it.
  - $rootScope can be accessed via the run method
  - Using $rootScope is like using global scope hence it is not a best practice.
  - We can use ng-app once per document.
- **ngController**
  - This directive is used to place a controller on a DOM element.
  - Instead of defining actions and models on $rootScope, use ng-controller

Built-In Directives

- **ngBind**
  - The ngBind attribute tells Angular to replace the text content of the specified HTML element with the value of a given expression and to update the text content when the value of that expression changes.
  - It is preferable to use ngBind instead of {{ expression }}. if a template is momentarily displayed by the browser in its raw state before Angular compiles it. Since ngBind is an element attribute, it makes the bindings invisible to the user while the page is loading.
- **ngBindTemplate**
  - The ngBindTemplate directive specifies that the element text content should be replaced with the interpolation of the template in the ngBindTemplate attribute. Unlike ngBind, the ngBindTemplate can contain multiple {{ }} expressions..

Built-In Directives

- **ngBindHtml**
  - Creates a binding that will innerHTML the result of evaluating the expression into the current element in a secure way.
  - **ngSanitize (angular-sanitize.js)** need to be included as module's dependencies to evaluate in a secure way, else it will throw error "Attempting to use an unsafe value in a safe context".
  - We can bypass sanitization by binding to an explicitly trusted value via **$sce.trustAsHtml**. Same can be done using **ng-bind-html-unsafe** which has been removed in Angular 1.2
- **ngShow**
  - The ngShow directive shows or hides the given HTML element based on the expression provided to the ngShow attribute.

… Continued

Built-In Directives

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

- **ngHide**
  - The ngHide directive shows or hides the given HTML element based on the expression provided to the ngHide attribute.
  - When the ngHide expression evaluates to a truthy value then the .ng-hide CSS class is added to the class attribute on the element causing it to become hidden. When falsy, the ng-hide CSS class is removed from the element causing the element not to appear hidden.

Built-In Directives

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

- **ngCloak**
  - It's an alternative to **ngBind** directive. It is used to prevent the Angular html template from being briefly displayed by the browser in its raw (uncompiled) form while your application is loading.
  - Use this directive to avoid the undesirable flicker effect caused by the html template display. The directive can be applied to the <body> element, but the preferred usage is to apply multiple ngCloak directives to small portions of the page to permit progressive rendering of the browser view.

…Continued

Built-In Directives

- We need to add the following styles in our HTML to make ngCloak to work
- *[ng\:cloak], [ng-cloak], [data-ng-cloak], [x-ng-cloak], .ng-cloak, .x-ng-cloak {*

  display: none !important;}
- In Legacy browsers, like IE7 we need to add the css class *ng-cloak* in addition to the ngCloak directive

ngStyle

- ngStyle directive allows you to set CSS style on an HTML element.
- CSS style names and values must be quoted.

ngClass

- ngClass directive allows you to dynamically set CSS classes on an HTML element by databinding an expression that represents all classes to be added.

Built-In Directives

- **ngRepeat**
  - ngRepeat directive instantiates a template once per item from a collection. Each template instance gets its own scope, where the given loop variable is set to the current collection item, and $index is set to the item index or key
- **ngClassEven**
  - Works exactly as ngClass, except they work in conjunction with ngRepeat and take effect only on even row elements i.e. 0,2,4,6 …
- **ngClassOdd**
  - Works exactly as ngClass, except they work in conjunction with ngRepeat and take effect only on odd row elements i.e. 1,3,5,7…

Built-In Directives

- **ngSrc**
  - Angular will tell the browser not to fetch the image via the given URL until all expressions provided to ng-src have been interpolated.
  - It is recommended to use in place of src.
- **ngHref**
  - Angular waits for the interpolation to take place and then activates the link's behavior.
  - It is recommended to use in place of href.
- **ngDisabled, ngChecked, ngReadonly & ngSelected**
  - Those directives works with HTML boolean attributes.

Built-In Directives

- **ngNonBindable**
  - ngNonBindable directive tells Angular not to compile or bind the contents of the current DOM element.
  - It will be useful to display Angular Code snippets
- **ngIf**
  - ng-if directive is used to completely remove or recreate an element in the DOM based on an expression. If the expression assigned to ng-if evaluates to a false value, then the element is removed from the DOM, otherwise a clone of the element is reinserted into the DOM.
  - Using ng-if when an element is removed from the DOM, its associated scope is destroyed. when it comes back into being, a new scope is created

Built-In Directives

ngModel

- ngModel directive provides the two-way data-binding by synchronizing the model to the view, as well as view to the model.
- ngModel will try to bind to the property given by evaluating the expression on the current scope. If the property doesn't already exist on this scope, it will be created implicitly and added to the scope.
- ngModel is responsible for:
  - Binding the view into the model, which other directives such as input, textarea or select.
  - Providing validation behavior (i.e. required, number, email, url).
  - Keeping the state of the control (valid/invalid, dirty/pristine, touched/untouched, validation errors).
  - Setting related css classes on the element (ng-valid, ng-invalid, ng-dirty, ng-pristine, ng-touched, ng-untouched) including animations.
  - Registering the control with its parent form.

Built-In Directives

- **ngSwitch**
  - The ngSwitch directive is used to conditionally swap DOM structure on your template based on a scope expression
  - It is used in conjunction with ng-switch-when and on="propertyName" to switch which directives render in our view when the given propertyName changes.
- **ngInit**
  - The ngInit directive is used to set up the state inside the scope of a directive when that directive is invoked.
  - It is a shortcut for using ng-bind without needing to create an element; therefore, it is most commonly used with inline text

Built-In Directives

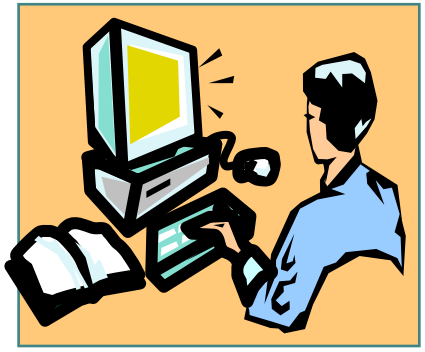
- **ngSwitch**
  - The ngSwitch directive is used to conditionally swap DOM structure on your template based on a scope expression
  - It is used in conjunction with ng-switch-when and on="propertyName" to switch which directives render in our view when the given propertyName changes.
- **ngInit**
  - The ngInit directive is used to set up the state inside the scope of a directive when that directive is invoked.
  - It is a shortcut for using ng-bind without needing to create an element; therefore, it is most commonly used with inline text

Built-In Directives

- **ngInclude**
  - ngInclude directive is used to fetch, compile and include an external HTML fragment into your current application.
  - By default, the template URL is restricted to the same domain and protocol as the application document. This is done by calling *$sce.getTrustedResourceUrl* on it
  - The URL of the template is restricted to the same domain and protocol as the application document unless white listed or wrapped as trusted values.
  - To access file locally in chrome use *chrome.exe -allow-file-access-from-files -disable-web-security*

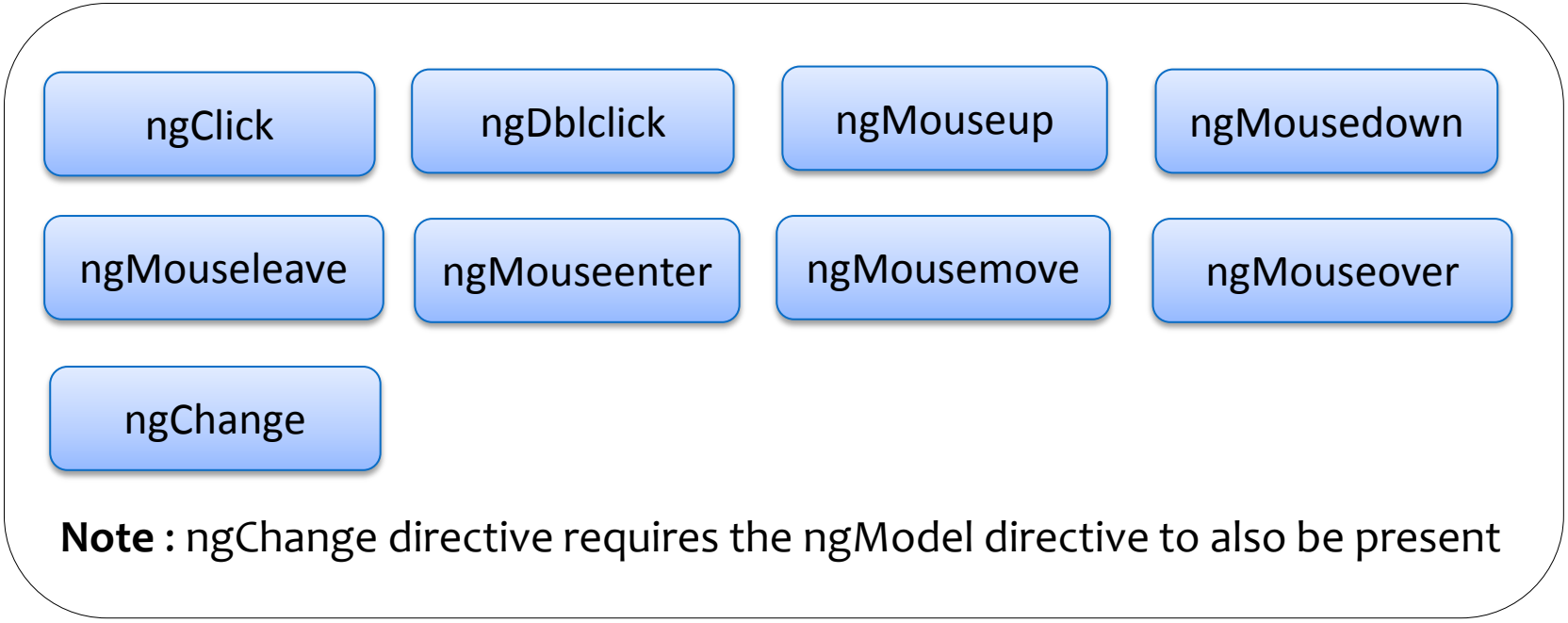
Demo

- **Built-InDirectives**



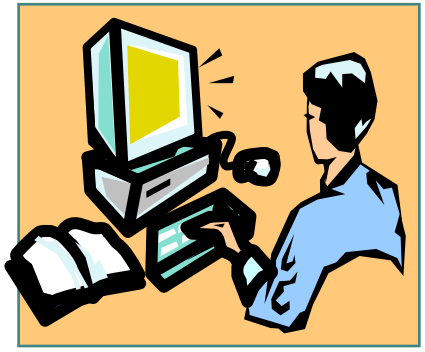
Built-In Event directives

- **When Angular parses the HTML, it look for directive and takes action based on that, when it looks for event directives it register the event on the DOM object.**



Demo

- **EventDirectives**

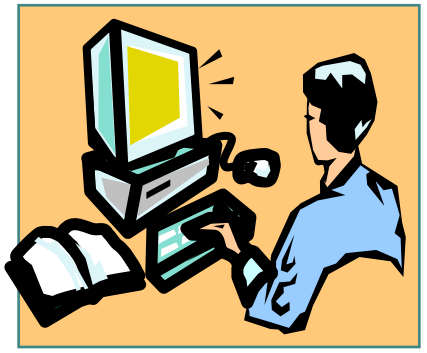


Form Validation

- **ngForm directive can be used, when we need to nest a form within another form, which is not allowed in normal HTML.**
- **The outer form is valid when all of the child forms are valid. It will be very useful when dynamically generating forms using ngRepeat directive. Angular will not submit the form to the server unless the form has an action attribute specified.**
- **The following CSS classes are set automatically, depending on the validity of the form:**
  - ng-valid when form is valid (is set if the form is valid)
  - ng-invalid when form is invalid
  - ng-pristine when form is pristine (the field has not been modified by user)
  - ng-dirty when form is dirty (the field has been modified by user)

Demo

- **FormValidation**

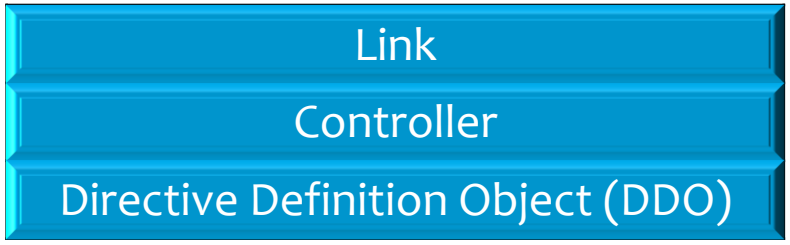


Custom Directives

- **Directives makes the Angular framework so powerful, we can also create our own directives. A directive is defined using the .directive() method on application's Angular module.**
- **Directives can be implemented in the following ways:**
  - Element directives : activated when AngularJS finds a matching HTML element in the HTML template
  - Attribute directives : activated when AngularJS finds a matching HTML element attribute
  - CSS class directives : activated when AngularJS finds a matching CSS Class
  - Comment directives : activated when AngularJS finds a matching HTML comment
- **AngularJS recommends to use element and attribute directives, and leave the CSS class and comment directives (unless absolutely necessary).**

Custom Directives

- **Directives consists of three (or less) important things**



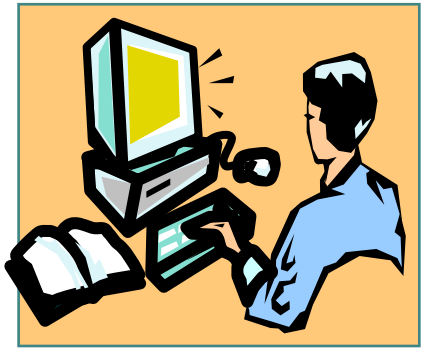
- **Link function is where the DOM manipulation occurs.**
- **Controller is constructed during the pre-linking phase and receives the $scope for the element.**
- **Directive Definition Object (DDO) tells the compiler how a Directive needs to be assembled. Common properties include the link function, controller function, restrict, template and templateUrl**

Applying restrictions to directives

- **Custom directive is restricted to attribute by default.**
- **In order to create directives that are triggered by element, class name & comment we need to use the restrict option.**
- **The restrict option is typically set to:**
  - 'A' - only matches attribute name
  - 'E' - only matches element name
  - 'C' - only matches class name
  - 'M' - only matches comment
- **These restrictions can also be combined**
  - 'AEC' - matches either attribute or element or class name

Demo

- **Directive01**



Custom Directives - template

- **template is an inline template specified using html as a string / function which gets appended / replaced (by setting replace:true) within the element where the directive was invoked.**
- **template has a scope that can be accessed using double curly markup, like {{ expression }}. Backslashes is used at the end of the each line to denote multi line string**
- **When a template string must be wrapped in a parent element. i.e, a root DOM element must exist.**

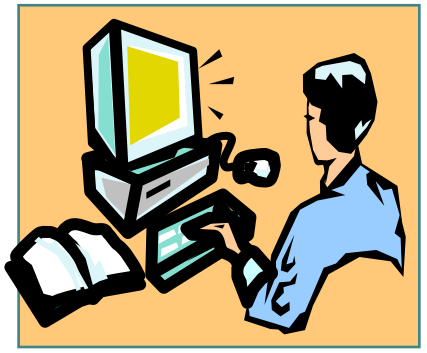
```
app.directive("helloworldattr",function(){
        return {
                replace:true,
                template: "<h1>Hello World Attribute from Directive</h1>"
         }
});
```

Custom Directives - templateUrl

- **templateUrl comes in handy when our template gets too big to inline.**
- **We can specify the path of an HTML file / a function which returns the path of an HTML file.**
- **By default, the HTML file will be requested on demand via Ajax when the directive is invoked.**
  - When developing locally, we should run a server in the background to serve up the local HTML templates from our file system. Failing to do so will raise a Cross Origin Request Script (CORS) error.
  - Start the chrome by typing **chrome.exe -allow-file-access-from-files -disable-web-security** in run(Win + R Key) to avoid CORS error, Mozilla Firefox wont give any errors. I.E doesn't support CORS
  - We can load templates directly into the cache in a script tag, or by consuming the $templateCache service directly.

Demo

- **Directive09**
- **Directive10**



Custom Directives – compile() & link function()

- **The compile() and link() functions define how the directive is to modify the HTML that matched the directive.**
- **When the directive is first compiled by AngularJS (first found in the HTML), the compile() function is called. The compile() function can then do any one-time configuration of the element needed.**
- **The compile() function finishes by returning the link() function. The link() function is called every time the element is to be bound to data in the $scope object.**
- **compile() function has to return the link() function when executed.**
- **We can even set only a link() function also for the custom directives.**

Custom Directives – compile() & link function()

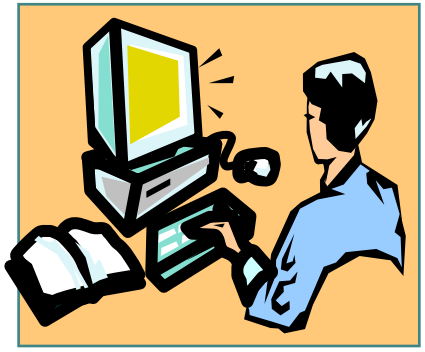
```
<div ng-controller="MyController" >
   <userinfo >This will be replaced</userinfo>
</div>

<script>
   var myapp = angular.module("myapp", []);
   myapp.directive('userinfo', function() {
      var directive = {};
      directive.restrict = 'E'; /* restrict this directive to elements */
      directive.compile = function(element, attributes) {
         element.css("border", "1px solid #cccccc");
         var linkFunction = function($scope, element, attributes) {
            element.html("This is the new content: " + $scope.firstName);
            element.css("background-color", "#ffff00");
         }
         return linkFunction;
      }
      return directive;
   });
   myapp.controller("MyController", function($scope, $http) {
                  $scope.firstName = "Karthik";
   });
</script>
```

Custom Directives – compile() & link function()

- **The template produced by a directive is meaningless unless it's compiled against the right scope. By default a directive does not get a new child scope. Rather, it gets the parent's scope. This means that if the directive is present inside a controller it will use that controller's scope. To utilize the scope, we can make use of a function called link.**
- **link takes a function with the following signature, function link(scope, element, attrs) { ... } where:**
  - scope is an Angular scope object.
  - element is the jqLite-wrapped element that this directive matches.
  - attrs is a hash object with key-value pairs of normalized attribute names and values.
- **The link function is mainly used for attaching event listeners to DOM elements, watching model properties for changes, and updating the DOM**

Demo

- **Directive02**

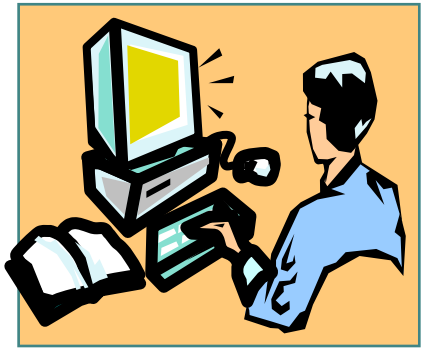


Custom Directives - controller Function and require

- **The controller function of a directive is used to establish the communication between two directives.**
- **controller function is used to create a UI component by combining two directives.**
- **require is used to inject the controller of the required directive as the fourth parameter of the current directive's linking function.**
- **require uses '^' prefix when the directive looks for the controller on its parents otherwise it looks for the controller on its own element.**

Demo

- **Directive04**



Custom Directives – Directive's Scope

- **By default a directive gets the parent's scope, so that they are free to modify the parent's controller scope properties.**
- **If directives need to add properties and functions for internal use there is no need to add it to the parent's scope.**
- **The scope can be configured with the scope property of the directive definition object**
  - A child scope – This scope prototypically inherits the parent's scope. (scope is set to true)
  - An isolated scope – A new scope that does not inherit from the parent and exists on its own. These bindings are specified by the attribute defined in HTML and the definition of the scope property in the directive definition object.

Custom Directives – Directive's Scope

```
<div ng-init="myProperty = 'Controller's Parent Scope'"></div>
 Surrounding scope: {{ myProperty }}
 <div my-inherit-scope-directive="SomeCtrl">
  Inside an directive with inherited scope: {{ myProperty }}
 </div>
 <div my-directive>
  Inside myDirective, isolate scope: {{ myProperty }}
 <div>
</div>
<script>
  angular.module('myApp', []).directive('myDirective', function() {
   return {
            scope: {}     //Isolated Scope
     };
  }).directive('myInheritScopeDirective', function() {
   return {
            scope: true   // child Scope
     };
  })
</script>
```

Custom Directives – Isolated Scope using '@ = &'

- **There are 3 types of binding options which are defined as prefixes in the scope property. The prefix is followed by the attribute name of HTML element.**
  - One Way Text Binding (Prefix: @)
  - Two-way Binding (Prefix: =)
  - Execute Functions in the Parent Scope(Prefix: &)
- **Text bindings are prefixed with @, and they are always strings. Whatever we write as attribute value, it will be parsed and returned as strings. If the parent scope changes, the isolated scope will reflect that change, but not the other way around.**
- **Two-way bindings are prefixed by = and can be of any type. Whenever the parent scope property changes, the corresponding isolated scope property also changes, and vice versa**

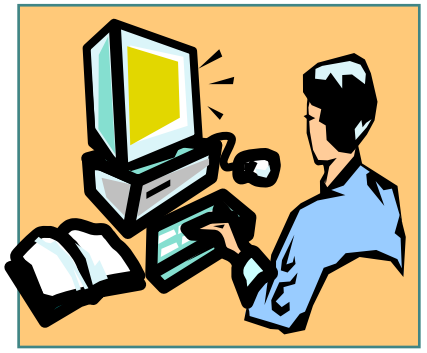
Custom Directives – Isolated Scope using '@ = &'

- **We can call a function on the parent scope from isolated scope using '&'**

```
<div ng-controller="MyCntrl">
  <div my-directive company="{{company}}"></div>
</div>

<script>
var app = angular.module("myApp",[]);
app.directive("myDirective", function(){
   return {
                restrict:"A",
                 scope: {
                          company:"@"
                  },
                 template: '{{company}}'
    }
});
app.controller("MyCntrl",function($scope){
          $scope.company = "IGATE";
});
</script>
```

Demo

- **Directive05**
- **Directive06**
- **Directive07**

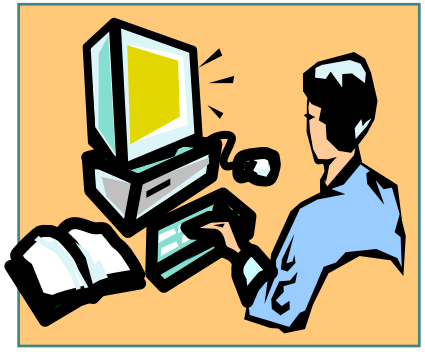


Custom Directives – Transclusion

- **Transclusion is a feature which lets us to wrap a directive around arbitrary content. We can later extract and compile it against the correct scope, and finally place it at the specified position in the directive template.**
- **Transclude allows us to pass in an entire template, including its scope, to a directive. Doing so gives us the opportunity to pass in arbitrary content and arbitrary scope to a directive. If the scope option is not set, then the scope available inside the directive will be applied to the template passed in.**
- **Transclusion is most often used for creating reusable widgets.**
- **ngTransclude directive marks the insertion point for the transcluded DOM of the nearest parent directive that uses transclusion.**
  - Any existing content of the element that this directive is placed on will be removed before the transcluded content is inserted.

Demo

- **Directive08**



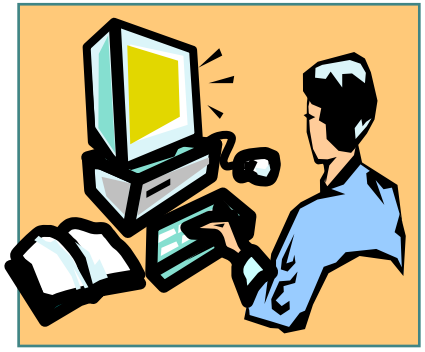
Working with jQuery UI

- **We can wrap the jQuery UI Datepicker into an Angular.js easily with the help of a custom directive**

```
<div>
        <input data-igate-date-picker="true" type="text"/>
</div>
<script>
var app = angular.module("myApp",[]);
app.directive('igateDatePicker',function(){
        return function(scope,element,attrs){
                element.datepicker({
                        changeMonth: true,
                        changeYear: true
                });
        }
});
</script>
```

Demo

- **jQueryUI-DatePicker**



Digest Cycle and $scope

- **Digest cycle can be considered as a loop, during which Angular checks if there are any changes occured to the variables watched being watched.**
- **Angular sets up a watcher on the scope model, which in turn updates the view whenever the model changes.**

```
$scope.$watch('modelVariable', function(newValue, oldValue) {
        //update the DOM with newValue
});
```

- **$digest cycle fires the watchers. When the $digest cycle starts, it fires each of the watchers. These watchers checks the current value of the scope model is different from old value. If there is a change, then the corresponding listener function executes.As a result any expressions in the view gets updated.**

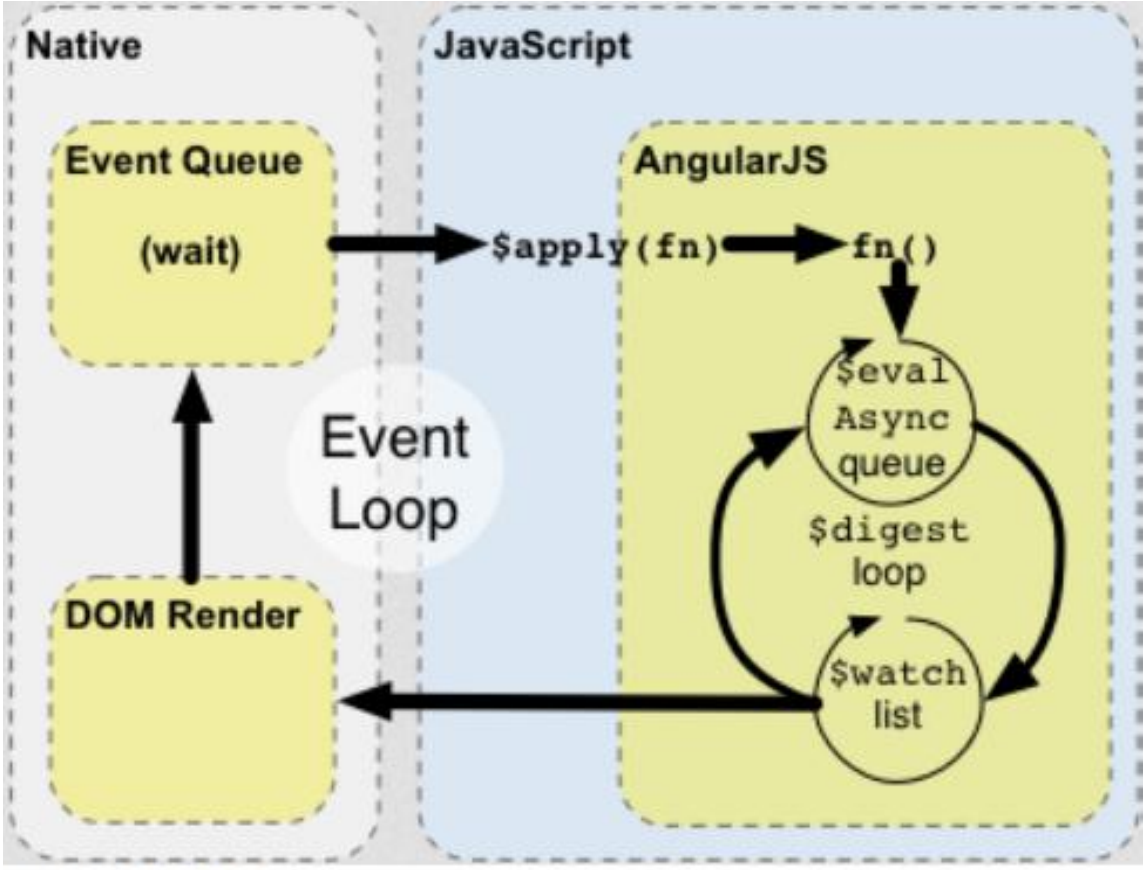
Digest Cycle and $scope

- **$digest cycle starts as a result of a call to $scope.$digest(). Angular doesn't directly call $digest(). Instead, it calls $scope.$apply(), which in turn calls $rootScope.$digest(). As a result of this, a digest cycle starts at the $rootScope, and subsequently visits all the child scopes calling the watchers along the way.**
- **AngularJS wraps the function calls (which updates the model) from view within $scope.$apply()**
- **The $apply() function comes in two flavors.**
  - The first one takes a function as an argument, evaluates it, and triggers a $digest cycle.
  - The second version does not take any arguments and just starts a $digest cycle.

Digest Cycle and $scope

- **Built-in directives/services (like ng-click, ng-repeat, ng-model, $timeout,$http etc) which changes the models automatically trigger a $digest cycle.i.e. It calls $apply() automatically and creates implicit watches to the model variables.**
- **If we change any model outside of the Angular context, then we need to inform Angular of the changes made by calling $apply() manually. For instance, if setTimeout() function updates a scope model, Angular wont have any idea about the model change then it becomes our responsibility to call $apply() manually, which in turn triggers a $digest cycle.**
- **If a directive that sets up a DOM event listener and changes models inside the handler function, we need to call $apply() to ensure the changes take effect.**

Digest Cycle and $scope



Source : AngularJS.org

$watch

- **$watches can be used to watch any value, and trigger a function call when that value changes. A $watch can be set up from any $scope by calling $scope.$watch().**
- **There are two ways to set up a watch (no difference between two)**
  - By Expression.

```
$scope.$watch('modelVariable', function(newValue, oldValue) {
        //update the DOM with newValue
});
```

  - By Function

```
$scope.$watch( function() { return $scope.modelVariable; }, function(newValue, oldValue) {
        //update the DOM with newValue
} });
```

$digest

- **$digest loops through all watchers on the scope on which it is called and it's child scopes. It evaluates them and executing the handlers if any changes found.**
- **To call $digest**
  - $scope.$digest();

$apply

- **It's a wrapper around $rootScope.$digest that evaluates any expression passed to it prior to calling $digest().**
- **Different ways to calling $apply:**
  - $scope.$apply('modelVariable= "test"');
  - $scope.$apply(function(scope) {

    scope. modelVariable = 'test';

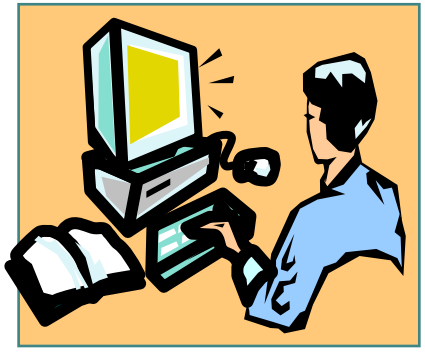
    });
  - $scope.$apply(function(){

    scope. modelVariable = 'test';

    });
  - $scope.$apply(); //Similar to $digest()

Demo

- **Directive03**



Summary

- scope is not a model actually it contains the model
- We can use JavaScript objects as model in angular.
- Double curly brace is the markup indicator for binding data to view
- ngSrc is used to bind and image's src. It delays fetching an image until binding has occured.
- angular directives can be written in three different ways : tag, attribute & class and we cannot write all the inbuilt directives in all the 3 ways.
- ngChange directive requires the ngModel directive to also be present

Summary

- ngCloak directive is used to avoid a flash of unbound html
- ngBind does not support multiple bindings where as ngBindTemplate supports multiple bindings
- ngClass directives has two companion directives : ngClassEven & ngClassOdd
- ngForm allow us to Nest forms
- Avoid custom tag name directives to make Angular support with older version of IE.
- Two way binding will update on every key stroke. Two way binding is supported by Input, Select and Textarea HTML elements

Summary

- Two way binding will update on every key stroke. Two way binding is supported by Input, Select and Textarea HTML elements
- The property will be created automatically, when we refer a property that doesn't exist in a ngModel directive.
- ngPattern directive allow us to create a regex for validation.
- form tag should have a name property to check the validity of form.
- restrict property of a directive can take the following values : E , A, C and M

Summary

- **template and templateUrl is used to specify the html which will be used by our directives.**
- **If we change any model outside of the Angular context, then we need to inform Angular of the changes made by calling $apply() manually.**