# Angular Componentization

Angular uses directives to encapsulate code that directly interacts with the DOM. This separation of concerns not only makes the code easier to maintain and test, but makes directives the basic building block of reusable UI components. Here’s a simple example:

<html ng-app=”angDemo”> <!-- this directive bootstraps the angDemo app -->

<body>

<page-title /> <!-- our custom component -->

….

<script src=”scripts/app.js”></script>

<script src=”scripts/directives/demoDirectives.js”></script>

</body>

</html>

// demoDirectives.js

angular.module('demo.directives',[]) // the module name

.directive('pageTitle', function() { // angular removes ‘-‘ and camel cases the name

return {

restrict: 'E', // the directive can only be used as an element

replace: true, // replace the directive with the template

template: '<div class="span12"><h3><span>Angular Demo</span></h3></div>'

};

});

// app.js

// demo.directives module gets injected into the angDemo app module

angular.module('angDemo', ['demo.directives'])

.config(function ($routeProvider) {

$routeProvider

.when('/', {

templateUrl: 'views/main.html',

controller: 'MainCtrl'

})

.otherwise({

redirectTo: '/'

});

});

The above Angular directive function creates a component called, pageTitle and packages it into a module called, demo.directives. To use the pageTitle directive you simply inject the demo.directives into the angDemo application module.

There are several base properties that control the construction and behavior of the directive. Here’s the pseudo-code template of the directive:

.directive('namespaceDirectiveName', function factory(dependency-injectables) {

restrict: string,

priority: number,

template: string,

templateUrl: string,

replace: bool,

transclude: bool,

scope: bool or object,

controller: function ($scope, $element, $attrs, $transclude),

require: string, // require that another directive be present

link: function postLink(scope, iElement, iAttrs) {},

compile: function (tElement, tAttrs, transclude) {

return {

pre: function preLink(scope, iElement, iAttrs, controller) { ... },

post: function postLink(scope, iElement, iAttrs, controller) { ... }

}

}

});

| **Property** | **Purpose** |
| --- | --- |
| restrict | Declare how directive can be used in a template as an element, attribute, class, comment, or any combination. |
| priority | Set the order of execution in the template relative to other directives on the element. Higher numbers run first. |
| template | Specify an inline template as a string. Not used if you’re specifying your template as a URL. |
| templateUrl | Specify the template to be loaded by URL. This is not used if you’ve specified an inline template as a string. |
| replace | If true, replace the current element. If false or unspecified, append this directive to the current element. |
| transclude | Lets you move the original children of a directive to a location inside the new template. |
| scope | Create a new scope for this directive rather than inheriting the parent scope. |
| controller | Create a controller which publishes an API for communicating across directives. |
| require | Require that another directive be present for this directive to function correctly. |
| link | Programmatically modify resulting DOM element instances, add event listeners, and set up data binding. |
| compile | Programmatically modify the DOM template for features across copies of a directive, as when used in ng-repeat. Your compile function can also return link functions to modify the resulting element instances. |

Transclusion

Angular directives can also ‘mix-in’ elements of the parent document when it constructs the template. This is called transclusion. To use transclusion, I change my html file:

<page-title>Transcluded Angular Demo</page-title>

Then I update the directive to show where I want the transcluded elements:

.directive('pageTitle', function() {

return {

restrict: 'E',

replace: true,

transclude: true,

template: '<div class="span12"><h3><span ng-transclude></span></h3></div>'

};

Scope

You will often want to access a scope from your directive to watch model values and make UI updates when they change, and to notify Angular when external events cause the model to change. This is most common when you’re wrapping some non-Angular component from jQuery, Closure, or another library, or implementing simple DOM events.

The directive below creates an element that ‘toggles’ the display of its content when the user clicks on the title bar. The scope property below creates a new isolate scope for the directive but data binds the *expanderTitle* attribute in the parent scope to e*Title* in the new scope.

.directive('expander', function() {

return {

restrict: 'EA', // use as element or attribute

replace: true,

transclude: true,

scope: { eTitle: '=expanderTitle' },

template: '<div><div class="title" ng-click="toggle()">{{eTitle}}</div>' +

'<div class="body" ng-show="showMe" ng-transclude></div></div>',

link: function(scope, element, attrs) {

scope.showMe = false;

scope.toggle = function toggle() {

scope.showMe = !scope.showMe;

}

}

}

});

Component Controllers

Angular components will at times need to interact with other elements in the app. For instance, components may need to interact with the model or with other components in the same scope. One way of doing this is to use a controller. The directive below exposes a simple api that closes other expanders in the group when a new expander is opened.

.directive('expanderGroup', function() {

return {

restrict: 'E',

replace: true,

transclude: true,

template: '<div ng-transclude></div>',

controller: function() {

var expanders = [];

this.gotOpened = function(selectedExpander) {

angular.forEach(expanders, function(expander) {

if (selectedExpander != expander) {

expander.showMe = false;

}

});

}

this.addExpander = function(expander) {

expanders.push(expander);

}

}

}

})

The expander directive is changed to include the controller:

.directive('expander', function() {

return {

restrict: 'E',

replace: true,

transclude: true,

require: '^?expanderGroup',

scope: { mytitle: '=expanderTitle' },

template: '<div><div class="title" ng-click="toggle()">{{mytitle}}</div>' +

'<div class="body" ng-show="showMe" ng-transclude></div></div>',

link: function(scope, element, attrs, groupController) {

scope.showMe = false;

if(groupController){

groupController.addExpander(scope);

}

scope.toggle = function toggle() {

scope.showMe = !scope.showMe;

if(groupController) groupController.gotOpened(scope);

}

}

}

})