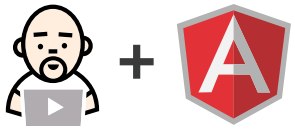

egghead.io presents

Directive Definition Object

Directive Definition Object

Object returned by directive declaration that provides instructions to the compiler.

```
// example myModule.
directive('directiveName',
  function () {
    return {
      restrict: 'E',
      scope: { myValue: '='},
      link: function(scope, el, attrs) {
        ...
      }
    };
  });
```

Attributes

priority

Specify the order in which directives are applied to the DOM. Directives are compiled in priority order, from highest to lowest. The *default* is 0.

terminal

If set to true then the current priority will be the last set of directives which will execute (any directives at the current priority will still execute as the order of execution on same priority is undefined).

scope

When true: then a new scope will be created for this directive. If multiple directives on the same element request a new scope, only one new scope is created. The new scope rule does not apply for the root of the template since the root of the template always gets a new scope.

When { } (object hash): a new *isolated* scope is created. This isolated scope **does not inherit from a parent scope.** Useful when creating reusable components, which should not accidentally read or modify data in the parent scope.

The isolate scope object defines the properties of the local scope. Used to pass data to the directive. The keys of the isolate scope object are the **property names**. The **values** define the relationship. The possible values are:

- `@` - binds to a **string** value.
- `=` - Bidirection binding to an *object*
- `&` - Binding to an *expression*

Optionally these can all be followed by an **attribute name**, for example `@localAlias`. If no attribute name is given, the name is assumed to be the same as the key.

controller

Controller function, or string name of the controller for the directive. The can be shared with other directives by using require attribute. The controller is provided the following injectable items:

```
myModule.directive('directiveName', function ( ) {
 return {
  scope: { },
  controller: 'SomeCtrl'
 };
});
```

- `$scope` - Current scope associated with the element
- `$element` - Current element
- `$attrs` - Current attributes object for the element
- `$transclude` - A transclude linking function pre-bound to the correct transclusion scope. The scope can be overridden by an optional first argument. function([scope], cloneLinkingFn).

Consider using a controller instead of large link function. This provides separation and enhanced testability.

require

Require another directive and inject its controller as the fourth argument to the linking function. Takes a string name (or array of strings) of the directive(s) to pass in. If an array is used, the injected argument will be an array in corresponding order.

The name can be prefixed with:

- (no prefix) - Locate the required controller on the current element. Throw an error if not found.
- `?` - Attempt to locate the required controller or pass null to the link fn if not found. Makes it *optional*
- `^` - Locate the required controller by searching the element's parents. Throw an error if not found.
- `?^` - Attempt to locate the required controller by searching the element's parents or pass null to the link fn if not found.

```
myModule.directive('directiveName',
 function ( ) {
  return {
   require: 'otherDirectiveWithCtrl',
   link: function(scope, el, attrs, ctrl) {
    ...
   }};
 } );
```

controllerAs

If using a controller function, you can use the 'controller as' syntax. Directive *must* define an isolated scope if this option is used. If using a string for the controller attribute, 'controller as' can be used.

restrict

Restricts the directive to a specific directive DOM declaration style. If omitted, the default (attributes only) is used.

- E - Element name: `<my-directive></my-directive>`
- A - Attribute (default): `<div my-directive="exp"></div>`
- C - Class: `<div class="my-directive: exp;"></div>`
- M - Comment: `<!-- directive: my-directive exp -->`

type

String representing the document type used by the markup. This is useful for templates where the root node is non-HTML content (such as SVG or MathML). The default value is "html".

- html - All root template nodes are HTML, and don't need to be wrapped. Root nodes may also be top-level elements such as `<svg>` or `<math>`.
- svg - The template contains only SVG content, and must be wrapped in an `<svg>` node prior to processing.
- math - The template contains only MathML content, and must be wrapped in an `<math>` node prior to processing.

If no type is specified, then the type is considered to be html.

template

replace the current element with the contents of the HTML string. Migrates classes/attributes from the replaced element.

You can specify `template` as a string representing

```
myModule.directive('directiveName',
 function ( ) {
  return {
   template: '<div>DOM in code!</div>'
  };
 } );
```

the template or as a function which takes two arguments `tElement` and `tAttrs` (described in the compile function api below) and returns a string value representing the template.

Consider using `templateUrl` instead of HTML strings in your javascript.

templateUrl

Same as template but the template is loaded from the specified URL. Because the template loading is

asynchronous the compilation/linking is suspended until the template is loaded.

transclude

compile the content of the element and make it available to the directive. Typically used with `ngTransclude`. The advantage of transclusion is that the linking function receives a transclusion function which is pre-bound to the correct scope. In a typical setup the widget creates an isolate scope, but the transclusion is not a child, but a sibling of the isolate scope.

The value can be:

```
myModule.directive('directiveName',
  function () {
    return {
      transclude: true,
      template: '<div ng-transclude>[...]
```

`true` - transclude the content of the directive. 'element' - transclude the whole element including any directives defined at lower priority.

compile

The compile function deals with transforming the template DOM. Since most directives do not do template transformation, it is not used often.

link

This property is used only if the `compile` property is not defined.

The link function is responsible for registering DOM listeners as well as updating the DOM. It is executed after the template has been cloned.

```
function link(scope, iElement, iAttrs,
controller, transcludeFn) { ... }
```

- `scope` - Scope - The scope to be used by the directive for registering watches.
- `iElement` - instance element - The element where the directive is to be used. It is safe to manipulate the children of the element only in `postLink` function since the children have already been linked.
- `iAttrs` - instance attributes - Normalized list of attributes declared on this element shared between all directive linking functions.
- `controller` - a controller instance - A controller instance if at least one directive on the element defines a controller. The controller is shared among all the directives, which allows the directives to use the controllers as a communication channel.
- `transcludeFn` - A transclude linking function pre-bound to the correct transclusion scope. The scope can be overridden by an optional first argument. This is the same as the `$transclude` parameter of directive `controllers. function([scope], cloneLinkingFn)`.

Notes