Angular Js

# Directives

## 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 or even transform the DOM element and its children.

e.g. inbuilt directives ngBind, ngModel, and ngClass.

What does it mean to "compile" an HTML template? For AngularJS, "compilation" means attaching event listeners 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.

Best Practice: Prefer using the dash-delimited format (e.g. ng-bind for ngBind). If you want to use an HTML validating tool, you can instead use the data-prefixed version (e.g. data-ng-bind for ngBind). The other forms shown above are accepted for legacy reasons but we advise you to avoid them.

## Directive types

In the following way directive can be implemented.

* attribute name
* tag name
* comments
* class name

Best Practice: Prefer using directives via tag name and attributes over comment and class names. Doing so generally makes it easier to determine what directives a given element matches.

## Creating 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.

Best Practice: Prefer using the definition object over returning a function.

**Best Practice:** In order to avoid collisions with some future standard, it's best to prefix your own directive names. For instance, if you created a <carousel> directive, it would be problematic if HTML7 introduced the same element. A two or three letter prefix (e.g.btfCarousel) works well. Similarly, do not prefix your own directives with ng or they might conflict with directives included in a future version of Angular.

Let’s take a simple example of directive.



This reproduces below output

Notice that we have bindings in this directive. After $compile compiles and links <div my-customer></div>, it will try to match directives on the element's children. This means you can compose directives of other directives.

**Note:** You do not currently have the ability to access scope variables from the templateUrl function, since the template is requested before the scope is initialized.

**When should I use an attribute versus an element?** Use an element when you are creating a component that is in control of the template. The common case for this is when you are creating a Domain-Specific Language for parts of your template. Use an attribute when you are decorating an existing element with new functionality.

## Isolating the Scope of a Directive

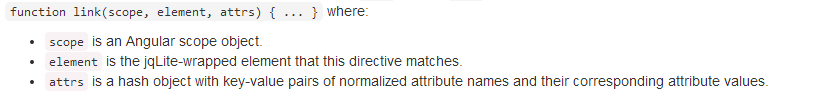
The simple example of the directive above is great but it has fatal flows. We can use this only once within the scope. To re-use it, we have to create a new scope (controller) every time.

To avoid this; we have to separate the directives inside scope with the outside scope, which is called as **‘isolated scope’.** To do this, we use directives “scope” option.



## Creating a Directive that Manipulates the DOM

The directive that manipulates the dom; uses **link** function of the directive. **Link** function has the following signature.



Just like module.controller API we use module.directive API, so that we can inject dependencies in the directive.

## Creating a Directive that Wraps Other Elements

We have seen that we can pass in models to the directives using an isolated scope. But sometimes we require passing the entire template rather than model or an object.

To do this, we need to use the transclude option.