

STAYING SHARP

*with
AngularJS*

Reusable Directives Level 4

External Libraries Section 3

Creating a Directive for 3rd-Party Plugins

What an ugly tooltip! Let's replace this tooltip with a prettier Bootstrap tooltip. We can create a directive to accomplish this!



Directives are markers on a **DOM element** that tell AngularJS's HTML compiler `$compile` to attach a specified behavior to that DOM element.

Directives

Markers on a **DOM element** that tell AngularJS's HTML compiler `$compile` to attach a specified behavior to that DOM element.

Including Bootstrap's tooltip.js

Before we create a directive to use a 3rd-party plugin, let's include the plugin.

📄 index.html

```
<script src="/js/vendor/bootstrap.js"></script>
```


Creating a Title Attribute Directive

Naming this directive 'title' will allow us to override the HTML title attribute and replace the hideous default tooltip. When creating a directive to override a default, do not give a namespace.

title.js

```
angular.module("NoteWrangler")
.directive("title", function() {
  return {
    restrict: "A",
    link: function(scope, element) {
      // Append Custom Tooltip here
    }
  };
});
```

`<div class="card" title="{{header}}">`

By default, directives are restricted to attribute 'A' type and are therefore redundant

Alternative Link Syntax in the Directive!

If a directive is only returning the link function, there is an alternative way to write it.

title.js

```
.directive("title", function() {  
  return {  
    link: function(scope, element) {  
  
    }  
  };  
});
```

The two became one

title.js

```
.directive("title", function() {  
  return function(scope, element) {  
  
  };  
});
```


📄 title.js

```
.directive("title", function() {  
    return function(scope, element) {  
  
    };  
});
```


Adding tooltip Code to link Function

Now we will call Bootstrap's .tooltip() method on our element inside link and pass it container:body.

title.js

```
angular.module("NoteWrangler")
.directive("title", function() {
  return function(scope, element) {
    element.tooltip({ container: "body" });
  };
});
```


Current Default tooltip

We already have a title on our cards to display their headers on hover in a tooltip.

📄 title.js

```
angular.module("NoteWrangler")
.directive("title", function() {
  return function(scope, element) {
    element.tooltip({ container: "body" });
  };
});
```


A Problem With Our New Tooltip

When we refresh, we get our new tooltip to replace the old one, but our header is no longer being interpolated ... What's happening here?

title.js

```
angular.module("NoteWrangler")
.directive("title", function() {
  return function(scope, element) {
    element.tooltip({ container: "body" });
  };
});
```

{{header}}

Markers on a **DOM element** that tell AngularJS's HTML compiler `$compile` to attach a specified behavior to that DOM element.

Because `link` is only run one time, it runs before Angular had the value for `{{header}}` and then the tooltip is stuck with `'{{header}}'` as its text for all time.

Adding the Attribute Directive to the Template

This is easily remedied by calling `$timeout`. This will cause Angular to run through an entire event loop before replacing our tooltip.

title.js

```
angular.module("NoteWrangler")
.directive("title", function() {
  return function(scope, element, $timeout) {
    $timeout(function(){
      element.tooltip({ container: "body" });
    });
  };
});
```

Directives

Markers on a **DOM element** that tell AngularJS's HTML compiler `$compile` to attach a specified behavior to that DOM element.

BEST PRACTICE

Directives Should Clean Up After Themselves

Our tooltip now works, but as a best practice, we need to clean up after our directive.

title.js

```
angular.module("NoteWrangler")
.directive("title", function() {
  return function(scope, element, $timeout) {
    ...
    scope.$on('$destroy', function() {
      element.tooltip('destroy');
    });
  };
});
```

A clean DOM is a happy DOM

Destroy method is called whenever the directive is removed (off hover) and its scope is destroyed.

STAYING SHARP with AngularJS

When to Use Controller/Link

Inside directives, it is best practice to use controller only when you want to share functions with other directives. All other times you should use link.

```
angular.module("NoteWrangler")
.directive("nwExample", function() {
  return {
    controller: function($scope) { },
    link: function(scope, element) { }
  };
});
```