Creating Custom AngularJS Directives Part 4 – Restriction and Transclusion

by Dan Wahlin

In Part 3 of this series I introduced how an isolate scope local property defined in a directive can be used to pass parameters to external functions (something that's a bit tricky at first glance but easy once you know the process). In this post I'll continue the series on building custom directives and discuss "transclusion" and how it can be used to add flexibility to directives.

Restriction in Directives

Directives can be defined in HTML as elements, as an attribute of an element, as a CSS class, or as a comment. How do you restrict how your custom directive can be used though?

To restrict how and where a directive can be used you use the **restrict** property. It accepts the following values:

Restriction	Description
E	Can be used as an element `<my-directive></my-directive>`

A	Can be used as an attribute that’s applied to an element `<div my-directive="exp"></div>`
C	Can be used as in a CSS class definition applied to an element (rare – I don’t recommend this) `<div class="my-directive: exp;"></div>`
M	Can be used in a comment (rare – I don’t recommend this) `<!-- directive: my-directive exp -->`

Here’s an example of restricting a directive to be used as either an element or an attribute. Notice that no separator is put between the values:

```
app.directive('myDirectiveWithRestriction', function () {
    return {
        restrict: 'EA',
        scope: {
            tasks: '='
        }
    };
});
```

Although the **C** and **M** values can be used, they're more rare. Most directives out there only use **E** and **A**.

Now that you've seen restriction, let's look at the concept of "transclusion" – everybody's favorite word!

Transclusion in Directives

The first time I heard the word "transclusion" I wondered if it was actually a real word (if you've seen my AngularJS in 60ish Minutes video on YouTube you'll know that's true). If you look it up in a Webster's dictionary you won't find anything but definitions can be found on Wikipedia and other sites. Most sites define "transclusion" in the following way:

The inclusion of a document or part of a document into another document by reference (see Wikipedia).

If you've ever loaded a CSS stylesheet or HTML template (an up and coming feature of HTML5) into an HTML page or loaded a header or footer HTML fragment into a server-side page (typically called a server-side include) then you've worked with a form of transclusion. Here's a short video that provides a quick overview of transclusion:

https://www.youtube.com/watch?v=HnYT3Wq2iS8

When applied to AngularJS directives, transclusion provides a way for a consumer of a directive to define a template that is imported into the directive and displayed. For example you might have a directive that outputs a table while allowing the consumer of the directive to control how the table rows are rendered. Or, you might have a directive that outputs an error message while allowing the consumer of the directive to supply HTML content that handles rendering the error using different colors. By supporting this type of functionality the consumer of the directive has more control over how specific parts of the HTML generated by the directive are rendered.

Two key features are provided by AngularJS to support transclusion. The first is a property that is used in directives named **transclude**. When a directive supports transclusion this property is set to **true**. The second is a directive named **ng-transclude** that is used to define where external content will be placed in a directive’s template. The directive code that follows uses these two AngularJS features to enable transclusion.

Here’s an example of a simple directive that handles collecting and storing tasks. It uses transclusion to allow the consumer of the directive to supply the HTML used to render each task. Looking through the code you can see that the directive’s **transclude** property is set to **true** and its **restrict** property is set to **E** so that it can only be used as an element. This allows the directive to import custom content defined by the consumer of the directive. Another property named **replace** is also included that handles replacing the directive’s element (for example **<isolated-scope-with-transclusion>** in this example) with the content defined in the template as the directive renders in the browser.

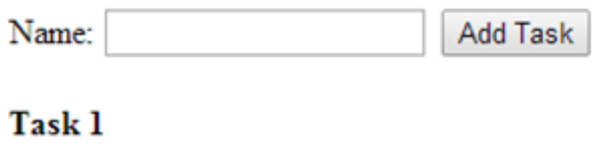
```
angular.module('directivesModule').directive(
  'isolatedScopeWithTransclusion', function () {

    return {

        restrict: 'E',

        transclude: true,

        replace: true,

        scope: {

            tasks: '='

        },
        controller: function ($scope) {
            $scope.addTask = function () {

                if (!$scope.tasks) $scope.tasks = [];

                $scope.tasks.push({
                    title: $scope.title
                });

            };
        },
        template: '<div>Name: <input type="text" ' +
                  'ng-model="title" />&nbsp;' +
                  '<button ng-click="addTask()">Add Task' +
                  '</button>' +
                  '<div class="taskContainer"><br />' +
                     '<ng-transclude></ng-transclude>' +
                  '</div></div>'
    };
});
```

Looking at the directive's template you can see that the **ng-transclude** directive is defined inside of a **<div>**element. Any content defined by the consumer of the directive will be placed where the **ng-transclude** directive is defined. Here's an example of using the directive and supplying content that will be imported/transcluded into the directive:

```
<isolated-scope-with-transclusion tasks="tasks">
```

```
    <div ng-repeat="task in tasks track by $index">
        <strong>{{ task.title }}</strong>
    </div>
</isolated-scope-with-transclusion>
```

In this example the consumer of the directive can pass in initial tasks to display (if any exist) by using the **tasks**local scope property. The code that handles iterating through the tasks is then placed inside of the **<isolated-scope-with-transclusion>** directive element. The custom HTML content will be placed where **ng-transclude** is defined within the directive's template. Assuming a single task is passed into the directive the output would look like the following when it first loads. As the end user adds additional tasks they'll be added to the list automatically.



Task 1

This example was kept quite simple so that we could focus on transclusion and understand how it works. But, the general concept shown here can certainly be extended to accommodate more advanced scenarios where HTML fragments need to be imported into a directive's template. Several 3rd party directives such as the UI Bootstrap project's Alert and Accordion directives rely on transclusion.

Conclusion

Transclusion is quite simple once you understand what the term means and know how directives can take advantage of it (although it can certainly get more advanced). By using **transclude** and **ng-transclude** within a directive you can provide customization capabilities within your custom directives.

Check out my AngularJS Custom Directives course for additional information about building your own directives.