# AngularJS Components

When creating a module that will be associated with an HTML document (as opposed to organizing code, which I describe shortly), the convention is to give the module a name with the suffix App. In the example, I used the name **exampleApp** for my module, and the benefit of this convention is that it makes it clear which module represents the top-level AngularJS application in your code structure—something that can be useful in complex apps that can contain multiple modules.

Defining a module in JavaScript is only part of the process; the module must also be applied to the HTML content using the **ng-app** attribute. When AngularJS is the only web framework being used, the convention is to apply the ng-app attribute to the **html** element. The **ng-app** attribute is used during the *bootstrap* phase of the AngularJS life cycle.

...

<html ng-app="exampleApp">

...



Controllers act as a conduit between the model and the views. Controllers are defined using the Module.controller method, which takes two arguments: the name of the controller and a Constructor function.

Applying Controllers to Views is done through the ng-controller attribute.

...

**<div class="panel" ng-controller="dayCtrl">**

...

## **directive**s

...

myApp.directive("highlight", function () {

return **function (scope, element, attrs) {**

**if (scope.day == attrs["highlight"]) {**

**element.css("color", "red");**

**}**

}

});

...

The return statement in the factory function returns another function, which AngularJS will invoke each time it needs to apply the directive, and this is the *worker function*. AngularJS will call the factory function when it wants to set up the building block and then calls the worker function when it needs to apply the building block

## Filters

The filter method is used to define a filter, and the arguments are the name of the new filter and a factory function that will create the filter when invoked. Filters are themselves functions, which receive a data value and format it so it can be displayed.

...

return function (input) {

**return angular.isNumber(input) ? dayNames[input] : input;**

};

...

...

<h4 ng-controller="dayCtrl" highlight="Monday">

**Today is {{day || "(unknown)" | dayName}}**

</h4>

...

Filters are applied after JavaScript expressions are evaluated, which allows me to use the || operator to check for null values and then the | operator to apply the filter.

Directives use the $filter service to get access to all of the filters that have benn defined, including custom filters.

...

myApp.directive("highlight", function **($filter**) {

**var dayFilter = $filter("dayName");**

return function (scope, element, attrs) {

if (**dayFilter(scope.day)** == attrs["highlight"]) {

element.css("color", "red");

}

}

});

...

## Services

Services are singleton objects that provide any functionality that you want to use throughout an application. There are some useful built-in services that come with AngularJS for common tasks. Some key AngularJS are delivered as services, including the $scope and $filter objects.

Three of the methods defined by the Module object are used to create services in different ways: service, factory, and provider.

The service method takes two arguments: the name of the service and a constructor function that is called to create the service object. When AngularJS calls the constructor function, it assigns a new object.

myApp.service("**days**", function () {

this.today = new Date().getDay();

this.tomorrow = this.today + 1;

});

...

myApp.controller("tomorrowCtrl", function ($scope, **days**) {

$scope.day = **days**.tomorrow;

});

...

## Values

The Module.value method lets you create services that return fixed values and objects. AngularJS assumes any argument to a factory function declares a dependency that it needs to resolve.

...

var now = new Date();

myApp.service("days", function (**now**) {

this.today = now.getDay();

this.tomorrow = this.today + 1;

});

...

you will see an error like this one in the browser JavaScript console:

Error: Unknown provider: nowProvider <- now <- days

The problem here is that AngularJS won’t use the local variable as the value for the **now** parameter when it calls the factory function, and the now variable will no longer be in scope when it is required.

# Using Modules to Organize Code

Any AngularJS module can rely on components defined in other modules. One common convention is to organize your application into modules that have the same type of component and to make it clear which building block a module contains by using the main module’s name and appending the block type, which is why it’s called exampleApp.Controllers.

The content of the controllers.js File

var controllersModule = angular.module("**exampleApp.Controllers**", [])

controllersModule.controller("dayCtrl", function ($scope, days) {

$scope.day = days.today;

});

controllersModule.controller("tomorrowCtrl", function ($scope, days) {

$scope.day = days.tomorrow;

});

**var myApp = angular.module("exampleApp", ["exampleApp.Controllers", "exampleApp.Filters", "exampleApp.Services", "exampleApp.Directives"]);**

# Module Life Cycle

myApp.**constant**("startTime", new Date().toLocaleTimeString());

myApp.**config**(function (startTime) {

console.log("Main module config: " + startTime);

});

myApp.run(function (startTime) {

console.log("Main module run: " + startTime);

});

…

angular.module("exampleApp.Services", [])

…

.config(function() {

console.log("Services module config: (no time)");

})

.run(function (startTime) {

console.log("Services module run: " + startTime);

});

The **constant** method is similar to the **value** method but creates a service that can be declared as a dependency by the **config** method (which you can’t do when you create values).

The config method accepts a function that is invoked after the module is loaded. *The config method is used to configure a module, usually by injecting values that have been obtained from the server, such as connection details or user credentials.*

The run method also accepts a function, but it will be invoked only when all of the modules have been loaded and their dependencies have been resolved.

Here is the sequence in which the callback functions are invoked:

1. The config callback on the **exampleApp.Services** module
2. The config callback on the **exampleApp** module
3. The run callback on the **exampleApp.Services** module
4. The run callback on the **exampleApp** module

# Data Binding Directives und Template Service

## Data Binding Directives

Data binding uses values from the model and inserts them into the HTML document.



### One-Way Bindings

One-way binding, means a value is taken from the data model and inserted into an HTML element. When the value is changed in the data model, the HTML element will be updated to display the new value.

The **ng-bind** directive is responsible for creating one-way data bindingsbindings, but it is rarely used directly because AngularJS will also create this kind of binding whenever it encounters the **{{** and **}}** characters in the HTML document.

<div>There are {{todos.length}} items</div>

<div>There are <span ng-bind="todos.length"></span> items</div>

<div **ng-bind-template="First: {{todos[0].action}}. Second: {{todos[1].action}}"**></div>

<div ng-non-bindable>

AngularJS uses {{ and }} characters for templates

</div>



### Two-Way Data Bindings

Two-way bindings are created with the ng-model directive.