Recipes with Angular.js

Practical concepts and techniques for rapid web application development

```
<body ng-app>
  <div ng-controller="MyCtrl">
    <button class="btn" ng-click="toggle()">Toggle</button>
    <p ng-show="isVisible()">Hello World!</p>
    <p>Debug Scope: visible = {{visible}}</p>
  </div>
</body>
```

by Frederik Dietz

beta version

Recipes with Angular.js

Practical concepts and techniques for rapid web application development

Frederik Dietz

Tweet This Book!

Please help Frederik Dietz by spreading the word about this book on Twitter!

The suggested hashtag for this book is #recipeswithangularjs.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search/#recipeswithangularjs

Contents

An Introduction to Angular.js

Including the Angular.js library Code in an HTML page

Problem

You want to use Angular.js on a web page.

Solution

In order to get your first Angular.js app up and running you need to include the Angular Javascript file via `script` tag and make use of the `ng-app` directive.

```
<html>
  <head>
    <script src="http://ajax.googleapis.com/ajax/libs/
      angularjs/1.0.4/angular.js">
    </script>
  </head>
  <body ng-app>
    <p>This is your first angular expression: {{ 1 + 2 }}</p>
  </body>
</html>
```

Tip: You can checkout a complete example on github[1].

Discussion

Adding the `ng-app` directive tells Angular to kick in its magic. The expression `{{ 1 + 2 }}` is evaluated by Angular and the result `3` is rendered. Note, that removing `ng-app` will result in the browser to render the expression as is instead of evaluating it. Play around with the expression! You can for example concatenate Strings and invert or combine Boolean values.

For Brevity reasons we skip the boilerplate code in the following recipes.

Binding a Text Input to an Expression

Problem

You want user input to be used in another part of your HTML page.

Solution

Use Angulars `ng-model` directive to bind the text input to the expression.

```
Enter your name: <input type="text" ng-model="name"></input>
<p>Hello {{name}}!</p>
```

Discussion

Assigning "name" to the `ng-model` attribute and using the name variable in an expression will keep both in sync automatically. Typing in the text input will automatically reflect these changes in the paragraph element below.

Consider how you would implement this traditionally using jQuery:

```
<html>
  <head>
    <script src="http://code.jquery.com/jquery.min.js"></script>
  </head>
  <body>
    Enter your name: <input type="text"></input>
    <p id="name"></p>

    <script>
      $(document).ready(function() {
        $("input").keypress(function() {
          $("#name").text($(this).val());
        });
      });
    </script>

  </body>
</html>
```

On document ready we bind to the keypress event in the text input and replace the text in the paragraph in the callback function. Using jQuery you need to deal with document ready callbacks, element selection, event binding and the context of this. Quite a lot of concepts to swallow and lines of code to maintain!

Responding to Click Events using Controllers

Problem

You want to hide an HTML element on button click.

Solution

Use the `ng-hide` directive in conjunction with a controller to change the visibility status on button click.

```
<html>
  <head>
    <script src="js/angular.js"></script>
    <script src="js/app.js"></script>
    <link rel="stylesheet" href="css/bootstrap.css">
  </head>
  <body ng-app>
    <div ng-controller="MyCtrl">
      <button ng-click="toggle()">Toggle</button>
      <p ng-show="visible">Hello World!</p>
    </div>
  </body>
</html>
```

And the controller in `js/app.js`:

```
function MyCtrl($scope) {
  $scope.visible = true;

  $scope.toggle = function() {
    $scope.visible = !$scope.visible;
  };
}
```

When toggling the button the “Hello World” paragraph will change its visibility

Discussion

Using the `ng-controller` directive we bind the `div` element including its children to the context of the `MyCtrl` Controller. The `ng-click` directive will call the `toggle()` function of our controller on button click. Note, that the `ng-show` directive is bound to the `visible` scope variable and will toggle the paragraph's visibility accordingly.

The controller implementation defaults the `visible` attribute to true and toggles its boolean state in the `toggle` function. Both the `visible` variable and the `toggle` function are defined on the `$scope` service which is passed to all controller functions automatically via dependency injection.

The next chapter will go into all the details of controllers in Angular. For now let us quickly discuss the MVVM (Model-View-ViewModel) pattern as used by Angular. In the MVVM pattern the model is plain javascript, the view is the HTML template and the ViewModel is the glue between the template and the model. The ViewModel makes Angular's two way binding possible where changes in the model or the template are in sync automatically.

In our example the `visible` attribute is the model, but it could be of course much more complex as for example retrieving data from a backend service. The controller is used to define the scope which represents the ViewModel. It interacts with the HTML template by binding the scope variable `visible` and the function `toggle()` to it.

Converting Expression Output with Filters

Problem

When presenting data to the user, you might need to convert the data to a more user-friendly format. In our case we want to uppercase the `name` value from the previous recipe in the expression.

Solution

Use the `uppercase` Angular filter.

```
1 Enter your name: <input type="text" ng-model="name"></input>
2 <p>Hello {{name | uppercase }}!</p>
```

Discussion

Angular uses the `|` (pipe) character to combine filters with variables in expressions. When evaluating the expression, the name variable is passed to the uppercase filter. This is similar to working with the Unix bash pipe symbol where an input can be transformed by another program. Also try the `lowercase` filter!

Creating Custom HTML elements with Directives

Problem

You want to render an HTML snippet but hide it conditionally.

Solution

Create a custom Directive which renders your Hello World snippet.

```
<body ng-app="MyApp">
  <label for="checkbox">
    <input id="checkbox" type="checkbox" ng-model="visible">Toggle me
  </label>
  <div show="visible">
    <p>Hello World</p>
  </div>
</body>
```

The `show` attribute is our directive, with the following implementation:

```
var app = angular.module("MyApp", []);

app.directive("show", function() {
  return {
    link: function(scope, element, attributes) {
      scope.$watch(attributes.show, function(value){
        element.css('display', value ? '' : 'none');
      });
    }
  };
});
```

The browser will only show the “Hello World” paragraph if the checkbox is checked and will hide it otherwise.

Discussion

Let us ignore the `app` module creation for now to look further into in a later chapter. In our example it is just the means to create a directive. Note, that the `show` String is name of the directive which corresponds to the `show` HTML attribute used in the template.

The directive implementation returns a `link` function which defines the directive's behaviour. It gets passed the `scope`, the HTML `element` and the HTML `attributes`. Since we passed the `visible` variable as attribute to our `show` directive we can access it directly via `attributes.show`. But, since we want to respond to `visible` variable changes automatically we have to use the `$watch` service, which provides us a function callback whenever the value changes. In this callback we change the CSS `display` property to either blank or `none` to hide the HTML element.

This was just a small glimpse of what can be achieved with directives and there is a whole chapter dedicated to them which goes into much more detail.