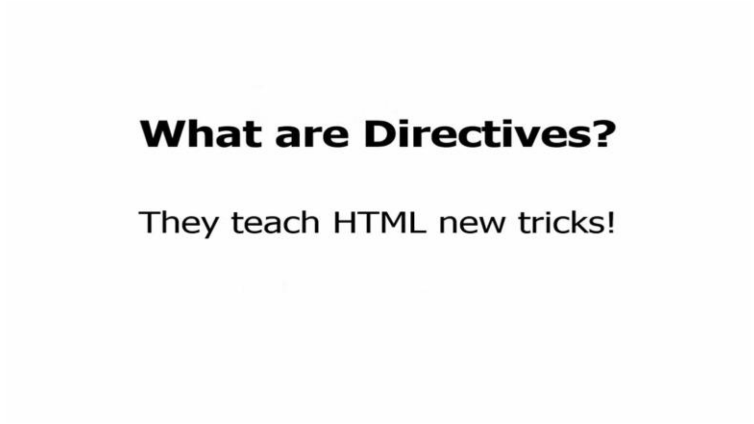
Directives, Filters and Data Binding

ANGULARJS
by Google

Directives, Filters and Data Binding

Once you've added the *AngularJS* script into a page now you're ready to start using it and the first thing we're going to talk about is something called **Directives**.

They're very, very critical and a kind of core concept in the AngularJS framework.



What are Directives?

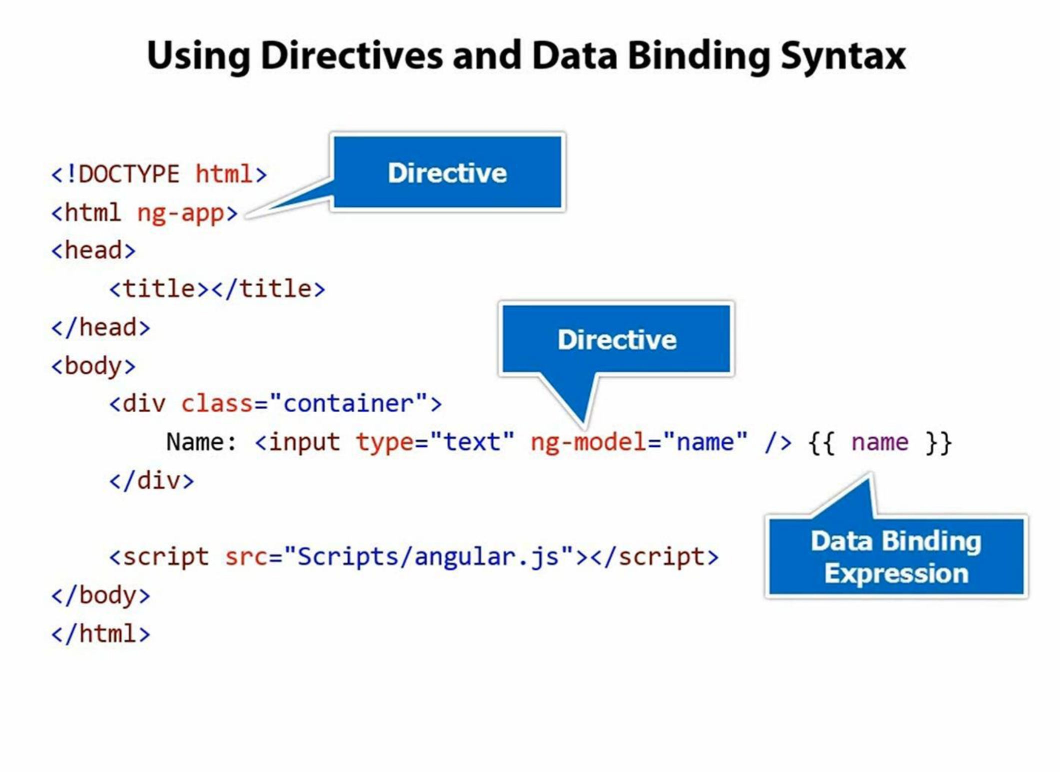
They teach HTML new tricks!

To start off, what is a directive?
Well I mentioned this earlier. A directive is really a way to teach HTML new tricks.

The web when it first came out was really just designed to display static pages. As we all know it's become very dynamic and we've dealt with that pretty well.

jQuery came out many years ago and it provided a way to do it. Even before then we could use raw, vanilla JavaScript.

Angular takes it up a whole notch and allows us to extend HTML very easily by simply adding attributes, elements or comments.



```
<!DOCTYPE html>
<html ng-app>
<head>
    <title></title>
</head>
<body>
    <div class="container">
        Name: <input type="text" ng-model="name" /> {{ name }}
    </div>

    <script src="Scripts/angular.js"></script>
</body>
</html>
```

Hello World – the AngularJS example

```
<html>
<head>
<script
src="http://ajax.googleapis.com/ajax/libs/angularjs/1.0.7/angular.js">
</script>
</head>
<body  ng-app ng-init="name = 'World'">
```

```
<h1>Hello, {{name}}!</h1>
</body>
</html>
```

Including the AngularJS library is not enough to have a running example. We need to bootstrap our mini application. The easiest way of doing so is by using the custom ng-app HTML attribute.

Closer inspection of the <body> tag reveals another non-standard HTML attribute: ng-init. We can use ng-init to initialize model before a template gets rendered.

The last bit to cover is the {{name}} expression which simply renders model value.

There is almost nothing special about the <input> HTML tag apart from the additional ng-model attribute.

The real magic happens as soon as we begin to type text into the <input> field. All of a sudden the screen gets repainted after each keystroke, to reflect the provided name!

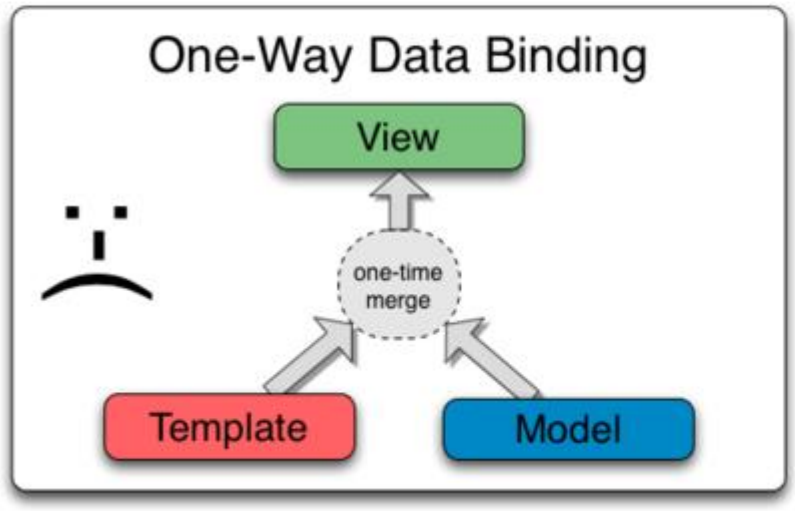
There is no need to write any code that would refresh a template, and we are not obliged to use any framework API calls to update the model. AngularJS is smart enough to detect model changes and update the DOM accordingly.

Most of the traditional templating system renders templates in a linear, one-way process: a model (variables) and a template are combined together to produce a resulting markup.

Any change to the model requires re-evaluation of a template. AngularJS is different because any view changes triggered by a user are immediately reflected in the model,
and any changes in the model are instantly propagated to a template.

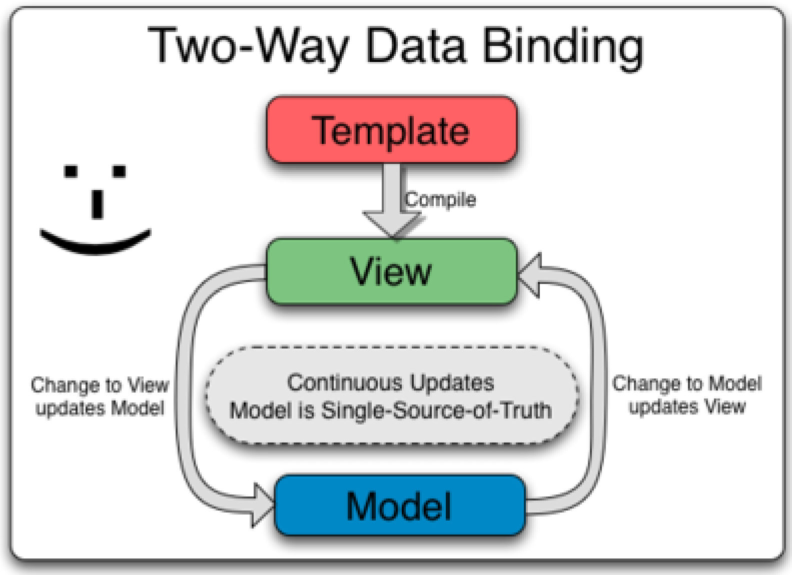
Data-binding in Angular apps is the automatic synchronization of data between the model and view components. The way that Angular implements data-binding lets you treat the model as the single-source-of-truth in your application. The view is a projection of the model at all times. When the modelchanges, the view reflects the change, and vice versa.

Data Binding in Classical Template Systems



Most templating systems bind data in only one direction: they merge template and model components together into a view. After the merge occurs, changes to the model or related sections of the view are NOT automatically reflected in the view. Worse, any changes that the user makes to the view are not reflected in the model. This means that the developer has to write code that constantly syncs the view with the model and the model with the view.

Data Binding in Angular Templates


Two-Way Data Binding
Template
Compile
View
Change to View updates Model
Continuous Updates Model is Single-Source-of-Truth
Change to Model updates View
Model

Angular templates work differently. First the template (which is the uncompiled HTML along with any additional markup or directives) is compiled on the browser. The compilation step produces a live view. Any changes to the view are immediately reflected in the model, and any changes in the model are propagated to the view. The model is the single-source-of-truth for the application state, greatly simplifying the programming model for the developer. You can think of the view as simply an instant projection of your model.

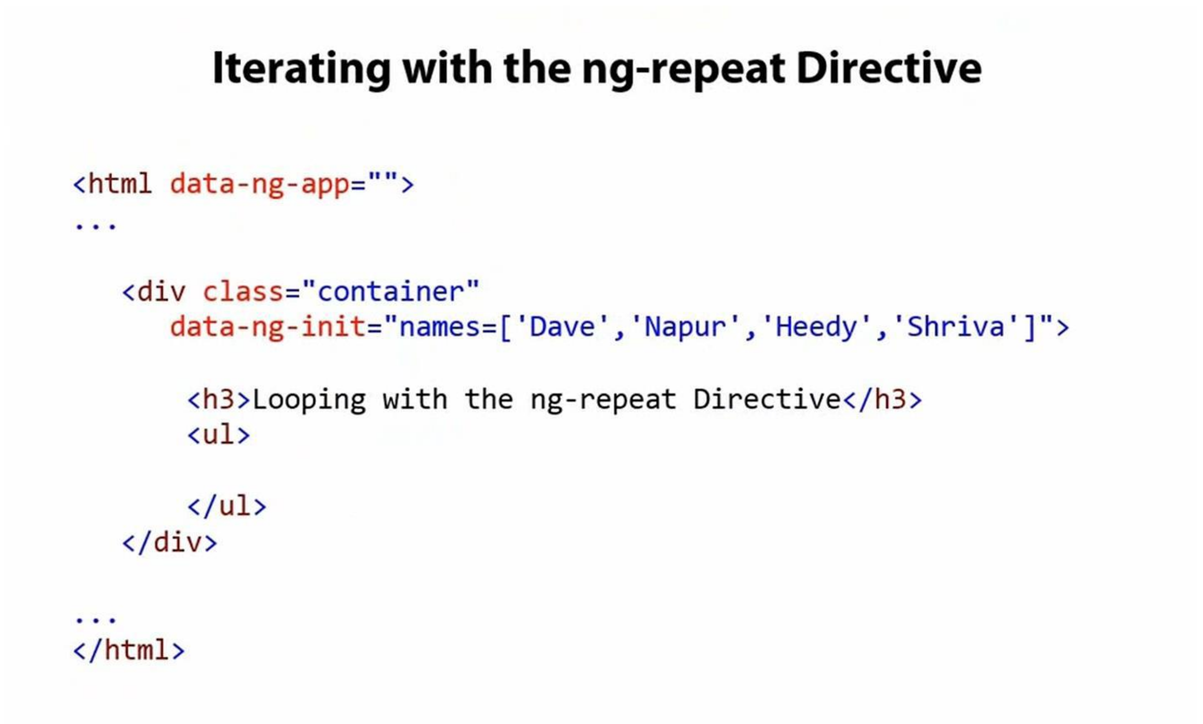
Because the view is just a projection of the model, the controller is completely separated from the view and unaware of it. This makes testing a snap because it is easy to test your controller in isolation without the view and the related DOM/browser dependency.

index.html

```
<div ng-app="">
        {{1 + 1}}
        {{"john" + "lindquist"}}
        {{3 * 3}}
      </div>
```

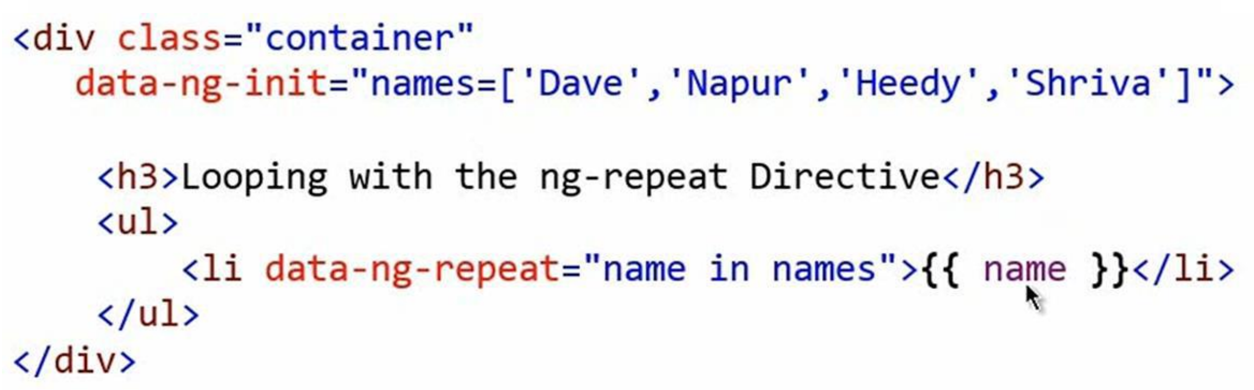
You should keep logic in the view to a minimum. Binding is most useful when you create an input or a way for the user to interact with the site of your app.

Iterating with the ng-repeat Directive



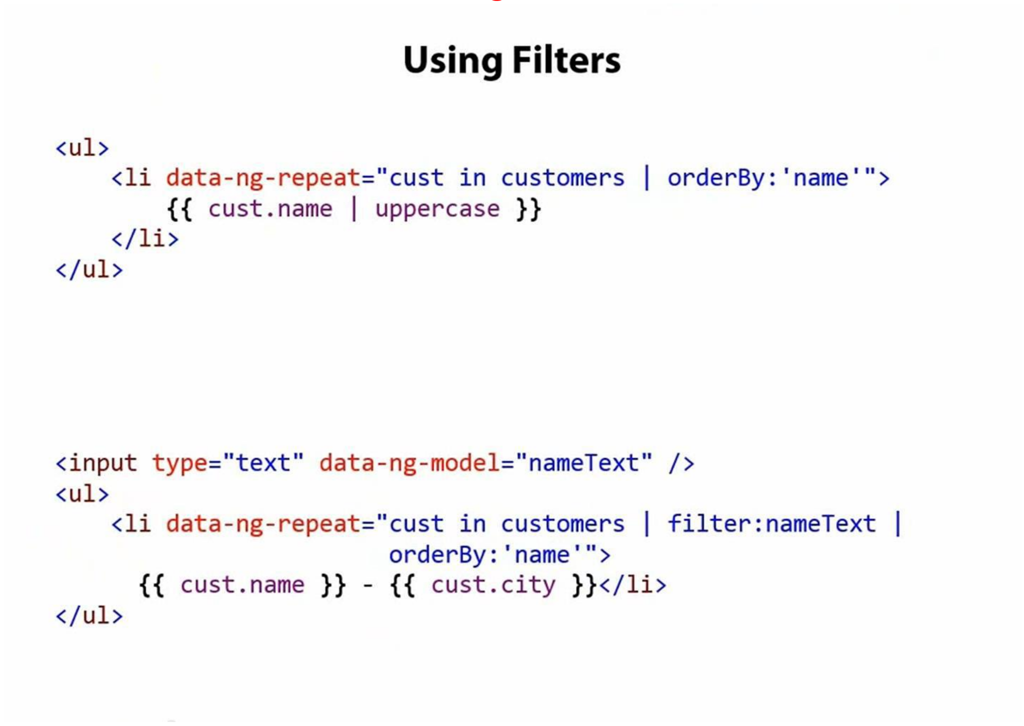
Iterating with the ng-repeat Directive

```
<html data-ng-app="">
...

   <div class="container"
      data-ng-init="names=['Dave','Napur','Heedy','Shriva']">

        <h3>Looping with the ng-repeat Directive</h3>
        <ul>

        </ul>
   </div>

...
</html>
```



```
<div class="container"
   data-ng-init="names=['Dave','Napur','Heedy','Shriva']">

    <h3>Looping with the ng-repeat Directive</h3>
    <ul>
        <li data-ng-repeat="name in names">{{ name }}</li>
    </ul>
</div>
```

Using Filters



Using Filters

```
<ul>
    <li data-ng-repeat="cust in customers | orderBy:'name'">
        {{ cust.name | uppercase }}
    </li>
</ul>
```

```
<input type="text" data-ng-model="nameText" />
<ul>
    <li data-ng-repeat="cust in customers | filter:nameText |
                        orderBy:'name'">
      {{ cust.name }} - {{ cust.city }}</li>
</ul>
```