**Udemy - Learn and Understand AngularJS**

1. N/A
2. N/A
3. N/A
4. N/A
   1. We have data on one side and HTML on the other, and we want to connect those two things. We want whatever happens on one to affect the other and vice versa, automatically.
   2. Model = our data. The thing that defines our data. Could be a variable in the case of JavaScript
   3. The view is the thing that people see, whatever people interact with (html, or really the DOM)
   4. In angular, the model and view are bound. Whatever happens in the view affect the model and vice versa.
   5. We call AngularJS MV\* (model view whatever)
5. HTML Aside: Custom Attributes
   1. When you see ng-, that’s really a custom attribute. AngularJS is using the fact that the browser DOM (memory) has that available to make decisions based on what it sees
6. JavaScript Aside – The Global Namespace
   1. When a webpage runs, we now have a piece of memory on our computer that’s dedicated to that webpage, where we can put our javascript objects and variables and do things.
   2. Variables in java by default go into the global namespace
   3. Don’t want to pollute the global namespace
   4. When you build an angularJS, there are certain elements/structural concepts that have the goal of not polluting the global namespace
7. N/A
8. Modules, Apps, and Controllers (The structure of an AngularJS application)
   1. With Angular, we’re only going to put one variable into the global namespace. That variable is our app.
   2. Angular comes with an angular object, and that object has an **angular.module** function. That function takes a name (the name of your app) and an array of dependencies.
   3. ng-app=”myApp”: angularJS looks for this attribute, and everything inside of where this is found is now connected to the myApp variable in the global namespace. And from here, that means that the HTML where the ng-app=”myApp” is embedded is the view for the myApp code
   4. So with ng-app angular looks for an angular module that has the name that matches the value you give to ng-app=””
   5. Now we add everything else to the myApp variable. Everything else will be underneath that object so it doesn’t pollute the global namespace and can take advantage of the things modules can do
   6. We can do myApp.controller because myApp is an angular module.
   7. The .controller(‘controllername’, function()) method takes a name and a function
   8. The function is the place to put the code associated with the controller.
9. JavaScript Aside – Dependency Injection
   1. Rather than creating an object inside a function, you pass it to a function
   2. AngularJS uses DI with regard to controllers and some other things.
10. The Scope Service
    1. A big part of what binds the model to the view = **scope**. It’s an object from something called the scope service
    2. All the code in the angularJS source code defines something call $scope. $scope is an object.
    3. Note: all angularJS services start with a $
    4. We can add variables and functions to the scope. The idea is that the scope becomes that middle piece, that piece between the view and the controller. You have some data, and the scope defines the data that will go back and forth between your model and your view.
11. JavaScript Aside – Functions and Strings
    1. You can take a function in javaScript and turn it into a string. This means that it’s possible to get the string representation of a function, parse it, and then make a decision based on what you find. This is what Angular does.
12. How Does Angular Do Dependency Injection?
13. Getting Other Services
    1. We can get any service just by passing them as parameters to our controller, thanks to Angular’s DI
    2. The second parameter in the angular.module() function is a list of dependencies – a list of modules that our myApp module uses
14. JavaScript Aside – Arrays and Functions
    1. In JavaScript, you can mix different types of things in the same array.
    2. You can also put a function inside an array as an element of an array.
    3. So, you could theoretically have an entire array of functions.
15. Dependency Injection and Minification
    1. Minification = shrinking the size of files for faster download.
    2. Method you want to use to create a controller when your code will be minified:
       1. The second parameter of your .controller function is now an array. The last element of your array is the function that defines the controller, and the elements of the array that come before should be whatever parameters get passed to the function (as strings).
16. Scope and Interpolation
    1. Interpolation = Creating a string by combining strings and placeholders
    2. We can attach variable names to the scope. We can put data into the scope and functions as well.
    3. Whatever’s sitting in the scope becomes available to the view that the controller is attached to
    4. When we use {{ something }}, angularJS is going to look at the scope object for something
    5. When a browser downloads HTML, it’s in a pre-modified state. angularJS hasn’t done anything to it, so if you look at the HTML, the {{ }} are still there. Once it gets downloaded, AngularJS runs and updates the text value within {{ }} in the DOM (in the browser’s memory)
17. Directives and Two-Way Data Binding
    1. **Directive** = An instruction to AngularJS to manipulate a piece of the DOM
    2. The ng-model directive says that we want to bind an element to a specific variable in the scope. So if we say an input element gets the ng-model=”handle” attribute, then the input element is now the view for the scope element handle (the model). And this is two-way binding –updating the variable changes the input field, and updating the input field changes the variable.
18. JavaScript Aside – the Event Loop
    1. Elements on an HTML page throw events, i.e. when something happens they say ‘this event just occurred’. JavaScript has something called the event loop, which basically means it sits there and waits for events to get thrown, and it throws events, all in a continuing cycle.
    2. With JavaScript of JQuery, you’re manually attaching code to events, waiting for them to occur and doing things. AngularJS takes advantage of those events to keep track of things for you.
19. Watchers and the Digest Loop
    1. We’ve already seen that JavaScript has an event loop – it constantly listens to events on the page (assuming we have an event listener) . If you’re using JQuery or regular JavaScript, you manually add event listeners, but AngularJS adds eventListeners for you. And, it’s extending the event loop in order to automatically control the binding between the model and view.
    2. On top of the built-in event loop, we have the Angular Context.
    3. When you add a variable to the scope & connect it to something in the view, Angular adds a Watcher to the Angular Context. There’s a watchlist and every time you put a variable or function that is on the scope on the page (in the HTML) Angular automatically adds a watcher to the watchlist. This means it’s keeping track of the old value and the new value, anytime something happens that might have affected the value (like a keypress event or a click). It has this list of things it’s going to watch to see if they changed.
    4. This watching happens in what’s called the Digest Loop. When you enter into the digest cycle, it goes through every variable in the watch list and sees if something has changed. If something has, it updates everything that’s connected to it. Then, it runs one more cycle to see if changing that thing changed something else. It continues doing this until all the new values match all the old values. Then it stops and waits for the JS event loop to say that another event has happened.
    5. $scope.$apply(function() {}) <- this tells angular that when this code runs, be sure to start your digest cycle. This can solve weird bugs where you have threading/asynchronous problems.
    6. For the most part, angular wraps everything you’re doing in $scope.$apply for you. But when you do certain things, that can cause a problem and you need to manually do this.
20. Common Directives
    1. Ng-if is a **directive**, so it’s telling the DOM what to do under certain circumstances. It removes or allows to exist entire pieces of the DOM based on a javascript expression.
    2. Ng-show appears to act similarly, but instead of removing elements from the DOM, it just adds/changes the class of the element that Ng-show’s applied to, and that class has a CSS hide rule applied to it.
    3. Ng-class takes a
    4. object (or JSON?)
    5. Ng-repeat is equivalent to (for type x : data structure) in java
21. Common Directives (Part 2)
    1. Ng-click takes a function that you have on the scope in response to clicks
    2. Ng-cloak hides an element in the DOM until JavaScript works on it (in case the user has a really slow computer and might see the raw HTML before js kicks in)
22. JavaScript Aside – The XMLHTTPRequest Object
    1. The XMLHttpRequest object is native to each browser – each one implements it differently.
    2. Idea: object that goes out and can make requests on its own via code and then do something with the result, as opposed to the browser making a request.
    3. Use JSON.parse() to parse text that we know is JSON.
23. External Data and $http
    1. The $http service wraps some of the trickier code in the XMLHttpRequest object.
    2. $http.get(‘/api’).success(fuction(result)) runs when the http object goes and gets data from the api successfully, it will call the success method and run the function you give it. And the result that you give to that function will be the data you get back.
    3. You can also attach the .error method: .error(function (data, status) { console.log(data);}
    4. $http.post() takes two parameters: first is where you’re sending data to, and the second is what you’re sending. And you send it as a JSON object
    5. As with get, you can append .success to this to determine what you do in the event of a success
24. Angular Aside – Multiple Controllers, Multiple Views
    1. AngularJS gives us a different instance of scope each time we ask for it in a controller. So each scope object is unique to each controller/view pairing
25. HTML and JavaScript Aside – Single page apps and the hash
    1. Window.addEventListener(‘hashchange’, function() {}); fires when the hash changes; when the fragment identifier changes while you’re on the page.
26. Routing, Templates, and Controllers (Single Page Applications)
    1. Once we have the correct script in our html file, we pass ‘ngRoute’ into our module as a parameter and then do myApp.config(function ($routeProvider) {});
    2. $routeProvider lets us specify routes, i.e. what should I do when I see a specific route in the hash?
    3. It will also match patterns.
    4. We do routeProvider.when(‘/’{templateURL: ‘templatelocation’, controller: mainController}).when()…
    5. ng-view says, whenever we’re doing single-page applications, and we say in the route to get a specific template, that template goes wherever you put the ng-view tag
    6. Also, the templates are bound to their respective views and have access to their scope objects
27. Routing, Templates, and Controllers (Part 2)
    1. .when(‘/second/:num’,) <- this is pattern matching. Says I’m going to look for /second/something, and then when I see that something, we can inject $routeParams into our controllers, and whatever the something was in your route, it will appear as a property in $routeParams.something

**Lynda – Angular JS 2 Essential Training**

* Angular is built on components.
* The starting point of an angular app is the bootstrapping on the parent component.
* Much like the HMTL DOM tree, angular runs on a component tree model.
* After angular loads the first component with the bootstrap call, it looks within that component’s HTML view and sees if it has any nested components.
* If so, it finds matches and runs the appropriate component code on those. This repeats down the tree.
* A component in Angular is used to render a portion of HTML, and provide functionality to that portion. It does this through a component class, in which you can define application logic for that component. For example, you can have a MediaItemComponent that can have a property named mediaItem that represents the data for a media item. That class could also have a method called onDeleteclick that can handle raising the delete mediaItem event.

1. **Architecture Overview**

* In angular, a component is actually a directive with a template. Directives provide functionality and can transform the DOM. There are two types of directives, structural and attribute.
* **Structural directives** modify layout by altering elements in the DOM.
* **Attribute directives** change the behavior or appearance of an existing DOM element. Since directives don’t have a template, they are something you can create with the intent of applying them to an existing element. Or in some cases, a template element to change that element in some way.
* Like a component, a directive gets configured with a selector that Angular will use to find a match and apply the directive.
* You’ll apply a directive in different ways. You can write an attribute on an element that matches your selector, or you can use the template syntax to add a directive in an assignment statement.
* A pipe takes in data, like a string or an array, and runs some logic to transform it.
* The most common way of displaying data in a view template is via interpolation, where you use a set of curly braces around a component property to tell angular to render the content of that property.
* You can also use directives to help display data.
* Directives give you the power to add client-side logic to your views.
* Dependency injection = you architect code where you provide modules the other modules they need to work, instead of modules going out and finding other modules.
* We typically refer to a class that we’ve written to encapsulate some logic as a service. These are where we put our application and business logic
* If you’re only concerned with storing data for the time in which the user is using your application, you could store that data in memory. You can create a Javascript class or object to store your data, provided to your app as something that can be injected in and then do constructor injection where needed to bring in the instance of that object.
* If you're looking to work with data stored in browser storage, such as local storage, you will find a need to write your own Javascript code to do so, and then use it with the services pattern, leveraging Angular's dependency injection to work with it throughout your application.
* If you are looking to work with data from a server via an API, Angular has some built-in framework stuff to help you with that. One way you can process data to and from an API is by leveraging the HTTP protocol.
* You can do this in two ways. One, by using the XML HTTP request or XHR, and the other by using JSONP. Angular provides an HTTP module in the framework for abstracting out, working with the way XHR and JSONP calls are done via client script. So you can do things like make getting post calls that work with JSON data as simple as passing a URL and a Javascript object to an Angular HTTP function, and subscribing to the results.

1. **Components**

* Angular leverages decorators to help configure ES2015 classes. A **decorator** is an expression that evaluates to a function that makes it possible to annotate that makes it possible to annotate and modify classes at design time.
* You use @Component() to tell angular you intend for a class to be a component
* @ sign, then decorator name, then parentheses with arguments
* The directive property takes in an array of types and is used to let Angular know that the component is going to contain components or directives in its template that we would like to be processed.
* **Interpolation** is a way to get data displayed in the view.
* You do interpolation by using a pair of matching curly braces in the markup. The contents of the double curly braces should be JavaScript expression that Angular will evaluate and then convert to a string.

**AngularJs tutorial – What is AngularJS**

* The ng-app directive is a starting point of angularJS application. The angular framework will check for this framework somewhere in the page then angular will bootstrap itself and start to manage the section of the page where ng-app is found

**AngularJS official site tutorial**

In Angular, the **View** is a projection of the model through the HTML **template**. This means that whenever the model changes, Angular refreshes the appropriate binding points, which updates the view.