Angular.js

AngularJS is a JavaScript framework that helps you build single-page applications. It is a MVW (Model-View-Whatever) framework, which means that it provides a way to separate your application's data, presentation, and behavior. It is a powerful and flexible framework that can be used to build a wide variety of web applications. It is a good choice for building single-page applications, as it provides a number of features that make this type of application development easier.

For this guide, we'll be covering the following topics of Angular.js:

Core Concepts

- Data binding
- Expressions
- Interpolation
- Directives
- Views and routes
- Filters
- HTML compiler
- Forms

Application Structure

- Modules
- Controllers
- Services
- Factories
- Providers
- Scopes
- Dependency injection

Other Features

- Events

- DOM manipulation
- AJAX
- Internationalization
- Unit testing
- End-to-end testing

Core Concepts

Data binding

Data binding is the process of connecting data from the model to the view. AngularJS provides two-way data binding, which means that changes to the model are automatically reflected in the view, and vice versa.

Here is an example of data binding:

```
<input type="text" ng-model="name">
```

In this example, the value of the `name` input is bound to the `name` variable in the controller. When the user types in the input, the value of the `name` variable is updated. And when the value of the `name` variable is updated, the text in the input is updated.

Expressions

Expressions are JavaScript code that can be used to evaluate values and perform actions. Expressions can be used in data bindings, directives, and filters.

Here is an example of an expression:

```
{{ name | uppercase }}
```

In this example, the `name` variable is evaluated and the result is converted to uppercase. The result is then displayed in the view.

Interpolation

Interpolation is a way to insert expressions into the HTML of a view. Interpolation is used to display the results of expressions in the view.

Here is an example of interpolation:

```
<h1>Welcome, {{ name }}</h1>
```

In this example, the `name` variable is interpolated into the HTML of the `h1` tag. The result is a heading that says "Welcome, [name]."

Directives

Directives are custom HTML attributes that can be used to add functionality to the view. Directives can be used to create custom components, add validation, and more.

Here is an example of a directive:

```
app.directive('myDirective', function() {
  return {
    link: function(scope, element, attrs) {
      // Do something when the directive is used
    }
  };
});
```

In this example, the `myDirective` directive is defined. The directive has a `link` function that is called when the directive is used. The `link` function can be used to add functionality to the view.

Views and routes

Views are the individual pages of an AngularJS application. Routes are used to navigate between views.

Here is an example of a route:

```
app.config(['$routeProvider', function($routeProvider) {
  $routeProvider.when('/', {
    templateUrl: 'home.html'
  });
  $routeProvider.when('/about', {
    templateUrl: 'about.html'
  });
});
```

In this example, two routes are defined: `/` and `/about`. The `/` route is associated with the `home.html` template, and the `/about` route is

associated with the `about.html` template.

Filters

Filters are functions that can be used to format data. Filters can be used to convert values to different formats, apply conditional logic, and more.

Here is an example of a filter:

```
app.filter('uppercase', function() {
  return function(value) {
    return value.toUpperCase();
  };
});
```

In this example, the `uppercase` filter is defined. The `uppercase` filter can be used to convert a value to uppercase.

HTML compiler

The HTML compiler is responsible for compiling AngularJS templates into JavaScript code. The HTML compiler is used to create a dependency graph between the different components of an AngularJS application.

Forms

Forms are used to collect user input. AngularJS provides a number of features for working with forms, such as validation, data binding, and custom controls.

Here is an example of a form:

```
<form ng-submit="submit()">
  <input type="text" ng-model="name">
  <button type="submit">Submit</button>
</form>
```

In this example, a form is created with an input field and a submit button. The `name` variable is bound to the input field. When the user clicks on the submit button, the `submit()` function in the controller is called.

Application Structure

Modules

AngularJS applications are organized into modules. A module is a collection of AngularJS code that defines a set of related functionality. Modules are loaded and unloaded dynamically, which makes AngularJS applications more flexible and scalable.

To create a module, you need to create a file with the .js extension and add the following code to it:

```
angular.module('myApp', []);
```

The `angular.module()` function takes two arguments: the name of the module and an array of dependencies. The name of the module is used to refer to the module in your code. The dependencies are other modules that the current module needs to function.

> Once you have created a module, you can start adding controllers, services, and other AngularJS code to it.

Controllers

Controllers are AngularJS objects that control the behavior of a portion of your application. Controllers are responsible for handling user input, updating the DOM, and communicating with services.

To create a controller, you need to create a JavaScript function with the `controller()` function. The `controller()` function takes two arguments: the name of the controller and an object that contains the controller's properties and methods.

The following code creates a controller named `MyController`:

```
app.controller('MyController', function($scope) {
  $scope.message = 'Hello, world!';
});
```

The `$scope` object is an AngularJS object that provides access to the current scope. The `message` property on the `$scope` object is used to store the message that will be displayed to the user.

Services

Services are AngularJS objects that provide reusable functionality. Services can be used to perform tasks such as accessing data from a server, validating user input, and formatting data.

To create a service, you need to create a JavaScript function with the `service()` function. The `service()` function takes two arguments: the name of the service and an object that contains the service's properties and methods.

The following code creates a service named `MyService`:

```
app.service('MyService', function($http) {
  this.getData = function() {
    return $http.get('/data');
  };
});
```

The `$http` object is an AngularJS object that provides access to the HTTP service. The `getData()` method on the `MyService` object is used to get data from the server.

Factories

Factories are AngularJS objects that are used to create other AngularJS objects. Factories are often used to create services.

To create a factory, you need to create a JavaScript function with the `factory()` function. The `factory()` function takes two arguments: the name of the factory and an object that contains the factory's properties and methods.

The following code creates a factory named `MyFactory`:

```
app.factory('MyFactory', function($http) {
  function MyService() {
```

```
    // ...
  }
  return MyService;
});
```

The `$http` object is an AngularJS object that provides access to the HTTP service. The `MyService` function is used to create a `MyService` object.

Providers

Providers are AngularJS objects that provide configuration and dependency injection services. Providers are often used to configure services and to inject dependencies into controllers.

To create a provider, you need to create a JavaScript function with the `provider()` function. The `provider()` function takes three arguments: the name of the provider, an object that contains the provider's properties, and a function that is used to initialize the provider.

The following code creates a provider named `MyProvider`:

```
app.provider('MyProvider', function() {
  this.myProperty = 'Hello, world!';

  function MyInitializer() {
    // ...
  }

  this.initialize = MyInitializer;
});
```

The `myProperty` property on the `MyProvider` object is used to store a message that will be displayed to the user. The `MyInitializer` function is used to initialize the `MyProvider` object.

Scopes

Scopes are AngularJS objects that are used to store data and to define the behavior of directives. Scopes are created by controllers and are passed down to directives.

Scopes have a number of properties and methods that can be used to store data, to define the behavior of directives, and to communicate with other scopes.

Dependency Injection

Dependency injection is a design pattern that allows you to decouple the code that needs a dependency from the code that provides the dependency. This makes your code more flexible and easier to test.

To use dependency injection in AngularJS, you need to use the `inject` annotation. The `inject` annotation tells AngularJS how to inject a dependency into a controller or service.

The following code shows how to use the `inject` annotation to inject a service into a controller:

```
app.controller('MyController', function($scope, MyService) {
  // ...
});
```

The `MyService` parameter is a dependency that is injected into the `MyController` controller.

Dependency injection is a powerful tool that can help you to write more flexible and easier to test AngularJS applications.

Here is a summary of the benefits of using dependency injection in AngularJS:

- **Flexibility:** Dependency injection makes your code more flexible because it allows you to change the implementation of a dependency without having to change the code that uses the dependency.
- **Easier testing:** Dependency injection makes your code easier to test because it allows you to mock or stub out dependencies in your unit tests.
- **Reusability:** Dependency injection makes your code more reusable because it allows you to create reusable components that can be injected into other applications.

Other Features

Events

AngularJS provides a way to bind events to DOM elements. This allows you to react to user interactions, such as clicks, mouseenters, and keystrokes.

To bind an event, you use the `ng-click`, `ng-mouseenter`, or `ng-keyup` directives. For example, the following code will bind a click event to the `submit` button and call the `submitForm()` function when the button is clicked:

```
<button ng-click="submitForm()">Submit</button>
```

DOM Manipulation

AngularJS provides a way to manipulate the DOM. This allows you to create dynamic and interactive user interfaces.

To manipulate the DOM, you use the `ng-repeat`, `ng-show`, and `ng-hide` directives. For example, the following code will repeat the `item` array in the DOM and hide the `error` element if the `isValid` variable is true:

```
<div ng-repeat="item in items">
  {{item.name}}
</div>
<div ng-show="isValid">
  <p>The form is valid.</p>
</div>
<div ng-hide="isValid">
  <p>The form is not valid.</p>
</div>
```

AJAX

AngularJS provides a way to make AJAX requests. This allows you to fetch data from a server without reloading the entire page.

To make an AJAX request, you use the `$http` service. For example, the following code will make an AJAX request to the `/api/users` endpoint and then display the results in the `users` div:

```
$http.get('/api/users').then(function(response) {
  $scope.users = response.data;
});
```

Internationalization

AngularJS provides a way to internationalize your application. This allows you to support multiple languages and locales.

To internationalize your application, you use the `ng-translate` directive. For example, the following code will translate the `Hello, world!` string to the user's preferred language:

```
<p ng-translate="hello">Hello, world!</p>
```

Unit Testing

AngularJS provides a way to unit test your application. This allows you to ensure that your code is working correctly before you deploy it to production.

To unit test your application, you use the Jasmine testing framework. For example, the following code will create a unit test for the `submitForm()` function:

```
describe('submitForm', function() {
  it('should submit the form', function() {
    // Arrange
    var scope = $scope.$new();
    var form = angular.element('<form ng-submit="submitForm()">');

    // Act
    form.submit();

    // Assert
    expect(scope.formSubmitted).toBe(true);
```

```
  });
});
```

End-to-End Testing

AngularJS provides a way to end-to-end test your application. This allows you to test your application from the user's perspective.

To end-to-end test your application, you use the Protractor testing framework. For example, the following code will create an end-to-end test for the `submitForm()` function:

```
describe('submitForm', function() {
  it('should submit the form', function() {
    // Arrange
    browser.get('http://localhost:8080');
    var form = element(by.css('form'));

    // Act
    form.submit();

    // Assert
    expect(element(by.css('p.success')).isPresent()).toBe(true);
  });
});
```