Web Technology

Unit V-
Client and Server Side Frameworks

Outline

- Angular JS
- Node JS
- Struts

Angular JS

Outline

- Angular JS : Overview,
- MVC architecture,
- directives, expression, controllers,
- filters,
- tables,
- modules,
- forms,
- includes,
- views,
- scopes,
- services,
- dependency injection,
- custom directives,
- Internationalization

AngularJS Introduction

- AngularJS is a **JavaScript framework**.
- It can be added to an HTML page with a <script> tag.
- AngularJS extends HTML attributes with **Directives**, and binds data to HTML with **Expressions**.

- **Syntax**

```
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.4/angular.min.js">
</script>
```

AngularJS Introduction

- **AngularJS Extends HTML**
- AngularJS extends HTML with **ng-directives**.
- The **ng-app** directive defines an AngularJS application.
- The **ng-model** directive binds the value of HTML controls (input, select, textarea) to application data.
- The **ng-bind** directive binds application data to the HTML view.

Outline

- Angular JS : Overview,
- MVC architecture,
- directives, expression, controllers,
- filters,
- tables,
- modules,
- forms,
- includes,
- views,
- scopes,
- services,
- dependency injection,
- custom directives,
- Internationalization

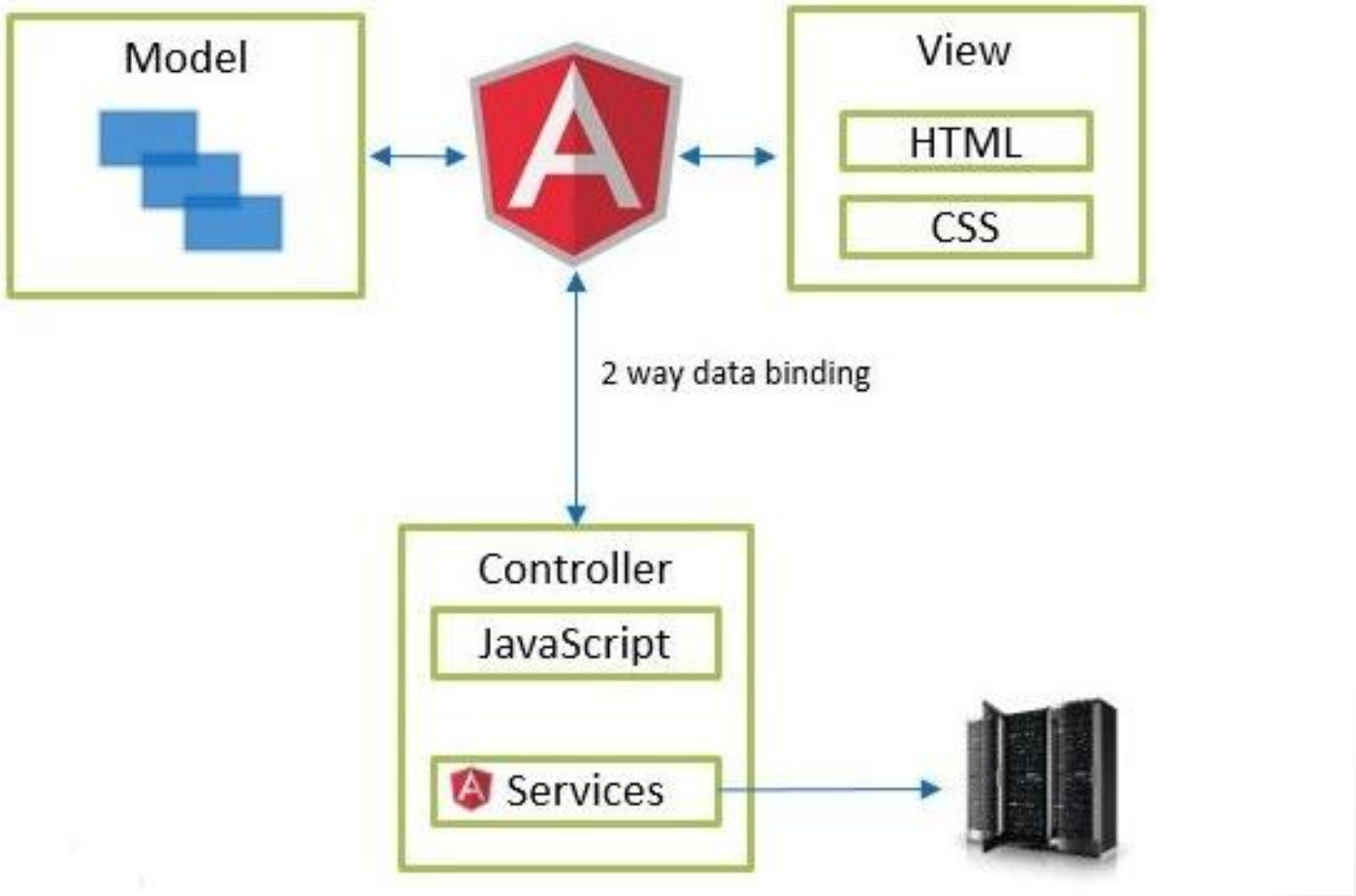
AngularJS - MVC Architecture

- **M**odel **V**iew **C**ontroller or MVC as it is popularly called, is a software design pattern for developing web applications.
- A Model View Controller pattern is made up of the following three parts −
  - **Model** − It is the lowest level of the pattern responsible for **maintaining data.**
  - **View** − It is responsible for **displaying all or a portion of the data to the user.**
  - **Controller** − It is a **software Code that controls the interactions between the Model and View.**

AngularJS - MVC Architecture

- **The Model**
  - The model is responsible for managing application data. It responds to the request from view and to the instructions from controller to update itself.
- **The View**
  - A presentation of data in a particular format, triggered by the controller's decision to present the data. They are script-based template systems such as JSP, ASP, PHP and very easy to integrate with AJAX technology.
- **The Controller**
  - The controller responds to user input and performs interactions on the data model objects. The controller receives input, validates it, and then performs business operations that modify the state of the data model.
  - AngularJS is a MVC based framework. In the coming chapters, we will see how AngularJS uses MVC methodology.

AngularJS - MVC Architecture



AngularJS – application 3 parts

- An AngularJS application consists of following three important parts –
  1. **ng-app** – This directive defines and **links an AngularJS application to HTML.**
  2. **ng-model** – This directive **binds** the values of **AngularJS** application **data** to **HTML input controls**.
  3. **ng-bind** – This directive **binds** the **AngularJS** Application **data** to **HTML tags.**

Steps to create AngularJS

- **Step 1 – Load framework-** using <Script> tag.
- <script src = "https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"> </script>

- **Step 2 – Define AngularJS Application using ng-app directive**
- ...

- **Step 3 – Define a model name using ng-model directive**
- Enter your Name: <input type = "text" ng-model = "name">

- **Step 4 – Bind the value of above model defined using ng-bind directive.**
- Hello !

AngularJS Example

```
<html>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.4/angular.min.js">
</script>
<body>

<div ng-app="">
  <p>Name: <input type="text" ng-model="name"></p>
  <p ng-bind="name"></p>
</div>

</body>
</html>
```

AngularJS Example Explained

- Example explained:
- AngularJS starts automatically when the web page has loaded.
- The **ng-app** directive tells AngularJS that the element is the "owner" of an AngularJS **application**.
- The **ng-model** directive binds the value of the input field to the application variable **name**.
- The **ng-bind** directive binds the **innerHTML** of the <p> element to the application variable **name**.

How AngularJS integrates with HTML

- **ng-app** directive indicates the **start of AngularJS** application.
- **ng-model** directive then **creates a model variable** named **"name"** which can be **used with** the html page and within the div having **ng-app** directive.
- **ng-bind** then **uses the name model** to be displayed in the html span tag whenever user input something in the text box.
- Closing** ** tag indicates the **end of AngularJS** application.

Outline

- Angular JS : Overview,
- MVC architecture,
- directives, expression, controllers,
- filters,
- tables,
- modules,
- forms,
- includes,
- views,
- scopes,
- services,
- dependency injection,
- custom directives,
- Internationalization

AngularJS - Directives

- AngularJS directives are used to extend HTML. These are special attributes starting with ng- prefix.

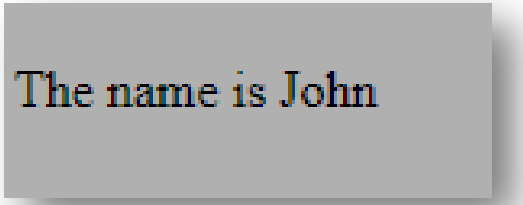
1. **ng-app** – This directive starts an AngularJS Application. It is also used to load various AngularJS modules in AngularJS Application.
   - ...
2. **ng-init** – This directive initializes application data. It is used to put values to the variables to be used in the application.
   -
3. **ng-model** – This directive binds the values of AngularJS application data to HTML input controls.
   - Enter your Name: <input type = "text" ng-model = "name">
4. **ng-repeat** – This directive repeats html elements for each item in a collection.
   -
   - <li ng-repeat = "country in countries"> {{ country.name}}
   -

AngularJS Directives- Example

- AngularJS directives are HTML attributes with an **ng** prefix.
- The **ng-init** directive initializes AngularJS application variables.

- **AngularJS Example**
-
  The name is

OUTPUT



AngularJS - Expressions

- Expressions are used to bind application data to html.
- Expressions are written inside double braces like
  - **{{ expression}}.**
- Expressions behaves in same way as ng-bind directives.

- **Using numbers**
  - Expense on Books : {{cost * quantity}} Rs
- **Using strings**
  - Hello {{fname+ “ “ +lname }}
- **Using object**
  - Roll No: {{student.rollno}}
- **Using array**
  - Marks(Math): {{marks[3]}}

AngularJS Expressions -Example

- AngularJS expressions are written inside double braces: **{{ expression }}**.
- AngularJS will "output" data exactly where the expression is written:

- **AngularJS Example**
- ```
  <html>
  <script
  src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.4/angular.min.j
  s"></script>
  <body>

    My first expression: {{ 5 + 5 }}

  </body>
  </html>
  ```

My first expression: 10
  ```

AngularJS - Controllers

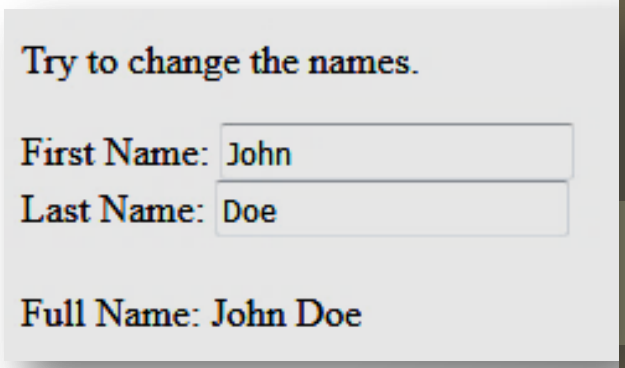
- AngularJS application mainly relies on controllers to control the flow of data in the application.
- A controller is defined using **ng-controlle**r directive.
- Each controller accepts **$scope** as a parameter which refers to the application/module that controller is to control.

  -

AngularJS Controllers- Example

- AngularJS **modules** define AngularJS applications.
- AngularJS **controllers** control AngularJS applications.
- The **ng-app** directive defines the application, the **ng-controller** directive defines the controller.
- **AngularJS Example**
-
  First Name: <input type="text" ng-model="firstName">
  Last Name: <input type="text" ng-model="lastName">
  Full Name: {{firstName + " " + lastName}}

- <script>
  var app = angular.module('**myApp**', []);
  app.controller('**myCtrl**', function($scope) {
  $scope.firstName= "John";
  $scope.lastName= "Doe";
  });
  </script>



AngularJS Controllers- Example Explained

- **AngularJS modules define applications:**
- var app = angular.module('myApp', []);

- **AngularJS controllers control applications:**
- app.controller('myCtrl', function($scope) {
  $scope.firstName= "John";
  $scope.lastName= "Doe";
  });

Outline

- Angular JS : Overview,
- MVC architecture,
- directives, expression, controllers,
- filters,
- tables,
- modules,
- forms,
- includes,
- views,
- scopes,
- services,
- dependency injection,
- custom directives,
- Internationalization

AngularJS - Filters

- Filters are used to change modify the data and can be clubbed in expression or directives using pipe character.

- Following is the list of commonly used filters.

Sr.No.	Name	Description
1	uppercase	converts a text to upper case text.
2	lowercase	converts a text to lower case text.
3	currency	formats text in a currency format.
4	filter	filter the array to a subset of it based on provided criteria.
5	orderby	orders the array based on provided criteria.

AngularJS – Filters- Example

- **uppercase filter**
  - Add uppercase filter to an expression using pipe character.

- **Example**
  - Enter first name:<input type = "text" ng-model = "firstName">
  - Name in Upper Case: {{firstName | uppercase}}

Outline

- Angular JS : Overview,
- MVC architecture,
- directives, expression, controllers,
- filters,
- tables,
- modules,
- forms,
- includes,
- views,
- scopes,
- services,
- dependency injection,
- custom directives,
- Internationalization

AngularJS - Tables

- Table data is normally repeatable by nature. ng-repeat directive can be used to draw table easily.
- The ng-repeat directive repeats a set of HTML, a given number of times.
- The set of HTML will be repeated once per item in a collection.
- The collection must be an array or an object.
- Following example states the use of ng-repeat directive to draw a table.
  - <table >
  - <tr>
  - <th> Name </th>
  - <th> City </th>
  - </tr>
  - <tr ng-repeat="entry in collection">
  - <td> {{entry.name}}</td>
  - <td> {{entry.city}} </td>
  - </tr>
  - </table>

AngularJS – Tables Example

- **AngularFormTable.html**

```
<html ng-app = "simpleApp">
  <head>
 <script src =
"https://ajax.googleapis.com/ajax/libs/angula
rjs/1.3.14/angular.min.js"></script>
  <script src = "index.js"></script>
 </head>
  <body>
  <div  ng-controller = "simpleController">
  <table border= 1>
<tr>
<th> No </th>
<th> Name </th>
<th> City </th>
</tr>

<tr ng-repeat="entry in collection">
<td> {{$index+1}} </td>
<td>  {{entry.name}}  </td>
<td> {{entry.city}} </td>
</tr>
</table>

<form ng-submit="addEntry()">
Name:<input type="text" ng-model="newData.name" >
city:<input type="text" ng-model="newData.city" >
<input type="submit" value="Add Entry">
</form>
</div>
</body>
</html>
```

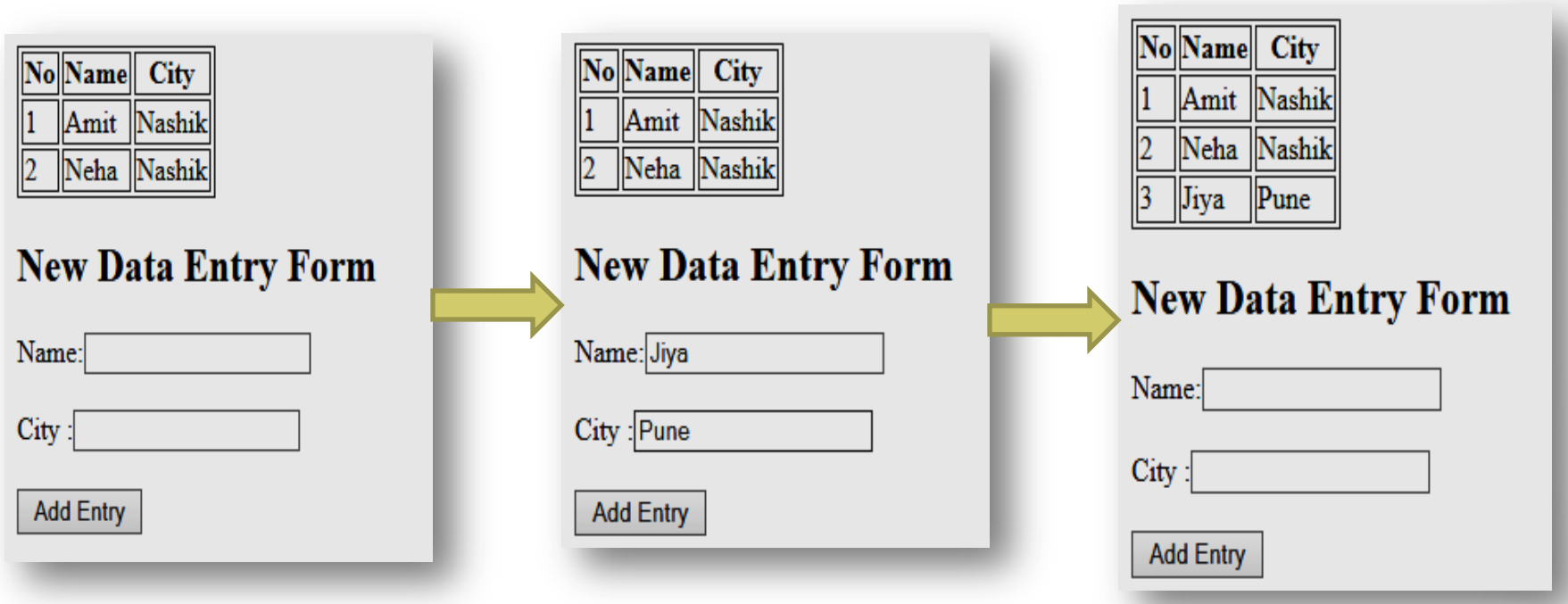
AngularJS – Tables Example

- **Index.js**

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

app.controller("simpleController",function($scope)
{
  $scope.collection=[
        {name:"Amit",city:"Nashik"},
        {name:"Neha",city:"Nashik"}];
  $scope.addEntry=function()
{
        $scope.collection.push($scope.newData);
        $scope.newData='';
};
});
```

AngularJS – Tables Example O/P



Outline

- Angular JS : Overview,
- MVC architecture,
- directives, expression, controllers,
- filters,
- tables,
- modules,
- forms,
- includes,
- views,
- scopes,
- services,
- dependency injection,
- custom directives,
- Internationalization

AngularJS - Modules

- Modules are used to separate logics say services, controllers, application etc. and keep the code clean.

- In this example we're going to create two modules.
- **Application Module** − used to initialize an application with controller(s).
- **Controller Module** − used to define the controller.

AngularJS - Modules

- **Application Module**
- var mainApp = angular.module("mainApp", []);
- Here we've declared an application **mainApp** module using angular.module function. We've passed an empty array to it. This array generally contains dependent modules.
- **Controller Module**
- mainApp.controller("studentController", function($scope)
- Here we've declared a controller **studentController** module using mainApp.controller function.
- **Use Modules**
-
- Here we've used application module using ng-app directive and controller using ng-controller directive.

Outline

- Angular JS : Overview,
- MVC architecture,
- directives, expression, controllers,
- filters,
- tables,
- modules,
- forms,
- includes,
- views,
- scopes,
- services,
- dependency injection,
- custom directives,
- Internationalization

AngularJS - Forms

- AngularJS enriches form filling and validation.
- We can use ng-click to handle AngularJS click on button and
- use $dirty and $invalid flags to do the validations in seemless way.
- Use novalidate with a form declaration to disable any browser specific validation.

- **Events**
- AngularJS provides multiple events which can be associated with the HTML controls.
- ng-click
- ng-dbl-click
- ng-mousedown
- ng-mouseup
- ng-mouseover
- ng-keydown
- ng-keyup
- ng-keypress

AngularJS - Forms

- Forms in AngularJS provides data-binding and validation of input controls.
- **Input Controls-**Input controls are the HTML input elements:
  - input elements
  - select elements
  - button elements
  - textarea elements
- **Validate data-**Following can be used to track error.
  - **$dirty** – states that value has been changed.
  - **$invalid** – states that value entered is invalid.
  - **$error** – states the exact error.

AngularJS – Forms: Textbox Example

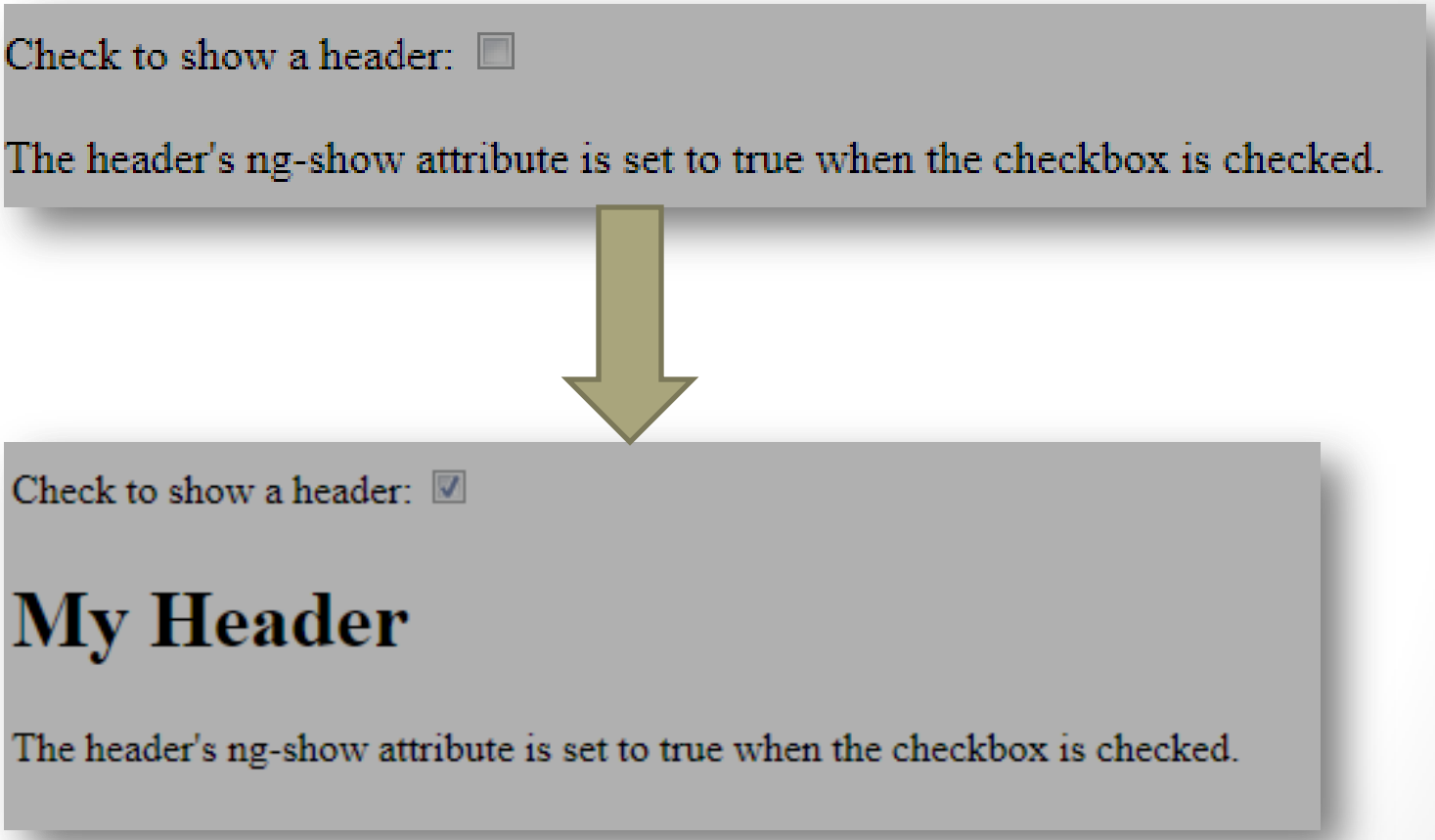
- **Data-Binding**
- Input controls provides data-binding by using the **ng-model** directive.
- **<input type="text" ng-model="firstname">**
- The application does now have a property named firstname.
- The ng-model directive binds the input controller to the rest of your application.
- **Example**
- <form>
  First Name: <input type="text" ng-model="firstname">
  </form>
  <h1>You entered: {{firstname}}</h1>

AngularJS – Forms: Checkbox Example

- A checkbox has the value true or false.
- Apply the ng-model directive to a checkbox, and use its value in your application.

```
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.4/angular.min.js">
</script>
<body>
<div ng-app="">
 <form>
  Check to show a header:
  <input type="checkbox" ng-model="myVar">
 </form>
 <h1 ng-show="myVar">My Header</h1>
</div>
<p>The header's ng-show attribute is set to true when the checkbox is checked.</p>
</body>
</html>
```

AngularJS – Forms: Checkbox Example O/P



Outline

- Angular JS : Overview,
- MVC architecture,
- directives, expression, controllers,
- filters,
- tables,
- modules,
- forms,
- includes,
- views,
- scopes,
- services,
- dependency injection,
- custom directives,
- Internationalization

AngularJS - Includes

- With AngularJS, you can include HTML from an external file using ng-include directive.

- **Example**

```
<body ng-app="">
<div ng-include="'myFile.htm'"></div>
</body>
```

AngularJS – Includes: Example

- *main.html*

Enter first name

<input type = "text" ng-model = "student.firstName" >

Enter last name

<input type = "text" ng-model = "student.lastName">

{{student.fullName()}}

- *tryAngularJS.htm*
- <body>
-
- ** **
- <script>
- var mainApp = angular.module("mainApp", []); mainApp.controller('studentController', function($scope) {
- $scope.student = {
- firstName: "Mahesh",
- lastName: "Parashar",
- fullName: function() {
- var studentObject;
- studentObject = $scope.student;
- return studentObject.firstName + " " + studentObject.lastName;
- } }; });
- </script> </body>

AngularJS – Includes:

Example O/P

Enter first name:	Mahesh
Enter last name:	Parashar
Name:	Mahesh Parashar

Enter first name:	Bhavana
Enter last name:	Khivsara
Name:	Bhavana Khivsara

Outline

- Angular JS : Overview,
- MVC architecture,
- directives, expression, controllers,
- filters,
- tables,
- modules,
- forms,
- includes,
- views,
- scopes,
- services,
- dependency injection,
- custom directives,
- Internationalization

AngularJS - Views

- AngularJS supports Single Page Application via multiple views on a single page.
- To do this AngularJS has provided
  1. ng-view
  2. ng-template directives
  3. $routeProvider services.

AngularJS - Views

- **ng-view**
- ng-view tag simply creates a place holder where a corresponding view can be placed .
  - <div ng-view>
- **ng-template**
- ng-template directive is used to create an html view using script tag. It contains "id" attribute which is used by $routeProvider to map a view with a controller.

  <script type = "text/ng-template" id = "addStudent.htm">
- **$routeProvider**
- $routeProvider is the key service which set the configuration of urls, map them with the corresponding html page or ng-template, and attach a controller with the same.

  mainApp.config(['$routeProvider', function($routeProvider) {

AngularJS – Views: Example

```
<html>
    <head>
     <title>Angular JS Views</title>
 <script src =
"https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"><
/script>
<script src =
"https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular-
route.min.js">
</script>
   </head>
```

AngularJS – Views: Example

```
<body>
   <div ng-app = "mainApp">
      <p><a href = "#add1">Add Student</a></p>
      <p><a href = "#view1">View Students</a></p>
      <div ng-view></div>
    <script type = "text/ng-template" id = "add">
         <h2> Add Student </h2>
         {{message}}
      </script>
    <script type = "text/ng-template" id = "view">
         <h2> View Students </h2>
         {{message}}
      </script>
   </div>
```

AngularJS – Views: Example

```
<script>
 var mainApp = angular.module("mainApp", ['ngRoute']);
 mainApp.config(['$routeProvider', function($routeProvider)
{
        $routeProvider.

        when('/add1', {
          templateUrl: 'add',
          controller: 'AddC'
        }).

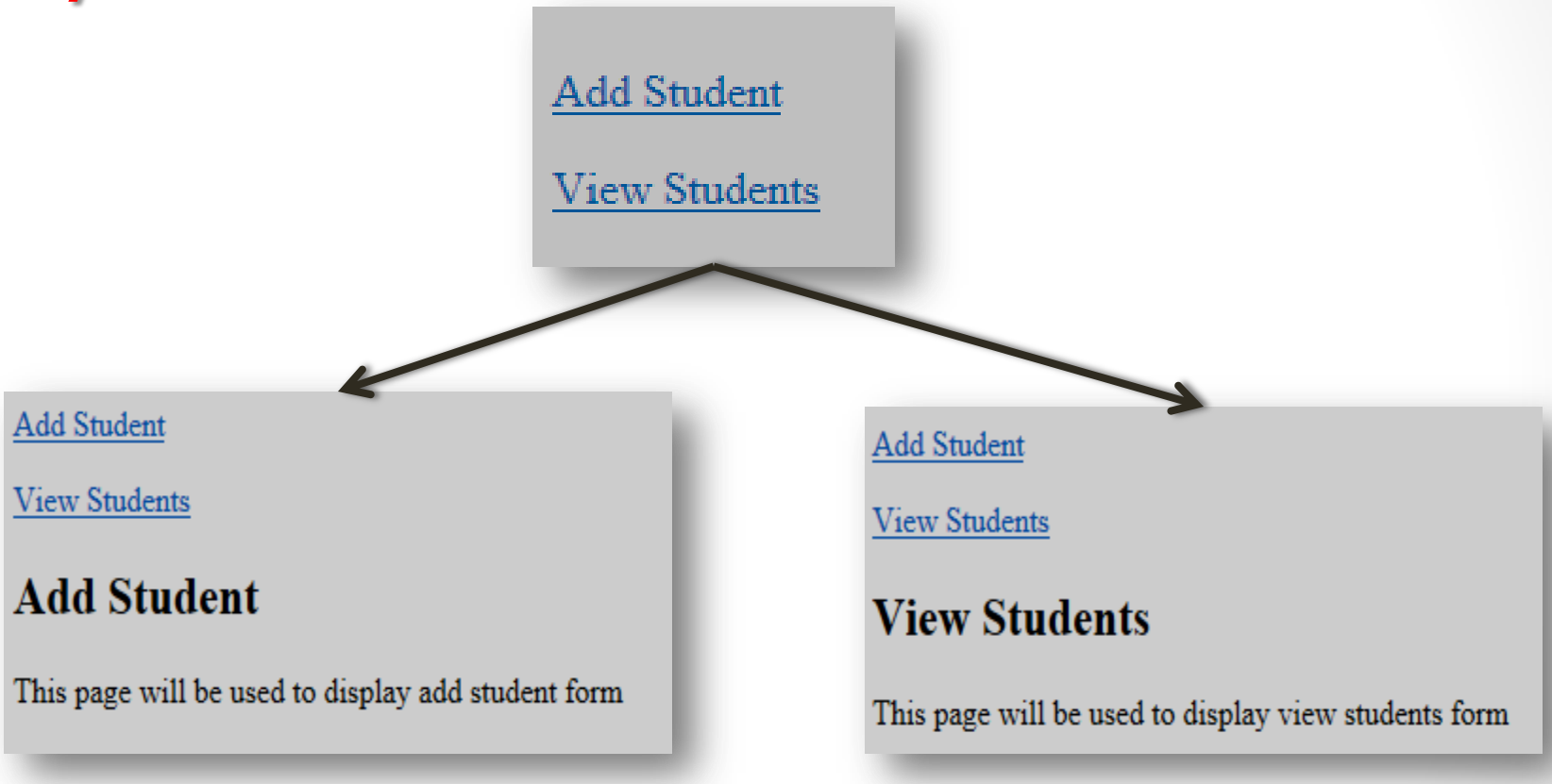
        when('/view1', {
          templateUrl: 'view',
          controller: 'ViewC'
        });
      }]);
```

```
 mainApp.controller('AddC', function($scope)
{
$scope.message = "This page will be used to
display add student form";
    });

  mainApp.controller('ViewC', function($scope) {
  $scope.message = "This page will be used to
display view students form";
        });

    </script>
</body>
</html>
```

AngularJS – Views: Example O/P



Outline

- Angular JS : Overview,
- MVC architecture,
- directives, expression, controllers,
- filters,
- tables,
- modules,
- forms,
- includes,
- views,
- scopes,
- services,
- dependency injection,
- custom directives,
- Internationalization

AngularJS - Scopes

- Scope is a special javascript object which plays the role of joining controller with the views.
- Scope contains the model data. In controllers, model data is accessed via $scope object.

```
<script>
   var mainApp = angular.module("mainApp", []);
      mainApp.controller("shapeController", function($scope) {
      $scope.message = "In shape controller";
      $scope.type = "Shape";
   });
</script>
```

AngularJS - Scopes

- Following are the important points to be considered in above example.
- $scope is passed as first argument to controller during its constructor definition.
- $scope.message and $scope.type are the models which are to be used in the HTML page.
- We've set values to models which will be reflected in the application module whose controller is shapeController.
- We can define functions as well in $scope.

AngularJS – Scopes: Example (scope Inheritance)

```
<script>
    var mainApp = angular.module("mainApp", []);

    mainApp.controller("shapeController", function($scope) {
      $scope.message = "In shape controller";   });

    mainApp.controller("circleController", function($scope) {
      $scope.message = "In circle controller";   });

    mainApp.controller("squareController", function($scope) {
      $scope.message = "In square controller";  });

</script>
```

Outline

- Angular JS : Overview,
- MVC architecture,
- directives, expression, controllers,
- filters,
- tables,
- modules,
- forms,
- includes,
- views,
- scopes,
- services,
- dependency injection,
- custom directives,
- Internationalization

AngularJS - Services

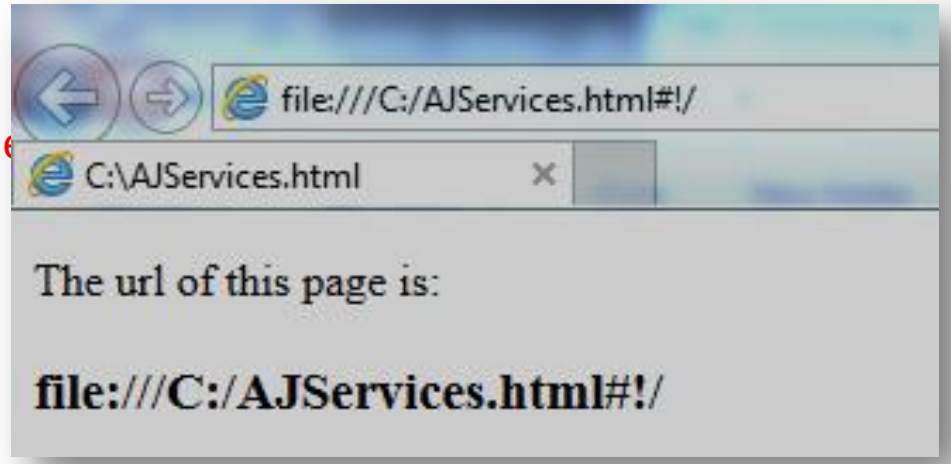
- In AngularJS, a service is a function that is available for AngularJS application.
- AngularJS has about 30 built-in services for example
  - $https:,
  - $route,
  - $window,
  - $location etc.
- Each service is responsible for a specific task for example, $https: is used to make ajax call to get the server data.
- $route is used to define the routing information and so on.
- Inbuilt services are always prefixed with $ symbol.

AngularJS – Services: Example

```
<html>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.4/angular.min.js"></s
cript>
<body>
<div ng-app="myApp" ng-controller="myCtrl">
<p>The url of this page is:</p>
<h3>{{myUrl}}</h3>
</div>
<script>
var app = angular.module('myApp', []);
 app.controller('myCtrl', function($scope, $location) {
    $scope.myUrl = $location.absUrl();
});
</script>
</body>
</html>
```

AngularJS – Services: Example O/P

```
<html>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.4/angular.min.js"></s
cript>
<body>
<div ng-app="myApp" ng-controlle
<p>The url of this page is:</p>
<h3>{{myUrl}}</h3>
</div>
<script>
var app = angular.module('myApp', []);
 app.controller('myCtrl', function($scope, $location) {
    $scope.myUrl = $location.absUrl();
});
</script>
</body>
</html>
```



AngularJS – Services

- There are two ways to create a service.
  - factory
  - service

Outline

- Angular JS : Overview,
- MVC architecture,
- directives, expression, controllers,
- filters,
- tables,
- modules,
- forms,
- includes,
- views,
- scopes,
- services,
- dependency injection,
- custom directives,
- Internationalization

AngularJS - Dependency Injection

- Dependency Injection is a software design pattern in which components are given their dependencies instead of hard coding them within the component.
- This relieves a component from locating the dependency and makes dependencies configurable.
- *This helps in making components reusable, maintainable and testable.*
- AngularJS provides a supreme Dependency Injection mechanism.
- It provides following core components which can be injected into each other as dependencies.
  1. **value**
  2. **factory**
  3. **service**
  4. **provider**
  5. **constant**

AngularJS - Dependency Injection

- **Value**
- value is simple javascript object and it is used to pass values to controller during config phase.

```
//define a module
var mainApp = angular.module("mainApp", []);
//create a value object as "defaultInput" and pass it a data.
mainApp.value("defaultInput", 5);
//inject the value in the controller using its name "defaultInput"
 mainApp.controller('CalcController', function($scope,  defaultInput) {
  $scope.number = defaultInput; });
```

AngularJS - Dependency Injection

- **factory**
- factory is a function which is used to return value. It creates value on demand whenever a service or controller requires. It normally uses a factory function to calculate and return the value.
- **service**
- service is a singleton javascript object containing a set of functions to perform certain tasks. Services are defined using service() functions and then injected into controllers.
- **provider**
- provider is used by AngularJS internally to create services, factory etc. during config phase
- **constant**
- constants are used to pass values at config phase considering the fact that value can not be used to be passed during config phase.

Outline

- Angular JS : Overview,
- MVC architecture,
- directives, expression, controllers,
- filters,
- tables,
- modules,
- forms,
- includes,
- views,
- scopes,
- services,
- dependency injection,
- custom directives,
- Internationalization

AngularJS - Custom Directives

- Custom directives used to extend the functionality of HTML.
- Custom directives are defined using "directive" function.
- A custom directive simply replaces the element for which it is activated.
- AngularJS application during bootstrap finds the matching elements and do one time activity using its compile() method then process the element using link() method .
- AngularJS type of elements.
  1. **Element directives** – Directive activates when a *matching element* is encountered.
  2. **Attribute** – Directive activates when a *matching attribute* is encountered.
  3. **CSS** – Directive activates when a *matching css style* is encountered.
  4. **Comment** – Directive activates when a *matching comment* is encountered.

AngularJS - Custom Directives

- **Understanding Custom Directive**

- ***Define custom html tags.***
- **<student name = "Mahesh"></student>
**
- **<student name = "Piyush"></student>**

- ***Define custom directive to handle above custom html tags.***
- **mainApp.directive('student', function() {**

AngularJS - Custom Directives: Example

```
<html>
    <head>
    <title>Angular JS Custom Directives</title>
  </head>
    <body>
    <h2>AngularJS Sample Application</h2>
      <div ng-app = "mainApp" ng-controller = "StudentController">
      <student name = "Mahesh"></student><br/>
      <student name = "Piyush"></student>
    </div>
  <script src =
"https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js">
</script>
```

AngularJS - Custom Directives: Example

```
<script>
    var mainApp = angular.module("mainApp", []);

     mainApp.directive('student', function() {
     var directive = {};
     directive.restrict = 'E';
     directive.template = "Student: {{student.name}} , Roll No: {{student.rollno}}";
     directive.scope = {
       student : "=name"
     }
     directive.compile = function(element, attributes) {
       var linkFunction = function($scope, element, attributes) {
element.html("Student: "+$scope.student.name +" Roll No: "+$scope.student.rollno+">");
         }  return linkFunction;
     }   return directive;
    });
```

Element directive

template replaces the complete element with its text.

AngularJS - Custom Directives: Example

```
mainApp.controller('StudentController', function($scope) {
            $scope.Mahesh = {};
            $scope.Mahesh.name = "Mahesh Parashar";
            $scope.Mahesh.rollno  = 1;

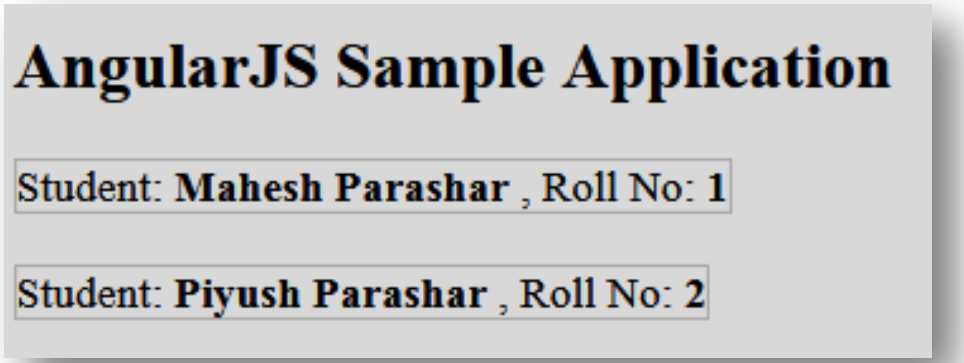
            $scope.Piyush = {};
            $scope.Piyush.name = "Piyush Parashar";
            $scope.Piyush.rollno  = 2;
         });

      </script>

   </body>
</html>
```

AngularJS - Custom Directives:

Example O/P



Outline

- Angular JS : Overview,
- MVC architecture,
- directives, expression, controllers,
- filters,
- tables,
- modules,
- forms,
- includes,
- views,
- scopes,
- services,
- dependency injection,
- custom directives,
- Internationalization

AngularJS - Internationalization

- AngularJS supports inbuilt internationalization for *three types of filters* ***currency, date*** *and* ***numbers.***
- We only need to incorporate corresponding js according to locale of the country.
- By default it handles the locale of the browser.

- For example, to use Danish locale, use following script.

- <script src = "https://code.angularjs.org/1.2.5/i18n/angular-locale_da-dk.js"></script>

AngularJS – Internationalization: Example

```
<html>
<body>
<div ng-app = "mainApp" ng-controller = "StudentController">
      {{fees | currency }} <br>
      {{admissiondate | date }} <br>
</div>
 <script src =
"https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>
<script>
       var mainApp = angular.module("mainApp", []);
         mainApp.controller('StudentController', function($scope) {
          $scope.fees = 100;
          $scope.admissiondate  = new Date();
       });
     </script>
    </body>
</html>
```

AngularJS – Internationalization: Example O/P

$100.00

Feb 23, 2018

Outline

- Angular JS
- Node JS
- Struts

Node JS

Introduction to Node JS

- **What is Node.js?**
  - Node.js is an open source server framework
  - Node.js is free
  - Node.js runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
  - Node.js uses JavaScript on the server
- **What Can Node.js Do?**
  - Node.js can generate dynamic page content
  - Node.js can create, open, read, write, delete, and close files on the server
  - Node.js can collect form data
  - Node.js can add, delete, modify data in your database
- **What is a Node.js File?**
  - Node.js files contain tasks that will be executed on certain events
  - A typical event is someone trying to access a port on the server
  - Node.js files must be initiated on the server before having any effect
  - Node.js files have extension ".js"

Introduction to Node JS

A common task for a web server can be to open a file on the server and return the content to the client.

How PHP or ASP handles a file request:	How Node.js handles a file request:
1. Sends the task to the computer's file system. 2. Waits while the file system opens and reads the file. 3. Returns the content to the client. 4. Ready to handle the next request.	1. Sends the task to the computer's file system. 2. Ready to handle the next request. 3. When the file system has opened and read the file, the server returns the content to the client.

- Node.js eliminates the waiting, and simply continues with the next request.
- Node.js runs single-threaded, non-blocking, asynchronously programming, which is very memory efficient.

Outline

- Angular JS
- Node JS
- Struts

Struts

Outline

- **Struts: Overview,**
- architecture,
- configuration,
- actions,
- interceptors,
- result types,
- validations,
- localization,
- exception handling,
- annotations.

Struts Overview

- Apache Struts 2 is an elegant, extensible framework for creating enterprise-ready Java web applications.
- This framework is designed to streamline the full development cycle from building, to deploying and maintaining applications over time.
- **Struts 2 Features**
  - Configurable MVC components
  - POJO based actions
  - AJAX support
  - Integration support
  - Various Result Types
  - Various Tag support
  - Theme and Template support

Struts Overview :Features

- **1) Configurable MVC components**
- In struts 2 framework, we provide all the components (view components and action) information in struts.xml file. If we need to change any information, we can simply change it in the xml file.
- **2) POJO based actions**
- In struts 2, action class is POJO (Plain Old Java Object) i.e. a simple java class. Here, you are not forced to implement any interface or inherit any class.
- **3) AJAX support**
- Struts 2 provides support to ajax technology. It is used to make asynchronous request i.e. it doesn't block the user. It sends only required field data to the server side not all. So it makes the performance fast.
- **4) Integration Support**
- We can simply integrate the struts 2 application with hibernate, spring, tiles etc. frameworks.
- **5) Various Result Types**
- We can use JSP, freemarker, velocity etc. technologies as the result in struts 2.
- **6) Various Tag support**
- Struts 2 provides various types of tags such as UI tags, Data tags, control tags etc to ease the development of struts 2 application.
- **7) Theme and Template support**
- Struts 2 provides three types of theme support: xhtml, simple and css_xhtml. The xhtml is default theme of struts 2. Themes and templates can be used for common look and feel.

Outline

- Struts: Overview,
- **architecture**,
- configuration,
- actions,
- interceptors,
- result types,
- validations,
- localization,
- exception handling,
- annotations.

Struts Architecture

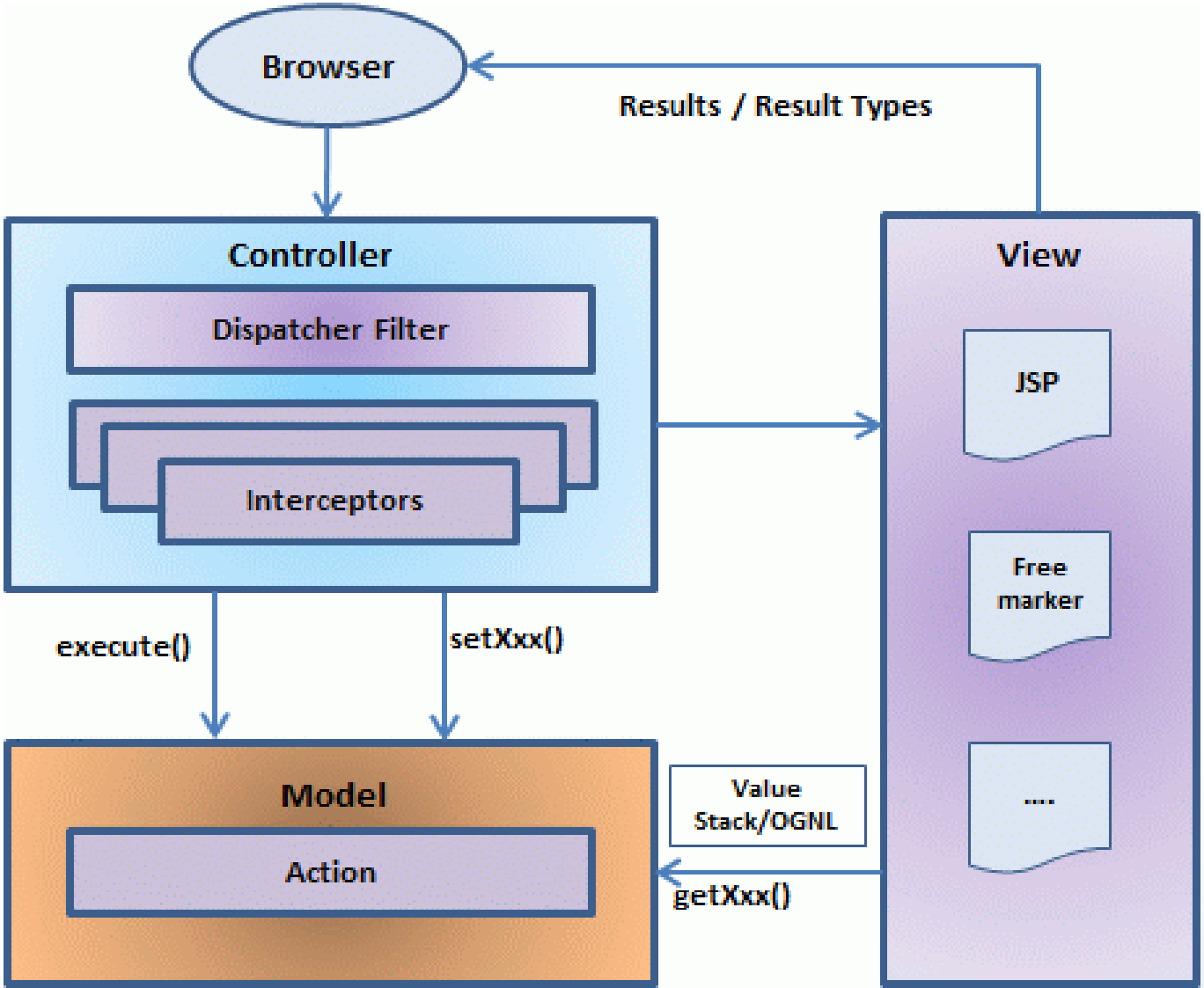
- The **struts 2 framework** is used to develop **MVC-based web application**.
- **M**odel **V**iew **C**ontroller or **MVC** is made up of the following three parts −
  1. **Model** The model represents the state (data) and business logic of the application.
  2. **View** The view module is responsible to display data i.e. it represents the presentation.
  3. **Controller** The controller module acts as an interface between view and model. It intercepts all the requests i.e. receives input and commands to Model / View to change accordingly.

- The Struts framework provides the configurable MVC support. In struts 2, we define all the action classes and view components in struts.xml file.

Struts Architecture

- The Model-ViewController pattern in Struts2 is implemented with the following **five core components** −
  1. Actions
  2. Interceptors
  3. Value Stack / OGNL
  4. Results / Result types
  5. View technologies

Struts Architecture



Struts Architecture

- **Request Life Cycle**
  - Based on the above diagram, you can understand the work flow through user's request life cycle in **Struts 2** as follows –
  - User sends a request to the server for requesting for some resource (i.e. pages).
  - The Filter Dispatcher looks at the request and then determines the appropriate Action.
  - Configured interceptor functionalities applies such as validation, file upload etc.
  - Selected action is performed based on the requested operation.
  - Again, configured interceptors are applied to do any post-processing if required.
  - Finally, the result is prepared by the view and returns the result to the user.

Struts Architecture

- **Advantage of Model 2 (MVC) Architecture**
  - **Navigation control is centralized** Now only controller contains the logic to determine the next page.
  - **Easy to maintain**
  - **Easy to extend**
  - **Easy to test**
  - **Better separation of concerns**

- **Disadvantage of Model 2 (MVC) Architecture**
  - We need to write the controller code self. If we change the controller code, we need to recompile the class and redeploy the application.

Struts 2 - Hello World Example

Create Following Four Components For Any Struts 2 Project

SN.	Components & Description
1	**Action** Create an action class which will contain complete business logic and control the interaction between the user, the model, and the view.
2	**Interceptors** Create interceptors if required, or use existing interceptors. This is part of Controller.
3	**View** Create a JSPs to interact with the user to take input and to present the final messages.
4	**Configuration Files** Create configuration files to couple the Action, View and Controllers. These files are struts.xml, web.xml, struts.properties.

Struts 2 - Hello World Example

1. **In Eclipse File > New > Dynamic Web Project** and enter project name as **HelloWorldStruts2**
2. Select all the default options in the next screens and finally check **Generate Web.xml deployment descriptor** option.
3. Copy following **Strut 2 jar files** in the lib folder of your project.
   - commons-fileupload-x.y.z.jar
   - commons-io-x.y.z.jar
   - commons-lang-x.y.jar
   - commons-logging-x.y.z.jar
   - commons-logging-api-x.y.jar
   - freemarker-x.y.z.jar
   - javassist-.xy.z.GA
   - ognl-x.y.z.jar
   - struts2-core-x.y.z.jar
   - xwork-core.x.y.z.jar

Struts 2 - Hello World Example

4. **Create Action Class-** create a java file HelloWorldAction.java under **Java Resources > src** with a package name **com.struts2** with the contents given below.

```
public class HelloWorldAction {
   private String name;

   public String execute() throws Exception {
      return "success";
   }

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }
}
```

Struts 2 - Hello World Example

5. **Create a View-**create the below jsp file **HelloWorld.jsp** in the WebContent folder in your eclipse project.

```
<%@ page contentType = "text/html; charset = UTF-8" %>
<%@ taglib prefix = "s" uri = "/struts-tags" %>

<html>
   <head>
      <title>Hello World</title>
   </head>

   <body>
      Hello World, <s:property value = "name"/>
   </body>
</html>
```

Struts 2 - Hello World Example

6. **Create Main Page-** We also need to create **index.jsp** in the WebContent folder. This file will serve as the initial action URL where a user can click to tell the Struts 2 framework to call a defined method of the HelloWorldAction class and render the HelloWorld.jsp view.

```
<%@ page language = "java" contentType = "text/html; charset = ISO-8859-1"
   pageEncoding = "ISO-8859-1"%>
<%@ taglib prefix = "s" uri = "/struts-tags"%>
   <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>
   <head>
      <title>Hello World</title>
   </head>

   <body>
      <h1>Hello World From Struts2</h1>
      <form action = "hello">
         <label for = "name">Please enter your name</label><br/>
         <input type = "text" name = "name"/>
         <input type = "submit" value = "Say Hello"/>
      </form>
   </body>
</html>
```

Struts 2 - Hello World Example

7. **Configuration Files-** We need a mapping to tie the URL, the HelloWorldAction class (Model), and the HelloWorld.jsp (the view) together. Hence, create struts.xml file under the WebContent/WEB-INF/classes folder. Eclipse does not create the "classes" folder by default, so you need to do this yourself.

```
<?xml version = "1.0" Encoding = "UTF-8"?>
<!DOCTYPE struts PUBLIC
   "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
   "http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>
   <constant name = "struts.devMode" value = "true" />

   <package name = "helloworld" extends = "struts-default">
     <action name = "hello"
         class = "com.tutorialspoint.struts2.HelloWorldAction"
         method = "execute">
         <result name = "success">/HelloWorld.jsp</result>
      </action>
   </package>
</struts>
```

Struts 2 - Hello World Example

- Next step is to create a **web.xml** file which is an entry point for any request to Struts 2. The entry point of Struts2 application will be a filter defined in deployment descriptor (web.xml). Hence, we will define an entry of org.apache.struts2.dispatcher.FilterDispatcher class in web.xml.

```
   <display-name>Struts 2</display-name>

   <welcome-file-list>
      <welcome-file>index.jsp</welcome-file>
   </welcome-file-list>

   <filter>
      <filter-name>struts2</filter-name>
      <filter-class>
         org.apache.struts2.dispatcher.FilterDispatcher
      </filter-class>
   </filter>

   <filter-mapping>
      <filter-name>struts2</filter-name>
      <url-pattern>/*</url-pattern>
   </filter-mapping>
</web-app>
```

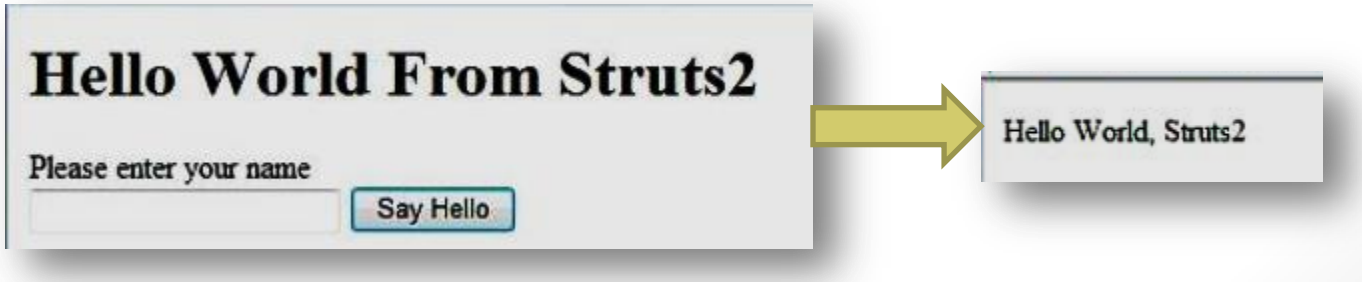
Struts 2 - Hello World Example

8. **To Enable Detailed Log-** You can enable complete logging functionality while working with Struts 2 by creating **logging.properties** file under **WEB-INF/classes** folder. Keep the following two lines in your property file −

```
org.apache.catalina.core.ContainerBase.[Catalina].level = INFO
org.apache.catalina.core.ContainerBase.[Catalina].handlers = \
   java.util.logging.ConsoleHandler
```

Struts 2 - Hello World Example

- **Procedure for Executing the Application**
- Right click on the project name and click **Export > WAR File** to create a War file.
- Then deploy this WAR in the Tomcat's webapps directory.
- Finally, start Tomcat server and try to access URL **http://localhost:8080/HelloWorldStruts2/index.jsp**. This will give you following screen −
- Enter a value "Struts2" and submit the page. You should see the next page



Outline

- Struts: Overview,
- architecture,
- **configuration**,
- actions,
- interceptors,
- result types,
- validations,
- localization,
- exception handling,
- annotations.

Struts Configuration

Basic configuration which is required for a **Struts 2** application.

Important configuration files are

- **web.xml,**
- **struts.xml,**
- **struts-config.xml,**
- **struts.properties**

Struts Configuration

- **The web.xml File**
- The web.xml configuration file is a J2EE configuration file that determines how elements of the HTTP request are processed by the servlet container.
- It is not strictly a Struts2 configuration file, but it is a file that needs to be configured for Struts2 to work.
- As discussed earlier, this file provides an entry point for any web application
- The web.xml file needs to be created under the folder **WebContent/WEB-INF**.

Struts Configuration

- **The Struts.xml File**
- The **struts.xml** file contains the configuration information that you will be modifying as actions are developed.
- This file can be used to override default settings for an application, for example *struts.devMode = false* and other settings which are defined in property file.
- This file can be created under the folder **WEB-INF/classes**.
- The first thing to note is the **DOCTYPE**.
- <struts> is the root tag element, under which we declare different packages using <package> tags.

Struts Configuration

- **The Struts-config.xml File**
- The struts-config.xml configuration file is a link between the View and Model components in the Web Client
- The configuration file basically contains following main elements −
  - **struts-config-** This is the root node of the configuration file.
  - **form-beans-**This is where you map your ActionForm subclass to a name.
  - **action-mappings-**This is where you declare form handlers
  - **Controller-**This section configures Struts internals
  - **plug-in-**This section tells Struts where to find your properties files, which contain prompts and error messages

Struts Configuration

- **The Struts.properties File**
- This configuration file provides a mechanism to change the default behavior of the framework.
- you can create this file under the folder **WEB-INF/classes**.
- The values configured in this file will override the default values configured in **default.properties** which is contained in the struts2-core-x.y.z.jar distribution.

Outline

- Struts: Overview,
- architecture,
- configuration,
- **actions,**
- interceptors,
- result types,
- validations,
- localization,
- exception handling,
- annotations.

Struts 2 - Actions

- **Actions** are the core of the Struts2 framework, as they are for any MVC (Model View Controller) framework.
- Each URL is mapped to a specific action, which provides the processing logic which is necessary to service the request from the user.
- But the action also serves in two other important capacities.
- Firstly, the action plays an important role in the transfer of data from the request through to the view, whether its a JSP or other type of result.
- Secondly, the action must assist the framework in determining which result should render the view that will be returned in the response to the request.

Struts 2 - Actions

- **Create Action**
- The only requirement for actions in **Struts2** is that there must be one noargument .
- If the no-argument method is not specified, the default is to use the execute() method.
- Extend the **ActionSupport** class which implements six interfaces including **Action** interface.
- The Action interface is as follows –

```
public interface Action {
   public static final String SUCCESS = "success"
   public static final String NONE = "none";
   public static final String ERROR = "error";
   public static final String INPUT = "input";
   public static final String LOGIN = "login";
   public String execute() throws Exception;
}
```

- Action method in the Hello World example –

```
public class HelloWorldAction extends ActionSupport {
   private String name;

   public String execute() throws Exception {
      if ("SECRET".equals(name)) {
         return SUCCESS;
      } else {
         return ERROR;
      }
   }

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }
}
```

Outline

- Struts: Overview,
- architecture,
- configuration,
- actions,
- **interceptors,**
- result types,
- validations,
- localization,
- exception handling,
- annotations.

Struts 2 - Interceptors

- Interceptors allow for crosscutting functionality to be implemented separately from the action as well as the framework. You can achieve the following using interceptors −
  - Providing preprocessing logic before the action is called.
  - Providing postprocessing logic after the action is called.
  - Catching exceptions so that alternate processing can be performed.
- Many of the features provided in the **Struts2** framework are implemented using interceptors;
- **Examples** include exception handling, file uploading, lifecycle callbacks, etc. In fact, as Struts2 emphasizes much of its functionality on interceptors

Struts 2 - Interceptors

Few of the important interceptors are listed below –

Interceptor & Description
Alias-Allows parameters to have different name aliases across requests.
Checkbox- Assists in managing check boxes by adding a parameter value of false for check boxes that are not checked.
I18n-Keeps track of the selected locale during a user's session.
Exception- Maps exceptions that are thrown from an action to a result, allowing automatic exception handling via redirection.
fileUpload- Facilitates easy file uploading.
Params- Sets the request parameters on the action.
Prepare- This is typically used to do pre-processing work, such as setup database connections.
Timer- Provides simple profiling information in the form of how long the action takes to execute.
Token-Checks the action for a valid token to prevent duplicate formsubmission.
Validation-Provides validation support for actions

Struts 2 - Interceptors

- **Create Interceptors**
- We will use the **timer** interceptor whose purpose is to measure how long it took to execute an action method.
- At the same time, using **params** interceptor whose purpose is to send the request parameters to the action.
- modify the **struts.xml** file to add an interceptor

```
<interceptor-ref name = "params"/>
<interceptor-ref name = "timer" />
```

Struts 2 - Interceptors

- **Create Custom Interceptors**
- Using custom interceptors in your application is an elegant way to provide crosscutting application features.
- init() method provides a way to initialize the interceptor, and the destroy() method provides a facility for interceptor cleanup.
- The **ActionInvocation** object provides access to the runtime environment.

```
public interface Interceptor extends Serializable {
   void destroy();
   void init();
   String intercept(ActionInvocation invocation)
   throws Exception;
}
```

Outline

- Struts: Overview,
- architecture,
- configuration,
- actions,
- interceptors,
- **result types,**
- validations,
- localization,
- exception handling,
- annotations.

Struts 2 - Results & Result Types

- **<results>** tag plays the role of a **view** in the Struts2 MVC framework.
- The action is responsible for executing the business logic.
- The next step after executing the business logic is to display the view using the **<results>** tag.
- For example, if the action method is to authenticate a user, there are three possible outcomes.
  - Successful Login
  - Unsuccessful Login - Incorrect username or password
  - Account Locked

Struts 2 - Results & Result Types

- Struts comes with a number of predefined **result types**
  1. **Dispatcher**
  2. **Velocity,**
  3. **Freemaker,**
  4. **Redirect**
  5. **XSLT** and **Tiles**.

Struts 2 - Results & Result Types

- **The Dispatcher Result Type**
- The **dispatcher** result type is the default type, and is used if no other result type is specified. It's used to forward to a servlet, JSP, HTML page, and so on, on the server. It uses the *RequestDispatcher.forward()* method.
  - <result name = "success" type = "dispatcher">
  - /HelloWorld.jsp
  - </result>
- **The FreeMaker Result Type**
- Freemaker is a popular templating engine that is used to generate output using predefined templates.
- Let us now create a Freemaker template file called **hello.fm** with the following contents −
  - Hello World ${name}
- The above file is a template where **name** is a parameter which will be passed from outside using the defined action.

Struts 2 - Results & Result Types

- **The Redirect Result Type**
- The **redirect** result type calls the standard *response.sendRedirect()* method, causing the browser to create a new request to the given location.
- We can provide the location either in the body of the <result...> element or as a <param name = "location"> element. Redirect also supports the **parse** parameter. Here's an example configured using XML –
  - <result name = "success" type = "redirect">
  - <param name = "location">
  - /NewWorld.jsp
  - </param >
  - </result>

Outline

- Struts: Overview,
- architecture,
- configuration,
- actions,
- interceptors,
- result types,
- **validations,**
- localization,
- exception handling,
- annotations.

Struts 2 - Validations Framework

- At the Struts core, we have the validation framework that assists the application to run the rules to perform validation before the action method is executed.
- we will take an example of an **Employee** whose age ,we will put validation to make sure that the user always enters a age in a range between 28 and 65.

Struts 2 - Validations Framework

- **Create Main Page-** JSP file **index.jsp**, which will be used to collect Employee related information mentioned above.

```
<%@ page language = "java" contentType = "text/html; %>
<%@ taglib prefix = "s" uri = "/struts-tags"%>
<html>
      <body>
      <s:form action = "empinfo" method = "post">
       <s:textfield name = "age" label = "Age" size = "20" />
         <s:submit name = "submit" label = "Submit" align="center" />
      </s:form>
   </body>
</html>
```

Struts 2 - Validations Framework

- **Create Views-** We will use JSP file success.jsp which will be invoked in case defined action returns SUCCESS.

```
<%@ page language = "java" contentType = "text/html; "%>
<%@ taglib prefix = "s" uri = "/struts-tags"%>
<html>
    <body>
    Employee Information is captured successfully.
  </body>
</html>
```

Struts 2 - Validations Framework

- **Create Action-** action class **Employee** with amethod called **validate()** in **Employee.java** file.

```
public class Employee extends ActionSupport {
private int age;
public String execute() {return SUCCESS;  }
public int getAge() {   return age;  }
 public void setAge(int age) { this.age = age;  }
 if (age < 28 || age > 65) {
        addFieldError("age","Age must be in between 28 and 65"); }
  }
```

Struts 2 - Validations Framework

- **Configuration Files**
- Finally, let us put everything together using the **struts.xml** configuration file as follows −

```
<?xml version = "1.0" Encoding = "UTF-8"?>
<!DOCTYPE struts PUBLIC ……>
 <struts>
  <constant name = "struts.devMode" value = "true" />
  <package name = "helloworld" extends = "struts-default">
    <action name = "empinfo"  method = "execute">
      <result name = "input">/index.jsp</result>
      <result name = "success">/success.jsp</result>
    </action>
  </package>
</struts>
```

Outline

- Struts: Overview,
- architecture,
- configuration,
- actions,
- interceptors,
- result types,
- validations,
- **localization,**
- exception handling,
- annotations.

Struts2 - Localization

- Internationalization (i18n) is the process of planning and implementing products and services so that they can easily be adapted to specific local languages and cultures, a process called localization.
- The internationalization process is called translation or localization enablement.
- Internationalization is abbreviated **i18n** because the word starts with the letter **“i”** and ends with **“n”**, and there are 18 characters between the first i and the last n.
- Struts2 provides localization, i.e., internationalization (i18n) support through **resource bundles**, interceptors and tag libraries in the following places −
  - The UI Tags
  - Messages and Errors.
  - Within action classes.

Struts2 – Localization

Resource Bundles

- Struts2 uses resource bundles to provide multiple language and locale options to the users of the web application.
- You don't need to worry about writing pages in different languages.
- All you have to do is to create a resource bundle for each language that you want.
- The resource bundles will contain titles, messages, and other text in the language of your user.
- Resource bundles are the file that contains the key/value pairs for the default language of your application.
- The simplest naming format for a resource file is −
  - **bundlename_language_country.properties**

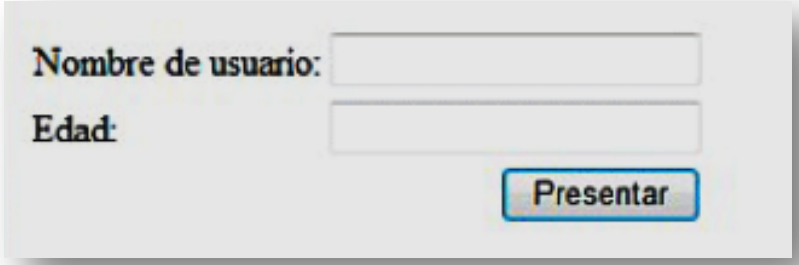
- Here, bundlename could be ActionClass, Interface, SuperClass, Model, Package, Global resource properties.
- Next part language_country represents the country locale for example,
  - **Spanish (Spain) locale is represented by es_ES, and**
  - **English (United States) locale is represented by en_US etc.**

Struts2 – Localization

- To develop your application in multiple languages, you should maintain multiple property files corresponding to those languages/locale and define all the content in terms of key/value pairs.
- For example, if you are going to develop your application for US English (Default), Spanish, and French, then you would have to create three properties files.
  - **global.properties** – By default English (United States) will be applied
  - **global_fr.properties** – This will be used for Franch locale.
  - **global_es.properties** – This will be used for Spanish locale.

Struts2 – Localization Example

- Now select any of the languages, let us say we select **Spanish**, it would display the following result –



Outline

- Struts: Overview,
- architecture,
- configuration,
- actions,
- interceptors,
- result types,
- validations,
- **localization,**
- **exception handling,**
- annotations.

Struts 2 - Exception Handling

- **Struts** provides an easier way to handle uncaught exception and redirect users to a dedicated error page. You can easily configure Struts to have different error pages for different exceptions.
- Struts makes the exception handling easy by the use of the "exception" interceptor.
- Create a file called **Error.jsp**
- **<html>**
- **<body>**
- **This is my custom error page**
- **</body>**
- **</html>**
- Let us now configure Struts to use this this error page in case of an exception.
- Let us modify the **struts.xml** by adding following line–
- **<result name = "error">/Error.jsp</result>**

Outline

- Struts: Overview,
- architecture,
- configuration,
- actions,
- interceptors,
- result types,
- validations,
- localization,
- exception handling,
- **annotations.**

Struts 2 - Annotations

- As mentioned previously, Struts provides two forms of configuration. The traditional way is to use the **struts.xml** file for all the configurations. The other way of configuring Struts is by using the Java 5 Annotations feature. Using the struts annotations, we can achieve **Zero Configuration**.
- To start using annotations in your project, make sure you have included the following jar files in your **WebContent/WEB-INF/lib** folder –
  - struts2-convention-plugin-x.y.z.jar
  - asm-x.y.jar
  - antlr-x.y.z.jar
  - commons-fileupload-x.y.z.jar
  - commons-io-x.y.z.jar
  - commons-lang-x.y.jar
  - commons-logging-x.y.z.jar
  - commons-logging-api-x.y.jar
  - freemarker-x.y.z.jar
  - javassist-.xy.z.GA
  - ognl-x.y.z.jar
  - struts2-core-x.y.z.jar
  - xwork-core.x.y.z.jar

Struts 2 – Annotations Types

Annotations Types
Namespace Annotation (Action Annotation)
Result Annotation - (Action Annotation)
Results Annotation - (Action Annotation) The @Results annotation defines a set of results for an Action.
After Annotation - (Interceptor Annotation) The @After annotation marks a action method that needs to be called after the main action method and the result was executed. Return value is ignored.
Before Annotation - (Interceptor Annotation) The @Before annotation marks a action method that needs to be called before the main action method and the result was executed. Return value is ignored.
EmailValidator Annotation - (Validation Annotation)
IntRangeFieldValidator Annotation - (Validation Annotation)
RequiredFieldValidator Annotation - (Validation Annotation)

Struts 2 – Annotations Example

Validation on Name field as required and Age as range between 28 and 65.

```
import org.apache.struts2.convention.annotation.Action;
import org.apache.struts2.convention.annotation.Result;
import org.apache.struts2.convention.annotation.Results;
import com.opensymphony.xwork2.validator.annotations.*;

public class Employee extends ActionSupport {
  private String name;
  private int age;

@RequiredFieldValidator( message = "The name is required" )

  public String getName() {  return name;   }

  public void setName(String name) {    this.name = name;   }

  @IntRangeFieldValidator(message = "Age must be in between 28 and 65", min = "28", max = "65")

  public int getAge() {  return age;   }

  public void setAge(int age) {   this.age = age;   }   }
```

References

- https://docs.angularjs.org/tutorial
- https://www.tutorialspoint.com/angularjs/angularjs_mvc_architecture.htm
- https://inkoniq.com/blog/googles-baby-angularjs-is-now-the-big-daddy-of-javascript-frameworks/
- https://www.tutorialspoint.com/angularjs/angularjs_first_application.htm
- https://www.tutorialspoint.com/angularjs/angularjs_directives.htm
- https://www.tutorialspoint.com/angularjs/angularjs_filters.htm
- https://www.w3schools.com/angular/angular_forms.asp
- https://www.tutorialspoint.com/angularjs/angularjs_views.htm
- https://www.tutorialspoint.com/angularjs/angularjs_scopes.htm
- https://www.tutorialspoint.com/angularjs/angularjs_services.htm
- https://www.tutorialspoint.com/angularjs/angularjs_dependency_injection.htm
- https://www.tutorialspoint.com/angularjs/angularjs_custom_directives.htm
- https://www.tutorialspoint.com/angularjs/angularjs_internationalization.htm
- https://www.w3schools.com/nodejs/nodejs_intro.asp
- https://www.javatpoint.com/struts-2-features-tutorial
- https://www.javatpoint.com/model-1-and-model-2-mvc-architecture
- https://www.tutorialspoint.com/struts_2/struts_architecture.htm
- https://www.tutorialspoint.com/struts_2/struts_examples.htm
- https://www.tutorialspoint.com/struts_2/struts_configuration.htm
- https://www.tutorialspoint.com/struts_2/struts_actions.htm
- https://www.tutorialspoint.com/struts_2/struts_interceptors.htm
- https://www.tutorialspoint.com/struts_2/struts_result_types.htm
- https://www.tutorialspoint.com/struts_2/struts_validations.htm
- https://www.tutorialspoint.com/struts_2/struts_localization.htm
- https://www.tutorialspoint.com/struts_2/struts_exception_handling.htm
- https://www.tutorialspoint.com/struts_2/struts_annotations.htm