

AngularJs - Base

Rivolto colleghi Add Value del team Arca

Redatto da Emanuele Lucchi, Giuseppe Giordano

Data Ottobre 2016

Cos'è Angular e Chi lo sviluppa

Angular (AngularJS) serve per lo sviluppo di applicazioni Web client side:

- è un **framework JavaScript**
- **estende** gli attributi HTML con **Directive**
- **gestisce** i dati (binds data) all'interno delle definizioni HTML con le **Espressioni**

Il framework è open source e principalmente sviluppato da:

- **Google**
- una **comunità** di sviluppatori

riferimenti

Siti interessanti dove reperire informazioni ed esempi sono:

- <https://angularjs.org/>
- <http://www.w3schools.com/angular/default.asp>
- <http://www.tutorialspoint.com//angularjs/index.htm>


Prerequisiti

I prerequisiti tecnici per utilizzare a pieno Angular sono:

- HTML
- CSS
- JavaScript

Strumenti/Tool

Gli strumenti per lo sviluppo sono:

- editor HTML
 - Browser
 - **Libreria Angular**
-  **Eclipse** (fornisce questo e altri strumenti)

- Scaricata in locale nel proprio workspace
- Importata via web (serve connessione internet), es:

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
```

Utility di sviluppo - Jetty

Utilizzato per creare un Application Server in locale sul quale poter eseguire l'applicazione

```
<build>
  <plugins>
    <plugin>
      <groupId>org.mortbay.jetty</groupId>
      <artifactId>jetty-maven-plugin</artifactId>
      <version>7.3.1.v20110307</version>
      <configuration>
        <reload>manual</reload>
        <webAppConfig>
          <contextPath>/corsoAngular</contextPath>
        </webAppConfig>
        <classesDirectory>../CorsoAngularJsAndSpringMVC/target/classes</classesDirectory>
        <webAppSourceDirectory>../CorsoAngularJsAndSpringMVC/src/main/webapp</webAppSourceDirectory>
        <stopPort>9966</stopPort>
        <stopKey>foo</stopKey>
      </configuration>
    </plugin>
  </plugins>
</build>
```

Il server viene lanciato attraverso il comando **mvn jetty:run**

Il server viene «stoppato» attraverso il comando **mvn jetty:stop**

Utility di sviluppo – Google Chrome

Utilizzato per fare debug della parte Javascript della nostra applicazione

The screenshot shows a web application running in Google Chrome. The application has two main sections: "Ricerca" (Search) and "Aggiungi" (Add). The "Ricerca" section has a text input field containing "aaaa" and a "Ricerca" button. The "Aggiungi" section has two text input fields, one labeled "titolo" and one labeled "autore", and an "Aggiungi" button.

The Chrome DevTools interface is open, showing the "Sources" panel. The file tree on the left indicates the application is running on localhost:8080. The selected file is "EsempioIntegrazioneMvc.js". The code editor shows the following JavaScript code:

```

1 angular.module('corsoBaseAngular.controllers')
2
3 .controller('EsempioIntegrazioneMvcController', [ '$scope', 'EsempioIn
4                                     , function($scope,E
5
6     $scope.autore = '';
7
8     $scope.risultatiRicerca ;
9
10    $scope.libroNew ;
11
12    $scope.ricerca = function() {
13        EsempioIntegrazioneMvcService.ricerca($scope.autore).success(
14
15        $scope.risultatiRicerca = risultatiRice "aaaa"
16
17    }).error(function(error) {
18        console.log("Errore nella ricerca");
19    });
20
21 };
22
23 $scope.aggiungi = function() {

```

A tooltip is visible over the value "aaaa" in the code, indicating the current value of the variable.

Esempio 1

```
<!DOCTYPE html>
<html>
  <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
  <body>

    <p>Try to change the names.</p>
    <div ng-app="myApp" ng-controller="myCtrl">
      First Name: <input type="text" ng-model="firstName"><br>
      Last Name: <input type="text" ng-model="lastName"><br>
      <br>
      Full Name: {{firstName + " " + lastName}}
    </div>

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

  </body>
</html>
```

Try to change the names.

First Name:

Last Name:

Full Name: John Doe

http://www.w3schools.com/angular/angular_intro.asp

Esempio 1

**H
T
M
L**

```

<!DOCTYPE html>
<html>
  <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
  <body>

    <p>Try to change the names.</p>
    <div ng-app="myApp" ng-controller="myCtrl">
      First Name: <input type="text" ng-model="firstName"><br>
      Last Name: <input type="text" ng-model="lastName"><br>
      <br>
      Full Name: {{firstName + " " + lastName}}
    </div>

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

  </body>
</html>

```

Import libreria

browser

**Javascript/
Angular**

Try to change the names.

First Name:

Last Name:

Full Name: John Doe

http://www.w3schools.com/angular/angular_intro.asp

Proprietà di Angular

Angular fornisce quanto occorre per creare applicazioni come le **Single Page**

Application: una applicazione web che può essere usata o consultata su una singola pagina web:

- **tutto il codice** necessario (HTML, JavaScript e CSS) è recuperato in un **singolo caricamento** della pagina
- le **risorse** appropriate sono **caricate dinamicamente** e aggiunte alla pagina quando necessario (di solito in risposta ad azioni dell'utente)

Tra le principali funzionalità a supporto dello sviluppo di tali applicazioni segnaliamo:

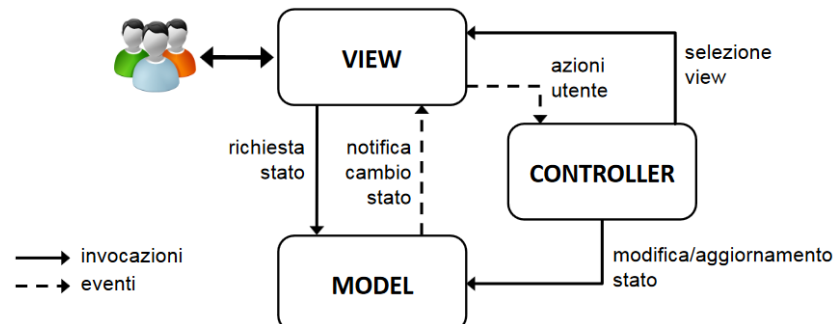
- il supporto al **pattern MVC**
- il **binding bidirezionale** (*two-way binding*)
- la **dependency injection**
- il supporto ai moduli
- la separazione delle competenze
- la testabilità del codice
- la riusabilità dei componenti

pattern MVC

Il pattern è basato sulla separazione dei compiti fra i **componenti** software che interpretano **tre ruoli** principali:

- il **model** fornisce i metodi per accedere ai dati utili all'applicazione;
➤ nel caso di un'applicazione per una biblioteca potrebbero essere le classi Libro, Autore, Scaffale
- il **view** visualizza i dati contenuti nel model e si occupa dell'interazione con utenti e agenti;
➤ nel caso della biblioteca potrebbero essere le pagine HTML del catalogo o i form di ricerca
- il **controller** riceve i comandi dell'utente (in genere attraverso il view) e li attua modificando lo stato degli altri due componenti.

Lo schema implica la **separazione** fra la **logica applicativa** (in questo contesto spesso chiamata "logica di business"), a carico del controller e del model, e l'**interfaccia** utente a carico del view.



binding bidirezionale (*two-way binding*)

In Angular la **gestione dei dati** tra model e view è sincronizzata:

- quando il dato nella *model* cambia, simultaneamente di riflesso la *view* cambia;
- quando il dato nella *view* cambia, il dato nella *model* è aggiornato.

Questo avviene istantaneamente e automaticamente.

dependency injection

La Dependency injection è uno **schema di progettazione** dove:

- la costruzione dell'oggetto A è gestita da un componente esterno (**injector**, assembler, provider o container) che si occupa della creazione e delle sue relative dipendenze;
- ad un oggetto B che ne ha bisogno, l'injector passa il componente A pronto per l'uso, così B può dedicarsi solo richiamarne le funzionalità.

Esempio 1

```
<!DOCTYPE html>
<html>
  <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
  <body>
    <p>Try to change the names.</p>
    <div ng-app="myApp" ng-controller="myCtrl">
      First Name: <input type="text" ng-model="firstName"><br>
      Last Name: <input type="text" ng-model="lastName"><br>
      <br>
      Full Name: {{firstName + " " + lastName}}
    </div>

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

View (points to the HTML structure)

Model Angular (points to `ng-model="firstName"`)

Controller Angular (points to the `myCtrl` controller function)

Dependency Injection (points to the `$scope` parameter in the controller function)

Binding bidirezionale (points to the `Full Name` expression and the input fields)

Try to change the names.

First Name:

Last Name:

Full Name: John Doe

In pratica

Angular:

- **estende** gli attributi HTML con **Directive**
attributi col prefisso **ng**
- **gestisce** i dati (binds data) all'interno del HTML con **Expressions**
espressioni racchiuse tra doppie parentesi grafe **{{ expression }}**.

Direttive

Le direttive base di Angular sono:

- **ng-app** - definisce e inizializza l'applicazione Angular nella pagina HTML
- **ng-init** - inizializza dati dell'applicazione.
- **ng-controller** - un elemento JavaScript che espone funzioni
 - consente inizializzazione/modifica variabili
- **ng-model** - collega le variabili dell'applicazione Angular con gli input HTML (input,select,textarea)

Vi sono altre direttive, che vedremo successivamente.

Esempio 1

```
<!DOCTYPE html>
<html>
  <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
  <body>

    <p>Try to change the names.</p>
    <div ng-app="myApp" ng-controller="myCtrl">
      First Name: <input type="text" ng-model="firstName"><br>
      Last Name: <input type="text" ng-model="lastName"><br>
      <br>
      Full Name: {{firstName + " " + lastName}}
    </div>

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

  </body>
</html>
```

Try to change the names.

First Name:

Last Name:

Full Name: John Doe

http://localhost:8080/corsoAngular/pages/00_esempiPresentazione/Esempio1.jsp

Espressioni

Dichiarazione che Angular valuta e inserisce nella pagina HTML il risultato della nostra espressione:

Possono essere scritte in **due modi equivalenti**:

{{ expression }}

ng-bind="expression"

Le espressioni Angular, ricalcano quelle Javascript, e possono contenere:

- valori letterali;
- operatori;
- variabili.

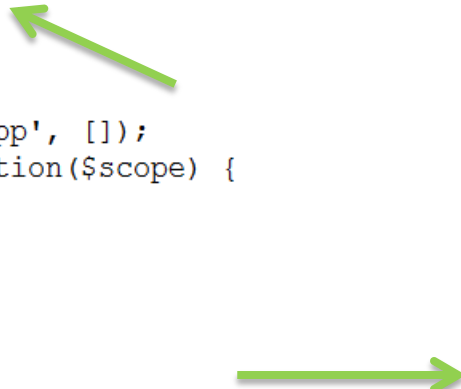
Esempio 1

```
<!DOCTYPE html>
<html>
  <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
  <body>

    <p>Try to change the names.</p>
    <div ng-app="myApp" ng-controller="myCtrl">
      First Name: <input type="text" ng-model="firstName"><br>
      Last Name: <input type="text" ng-model="lastName"><br>
      <br>
      Full Name: {{firstName + " " + lastName}}
    </div>

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

  </body>
</html>
```



Try to change the names.

First Name:

Last Name:

Full Name: John Doe

http://localhost:8080/corsoAngular/pages/00_esempiPresentazione/Esempio1.jsp

External File

è possibile definire i controller all'interno di file **.js** e includerli nell'applicazione.

```

1
2 <!DOCTYPE html>
3 <html>
4   <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
5   <script src="personApp.js"></script>
6   <script src="personController.js"></script>
7 </body>
8
9 <p>Try to change the names. Controller with function in external file</p>
10
11 <div ng-app="myApp" ng-controller="personCtrl">
12
13   First Name: <input type="text" ng-model="firstName"><br>
14   Last Name: <input type="text" ng-model="lastName"><br> <br>
15   Full Name: {{fullNameUpp()}}
16
17 </div>
18
19 <!-- il controller definito come script prima, importato -->
20
21
22
23 </body>
24 </html>
25
26

```

```

1 /**
2  * file di definizione dell'applicazione Angular
3  */
4
5 var app = angular.module('myApp', ['myApp.controllers']);
6 angular.module('myApp.controllers', []);
7

```

```

1 /**
2  * file di definizione del nostro controller
3  */
4
5 angular.module('myApp.controllers')
6 .controller('personCtrl', ['$scope', function($scope) {
7   $scope.firstName = "John";
8   $scope.lastName = "Doe";
9   $scope.fullNameUpp = function() {
10     return $scope.firstName.toUpperCase() + " " + $scope.lastName.toUpperCase();
11   };
12 } ]);

```

http://localhost:8080/corsoAngular/pages/00_esempiPresentazione/Esempio1c.jsp

Riassunto prima lezione

[Link riassunto](http://goo.gl/ChjOEp)
<http://goo.gl/ChjOEp>

Confronto

Javascript

```

24 var persone = [];
25 var aggiungiPersona = function(nome) {
26     if(nome.trim().length > 0){
27         persone.push(nome) ;
28         fLen = persone.length;
29         text = "";
30         for (i = 0; i < fLen; i++) {
31             text += "<tr><td align=\"right\">" + persone[i] + "</td></tr>";
32         }
33         document.getElementById("data").innerHTML = text;
34     }
}

var funzioneRicerca = function(){
    this.rows = document.getElementById('data').rows;
    this.rows_length = rows.length;
    var chiave = document.getElementById('stringaRicerca').value.toUpperCase();
    var row;
    for (var i=0;i< rows_length;i++){
        row = this.rows[i];
        row_text = row.innerHTML;
        row.style.display = ((row_text.indexOf(chiave) != -1) || chiave === '')?'':'none';
    }
};

```

Angular

```

$scope.aggiungiPersona = function() {
    nome = $scope.fullNameUpp();
    if(nome.trim().length > 1){
        $scope.persone.push($scope.fullNameUpp()) ;
    }
}

<tbody id="data">
    <tr ng-repeat="x in persone | filter : filtro">
        <td>{{ x }}</td>
    </tr>

```

http://localhost:8080/corsoAngular/pages/01_esempioConfronto/Javascript/Esempio1dJavascript.jsp

http://localhost:8080/corsoAngular/pages/01_esempioConfronto/Angular/Esempio1dAngular.jsp

Controller

I controller Angular:

- gestiscono i dati nelle applicazioni;
- sono scritti in JavaScript.

Per creare un controller occorre:

- associarlo al modulo

`angular.module(<nomeModulo>)`

- definirlo all'interno di un script js con la sintassi:

`.controller(<nome>, ['$scope', function($scope) { ...corpo controller ... }]);`

Il nostro controller va richiamato nella pagina con cui interagisce

```
angular.module('corsoBaseAngular.controllers')  
  
  .controller('MyCtrl', [ '$scope', function($scope) {  
    $scope.counter = 3;  
    $scope.saluto = 'hello Angular!';  
  
  } ]);
```

```
<div ng-controller="MyCtrl">  
  <h2>{{saluto}}</h2>  
  <h3>{{counter+1}}</h3>  
</div>  
<div ng-controller="MyCtrl">  
  <h2>{{saluto}}</h2>  
  <h3>{{counter+5}}</h3>  
</div>
```

http://localhost:8080/corsoAngular/pages/02_controller/BasicController.jsp

Data Binding

Se nella pagina si dichiara più volte l'uso della stessa variabile, il framework mantiene sempre il dato aggiornato indipendentemente sia modificata da un punto o un altro.

Avviene un **digest loop**, ovvero un ciclo di risoluzioni delle modifiche, che termina quando non ci sono più aggiornamenti da eseguire a seguito dell'evento iniziale.

Nel nostro caso la modifica di una casella di testo avvia un processo di valutazioni legata a ciascuna dichiarazione della variabile nella pagina e nei controller.

Nel nostro esempio termina con l'aggiornamento dell'altro input e della valutazione di uguaglianza

```
<input type="text" ng-model="username"
      class="form-control"
      placeholder="Your first name here...">
```

```
<input type="text" ng-model="username"
      class="form-control"
      placeholder="... or here">
```

```
<h4>Who are you: {{username}}</h4>
```

```
<h2 ng-show="username === 'mario'">
  Hey Mario! What are you doing?
```

Binding Example

Who are you: mario

Hey Mario! What are you doing?

http://localhost:8080/corsoAngular/pages/03_databinding/DataBinding2.jsp

Direttive (2)

Altre direttive comunemente usate di Angular sono:

- **ng-show** - visualizza o nasconde un elemento HTML sulla base di una espressione passata alla direttiva.
- **ng-hide** - come il precedente ma con comportamento opposto
- **ng-if** - **rimuove** dal DOM della pagina l'elemento
- **ng-disabled** - imposta un elemento come disabilitato o meno
- **ng-repeat** - per ciascun elemento della collezione passata, inserisce un'istanza del tag HTML in cui è dichiarata la direttiva stessa

Direttive (3)

- **ng-click** - definisce il comportamento da associare all'evento onclick dell'elemento
- **ng-href** - ridefinisce l'attributo *href* del tag `<a>` rendendolo dinamico
- **ng-options** - rende dinamica la creazione dei tag option in un elemento select
- **ng-switch** - permette, in una sezione di elementi HTML, di mostrare quello che rende vera una condizione (da utilizzare con `ng-switch-when` e `ng-switch-default`)
- **ng-style** - specifica degli attributi CSS per l'elemento

http://localhost:8080/corsoAngular/pages/04_direttivePrincipali/04_direttivePrincipali.jsp

Scope

Se consideriamo un'applicazione Angular composta da:

- *View*, che è il codice HTML;
- *Model*, sono i dati disponibili per la view -> lo **\$scope** è il **Model**;
- *Controller*, sono le funzionalità JavaScript che gestiscono i dati.

Lo **\$scope** è un oggetto fornito dal framework con proprietà e metodi, che sono raggiungibili sia dalla View sia dal Controller.

Usiamo gli scope locali, non la `$rootScope`:

each inner piece of the repeat, gets it's own copy of the element that it repeats.

`$rootScope` exists, but it can be used for evil

In una applicazione possono esserci più scope, definiti in maniera trasparente in una struttura gerarchica che ricalca la struttura dei nodi dell'HTML della pagina (es:annidamenti).

I componenti possono scambiarsi informazioni attraverso variabili dello stesso scope oppure attraverso elementi condivisi.

Esistono anche dei «segnali» (vedremo più avanti)

http://localhost:8080/corsoAngular/pages/05_scope/Login1_separati.jsp

http://localhost:8080/corsoAngular/pages/05_scope/Login2_annidati.jsp

http://localhost:8080/corsoAngular/pages/05_scope/Login3_comunicanti.jsp

http://localhost:8080/corsoAngular/pages/05_scope/Login3_comunicantiScope.jsp

Validazione

Angular fornisce anche gli strumenti per la validazione dei dati lato client.

Modificando il dato nel campo di una form Angular:

- permette di creare delle notifiche della correttezza dei dati al momento inseriti;
- tiene anche traccia se un campo è stato selezionato, modificato o nulla.

I campi Input, hanno i seguenti stati che possono valere *vero* o *falso*:

- **\$valid** il contenuto del campo è valido
- **\$invalid** il contenuto del campo NON è valido
- **\$untouched** il campo non è stato toccato (focus)
- **\$touched** il campo è stato toccato (unfocus)
- **\$pristine** il campo non è stato modificato (lett.: incontaminato, puro)
- **\$dirty** il campo è stato modificato

Le intere form che li contengono, possono a loro volta avere i seguenti stati (*vero* o *falso*):

- **\$valid** tutto il contenuto della form è valido
- **\$invalid** il contenuto della form è non valido
- **\$pristine** nessun campo è stato al momento modificato
- **\$dirty** uno o più campi contenuti sono stati modificati
- **\$submitted** la form è stata inviata

Angular permette di creare delle direttive per funzioni di validazione custom.

http://localhost:8080/corsoAngular/pages/06_validazione/validazione.jsp

Submit

In Angular è possibile definire la submit di un form in due modi:

- tramite la direttiva **ng-submit** nella definizione della **form**

```
<form ng-submit="myFunc()">
  <input type="text">
  <input type="submit">
</form>
```

- tramite la direttiva **ng-click** in un **campo submit** (`input[type=submit]`) della form

```
<form>
  <input type="text">
  <input type="submit" ng-click="myFunc()">
</form>
```

(questa seconda utile se si vogliono avere form differenti)

Per evitare rischi di doppie invocazioni, utilizzare solo una delle due opzioni.

http://localhost:8080/corsoAngular/pages/07_submit/submit.jsp

Filters e Formatter

Angular mette a disposizione diverse funzionalità per interagire velocemente con gli elementi contenuti nel DOM della pagina

- **ng-repeat** : per popolare una tabella data una lista di elementi
`<tr ng-repeat="x in users"`

A cui è possibile definire dei filtri da applicare alla lista:

- **filter** per mantenere gli elementi che rendono vera la condizione
- **orderBy** per riordinare gli elementi

Es.: `<tr ng-repeat="x in users | filter : varRuolo | orderBy: varOrdine">`

- Formattazione **UPPERCASE** / **lowercase**:

Es.: `<td>{{ x.role | uppercase }}</td>`

- Formattazione di una cifra tramite **currency**:

`{{ currency_expression | currency : symbol : fractionSize }}`

Es.: `<td>{{ x.salary | currency:"€":1 }}</td>`

http://localhost:8080/corsoAngular/pages/08_filter/filter1.jsp

CUSTOM DIRECTIVES

Angular fornisce la possibilità di creare delle direttive personalizzate.
Il costrutto base per crearne una è:

.directive

Per ogni direttiva custom, possono essere definite le seguenti proprietà

proprietà	descrizione
restrict	definisce quando la direttiva può essere usata (E - elemento, A - attributo, C - classe CSS, M - commento).
scope	usato per creare un nuovo scope di un elemento annidato o uno scope isolato.
template	il contenuto da porre in output. Può essere composto da HTML, data binding di espressioni e altre direttive.
templateUrl	url del template usato per la direttiva.
controller	per creare un controller apposito per la direttiva
link	funzione per la direttiva, vede il suo scope
transclude	inverte lo scope della direttiva, facendole accedere ai dati dello scope padre

es:

```
.directive('myCustomer', function() {
  return {
    template: 'Name: {{customer.name}} Address: {{customer.address}}'
  };
});
```

http://localhost:8080/corsoAngular/pages/09_direttive/submit_direttive.jsp

PROMISES

Gli oggetti **Promise** servono per creare e gestire funzionalità in modalità asincrona. Una Promise rappresenta un'operazione che non è ancora completata, ma lo sarà in futuro.

\$q è un servizio nativo del framework Angular per gestire questo tipo di richieste ed è utilizzabile in tutta la nostra applicazione attraverso la dependency injection.

Vi sono due metodologie per implementare le promise di questo tipo:

- `return $q(function(resolve, reject) { [...]})`
- `$q.defer()` attraverso le funzioni della **Deferred API** (negli esempi usata questa)

Deferred è l'oggetto che rappresenta il task e che sarà completato, e la sua API fornisce le funzioni:

- `resolve(value)` - risolve la promise
- `reject(reason)` - la respinge
- `notify(value)` - fornisce un aggiornamento

Una promise viene creata tramite la chiamata: `deferred.promise`.

La gestione dei casi della promise viene tramite i metodi:

- `promise.then()`
- `.catch()`
- `.finally()`

http://localhost:8080/corsoAngular/pages/10_promises/submit_promises.jsp

PROMISES 2

Si possono unire diverse promise: **`$q.all(promises)`**;
il risultato è a sua volta una promise che viene risolta quando tutte quelle che la compongono sono risolte.

```
var promises = [];  
promises.push(promise1); //e altre  
$q.all(promises).then( ).catch( ).finally( )
```

Nel la promise composta fallisce quando fallisce la prime delle sue componenti.

Nel caso `.then(` si può definire una funzione che gestisce un array di risposte fornite dalle varie chiamate:
function completed(**result**) {...}

http://localhost:8080/corsoAngular/pages/10_promises/submit_promisesAll.jsp

Eventi: emit e broadcast

Sono due tipi di segnali che si possono inviare tra elementi annidati negli scope della nostra applicazione:

- dal padre a tutti i figli -> broadcast
- da un figlio verso il padre -> emit

Per lanciare un segnale di questi tipi, occorre definire una funzione che dia un nome all'evento e la direzione in cui propagarlo:

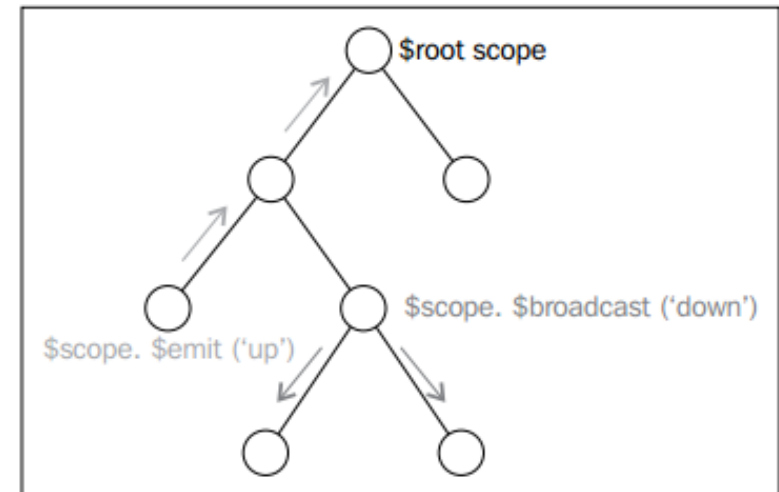
```
$scope.chiamataBroadcastFiglio = function() {  
  $scope.$broadcast('eventoBroadcast');  
};
```

```
$scope.chiamataEmitFiglio = function() {  
  $scope.$emit('eventoEmit');  
};
```

E un'altra funzione per definire nell'elemento destinatario cosa debba essere fatto:

```
$scope.$on('eventoEmit', function() {  
  console.log('EMIT FIGLIO: Sono stato intercettato');  
});
```

```
$scope.$on('eventoBroadcast', function() {  
  console.log('BROADCAST FIGLIO: Sono stato intercettato');  
});
```



http://localhost:8080/corsoAngular/pages/11_events/eventiAnnidati.jsp

Eventi: watchers

Un watcher è un elemento che viene invocato ogni volta che si verifica una condizione.

Un modo semplice per la creazione di un watcher è tramite l'istruzione :

```
$watch(watchExpression, listener);
```

che registra una listener callback (una funzione) in attesa di essere eseguita ogniqualvolta la *watchExpression* cambia.

Dove *listener* è una funzione che può avere i parametri:

```
function(newVal, oldVal [, scope])
```

dove:

newVal: contiene il valore corrente della *watchExpression*

oldVal: contiene il valore precedente della *watchExpression*

[*Scope*: si riferisce al current scope].

La *watchExpression* viene valutata ad ogni cambiamento innescato nella digest loop: attenzione in quanto la funzione può essere invocata più volte a seguito della stessa input iniziale.

http://localhost:8080/corsoAngular/pages/12_watcher/submit.jsp

Routing in AngularJs

Definizioni Base

Routing

Associazione di una vista/pagina ad un URL

<http://www.html.it/pag/53894/single-page-application-e-routing-in-angularjs/>

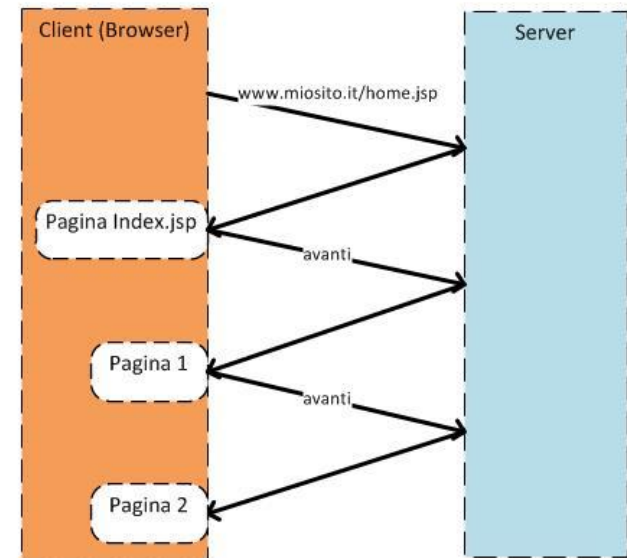
SPA (Single Page Application)

Con Single-page Application si intende una applicazione web o un sito web che può essere usato o consultato su una singola pagina web con l'obiettivo di fornire una esperienza utente più fluida e simile alle applicazioni Desktop dei sistemi operativi tradizionali.

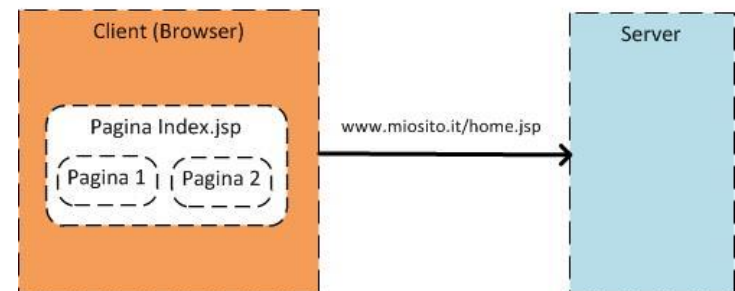
Single Page Application

- Tutto il codice necessario (HTML, JavaScript e CSS) e le relative risorse (immagini, etc) sono recuperate in un unico caricamento e aggiunte alla pagina quando necessario, di solito in risposta ad azioni dell'utente
- Permette la creazione di applicazioni responsive
- Diminuzione dei tempi di attesa

Applicazione Classica



Applicazione SPA



SPA in AngularJs

1. Importare la libreria nativa angular-route

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular-route.js"></script>
```

2. Dichiarare il modulo ngRoute nel contesto Angular

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

3. Utilizzare i costrutti ng-view e \$routeProvider forniti da Angular

Esempio: <http://localhost:8080/corsoAngular/pages/14 angularRouting/Index.jsp>

Chiamare servizi di Back-end con Angularjs

Angularjs Service

1. Componenti che offrono funzionalità che non dipendono dall'interfaccia (lo scope dei controller dipende dalla pagina, quello dei service no)
2. Hanno scope Singleton
3. Utilizzabili per condividere funzionalità tra più componenti (controller, direttive custom, etc)
4. E' possibile definirli come Service, Factory o Provider, la differenza è solamente sintattica

```
.factory('salutaService', [function() {  
    return {  
        saluta : function(nome) {  
            return 'Ciao ' + nome ;  
        }  
    };  
}]);
```

Il costrutto \$http

1. Consente di effettuare chiamate Ajax al server e di gestirne le risposte
2. Espone i seguenti metodi per effettuare chiamate http

```
$http.get(url)  
$http.post(url, dati)  
$http.put(url, dati)  
$http.delete(url)  
$http.head(url)
```

} Permettono il passaggio di dati (parametri al server)

3. Restituiscono tutti una promise che espone i metodi success() ed error() utili alla gestione della risposta del server

```
.factory('ricercaService', [function() {  
  return {  
    ricerca : function(nome,cognome) {  
      var requestForm = {  
        nome: nome,  
        cognome: cognome  
      };  
      return $http.post('www.servizidiprova.it/ricerca', requestForm);  
    }  
  };  
}]);
```

Esempio: http://localhost:8080/corsoAngular/pages/15_costruttoHttp/CostruttoHttp.jsp

I servizi di back-end...

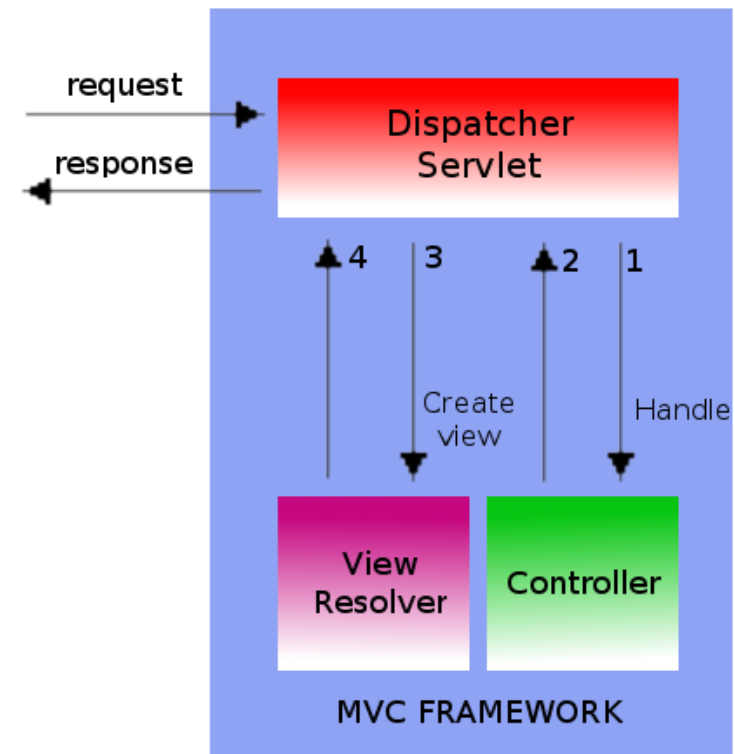
Spring MVC

E' un framework fornito da Spring, che permette di:

- mappare metodi e classi Java con determinati url
- restituire al client risorse presenti lato server
- restituire al client le viste (pagine web)

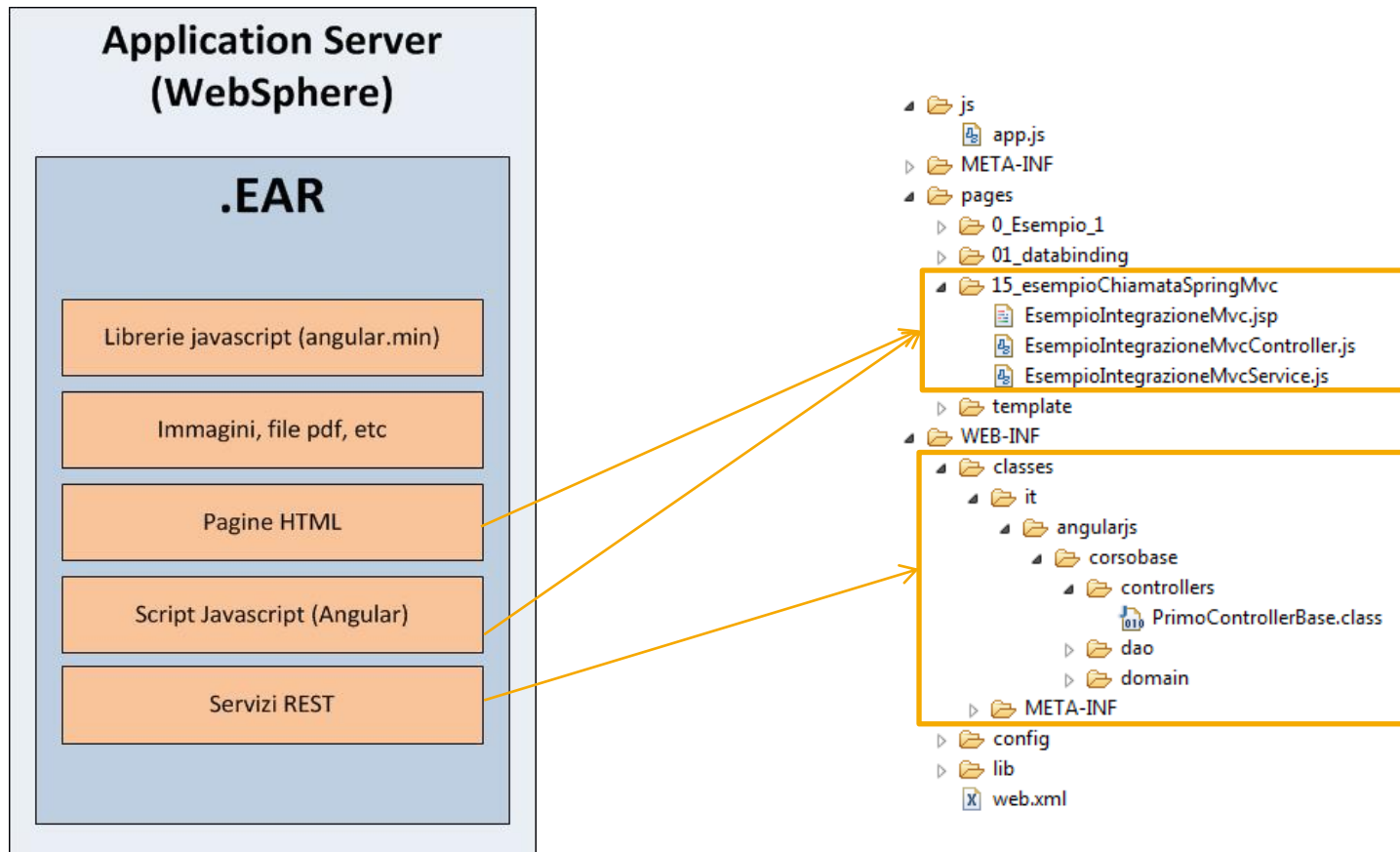
Il framework applica il pattern nel seguente modo:

- **Model** → sono gli oggetti utilizzati per il passaggio dei dati
- **Controller** → sono rappresentati da classi (chiamate Controller) che espongono metodi richiamati attraverso specifici URL
- **View** → sono le pagine web



Spring MVC – utilizzo in applicazioni AngularJs

Viene utilizzato per poter recuperare dal client le risorse presenti sul server (all'interno dell'EAR/WAR)



Configurazione Spring MVC – Web.xml

```
<servlet>
  <servlet-name>dispatcherServlet</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/config/mvc-application-config.xml</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>dispatcherServlet</servlet-name>
  <url-pattern>/</url-pattern>
</servlet-mapping>
```

Qualsiasi request viene «intercettata»
dalla DispatcherServlet

Configurazione Spring MVC - Context

```
<bean id="jsonMessageConverter"
  class="org.springframework.http.converter.json.MappingJackson2HttpMessageConverter">
  <property name="objectMapper">
    <bean
      class="org.springframework.http.converter.json.Jackson2ObjectMapperFactoryBean">
      <property name="featuresToDisable"
        value="#{T(com.fasterxml.jackson.databind.DeserializationFeature).FAIL_ON_UNKNOWN_PROPERTIES}" />
      </bean>
    </property>
  </bean>
</bean>
```

Configurazione «base» per lo scambio dei messaggi JSON

```
<mvc:annotation-driven>
  <mvc:message-converters register-defaults="true">
    <ref bean="jsonMessageConverter" />
  </mvc:message-converters>
</mvc:annotation-driven>
```

```
<mvc:resources mapping="/js/**" location="/js/" />
<mvc:resources mapping="/pages/**" location="/pages/" />
<mvc:resources mapping="/resources/**" location="/resources/" />
<mvc:resources mapping="/webjars/**"
  location="classpath:/META-INF/resources/webjars/" />
```

Configurazione per rendere disponibili le risorse al client

I Messaggi JSON (JavaScript Object Notation)

Standard **universale** per lo scambio dati in applicazioni client-server

Obj Javascript

```
$scope.persona = {  
  "nome" : "Giuseppe",  
  "cognome" : "Giordano"  
};
```

Rappresentazione JSON

```
{  
  "nome" : "Giuseppe",  
  "cognome" : "Giordano"  
}
```

Obj Java

```
public class Persona {  
    private String nome;  
    private String cognome;
```

Recupero Risorse dal client

Recupero librerie AngularJs

PRIMA

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
```

DOPO

```
<script src="{pageContext.request.contextPath}/webjars/angularjs/1.5.5/angular.min.js"></script>
```

Recupero altre risorse (file pdf, immagini, etc)

Esempio: <http://localhost:8080/corsoAngular/resources/FileTestProva.txt>

Servizi REST - Esempio

```
@RequestMapping(value = "/cercaLibri")
@ResponseBody
public List<Libro> cercaLibri(@RequestParam String autore) {

    List<Libro> resultList = cercaLibro(autore);

    return resultList;
}
```

Esempio: <http://localhost:8080/corsoAngular/cercaLibri?autore=pippo>

Servizi REST - Spring MVC Annotation

@Controller

@RequestMapping

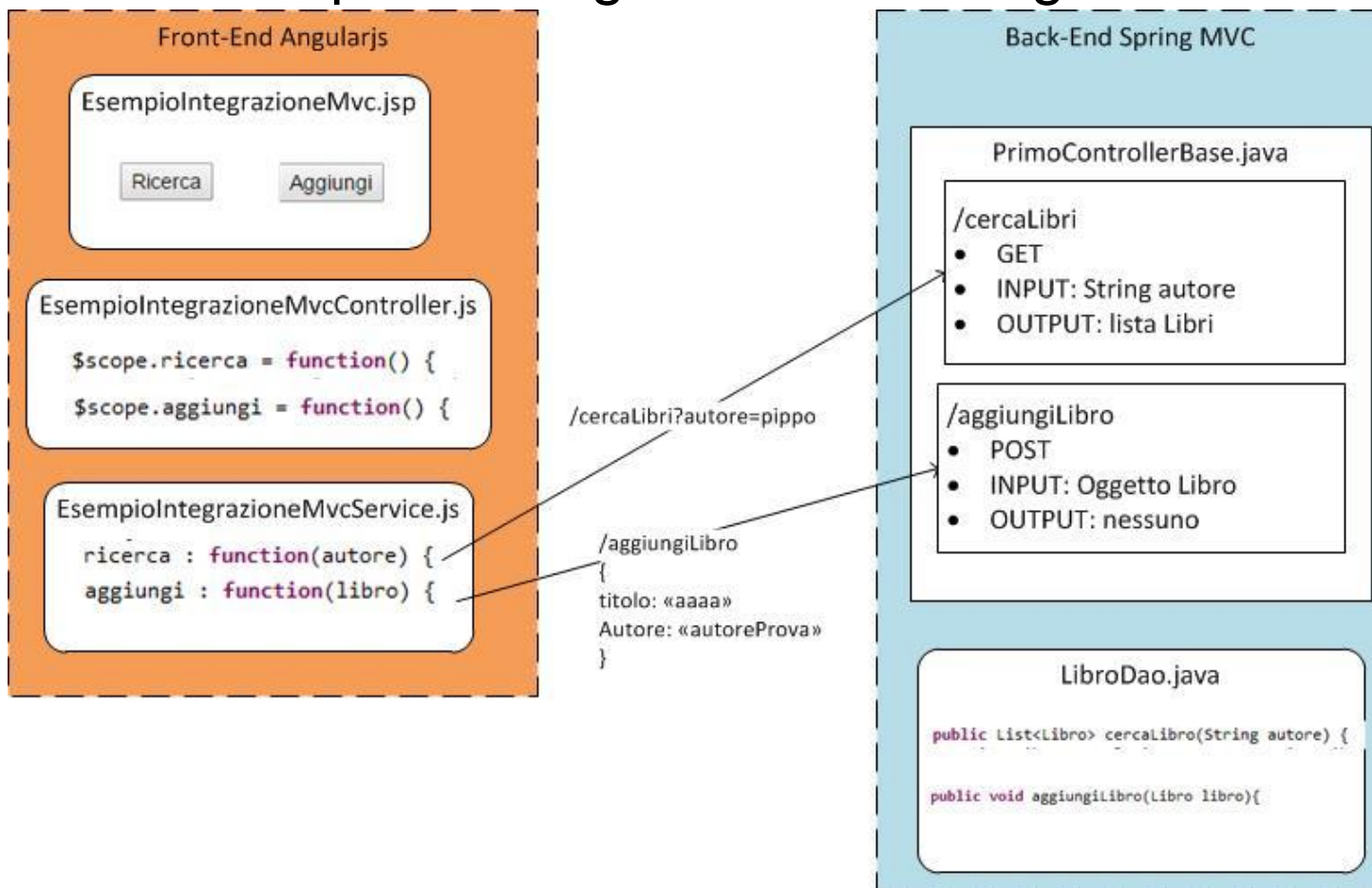
@PathVariable

@RequestParam

@RequestBody

@ResponseBody

Esempio di integrazione con Angular



Esempio: http://localhost:8080/corsoAngular/pages/16_esempioChiamataSpringMvc/EsempioIntegrazioneMvc.jsp

Struttura del progetto utilizzato per gli esempi (1/2)

Servizi

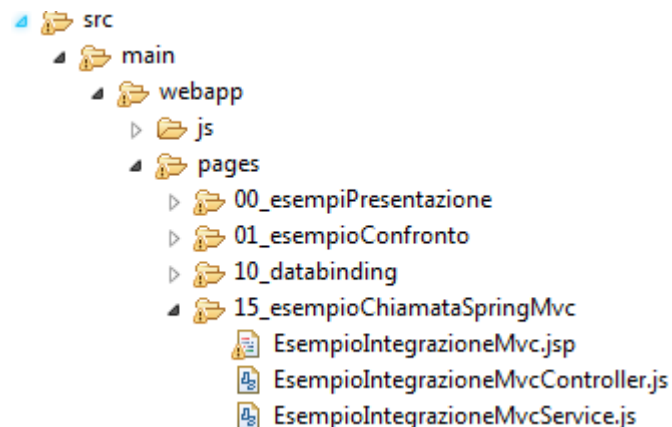
- src/main/java
 - it.angularjs.corsobase
 - controllers
 - PrimoControllerBase.java
 - dao
 - LibroDao.java
 - domain
 - Libro.java

Webjar e Altre Librerie

- Maven Dependencies
 - spring-webmvc-3.2.2.RELEASE.jar - D:\Use
 - spring-beans-3.2.2.RELEASE.jar - D:\Users\
 - spring-context-3.2.2.RELEASE.jar - D:\User:
 - spring-aop-3.2.2.RELEASE.jar - D:\Users\ar
 - spring-core-3.2.2.RELEASE.jar - D:\Users\ai
 - commons-logging-1.1.1.jar - D:\Users\an
 - spring-expression-3.2.2.RELEASE.jar - D:\U
 - spring-web-3.2.2.RELEASE.jar - D:\Users\ar
 - jstl-1.2.jar - D:\Users\an50011\m2\reposit
 - servlet-api-2.5.jar - D:\Users\an50011\m2'
 - jsp-api-2.2.jar - D:\Users\an50011\m2\rep
 - jstl-api-1.2.jar - D:\Users\an50011\m2\rep
 - jstl-impl-1.2.jar - D:\Users\an50011\m2\re
 - el-api-2.2.jar - D:\Users\an50011\m2\repc
 - jackson-databind-2.2.3.jar - D:\Users\an50
 - jackson-annotations-2.2.3.jar - D:\Users\ar
 - jackson-core-2.2.3.jar - D:\Users\an50011\
 - bootstrap-2.3.2.jar - D:\Users\an50011\m
 - jquery-1.9.0.jar - D:\Users\an50011\m2\re
 - angular-ui-bootstrap-0.11.0.jar - D:\Users\
 - angularjs-1.5.5.jar - D:\Users\an50011\m2
 - spring-jdbc-3.2.2.RELEASE.jar - D:\Users\ai
 - spring-tx-3.2.2.RELEASE.jar - D:\Users\an5
 - aopalliance-1.0.jar - D:\Users\an50011\m
 - spring-context-support-3.2.2.RELEASE.jar -
 - commons-dbcp-1.4.jar - D:\Users\an5001
 - commons-pool-1.5.4.jar - D:\Users\an500

Struttura del progetto utilizzato per gli esempi (2/2)

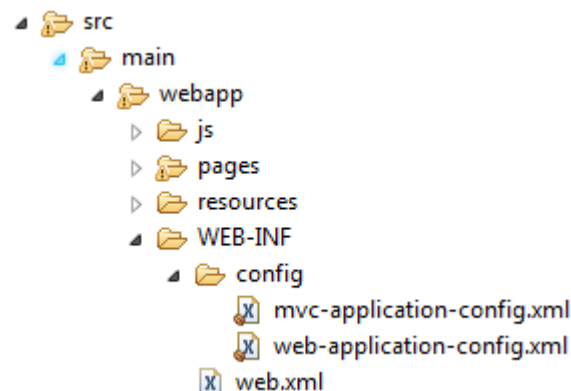
Pagine HTML e Angular



Resources



Configurazione



Distribuzione del software – File .ear (1/3)

```
<build>
  <finalName>myFirstApplication</finalName>
  <plugins>
    <plugin>
      <artifactId>maven-ear-plugin</artifactId>
      <version>2.6</version>
      <configuration>
        <applicationName>myFirstApplication</applicationName>
        <modules>

          <webModule>
            <groupId>com.arca.training.angularJS</groupId>
            <artifactId>myFirstApplication-web</artifactId>
            <contextRoot>/myFirstApplication</contextRoot>
          </webModule>

        </modules>
      </configuration>
    </plugin>
  </plugins>
</build>
```

Il plugin viene lanciato attraverso il comando **mvn package**

Esempio

Distribuzione del software – File .ear (2/3)

Servizi

- ▲ CorsoAngularJsAndSpringMVC-0.0.1-SNAPSHOT
 - ▷ js
 - ▷ META-INF
 - ▷ pages
 - ▲ WEB-INF
 - ▲ classes
 - ▲ it
 - ▲ angularjs
 - ▲ corsobase
 - ▲ controllers
 - PrimoControllerBase.class
 - ▲ dao
 - LibroDao.class
 - LibroDao\$1.class
 - ▲ domain
 - Libro.class

Webjar e Altre Librerie

- ▲ CorsoAngularJsAndSpringMVC-0.0.1-SNAPSHOT
 - ▷ js
 - ▷ META-INF
 - ▷ pages
 - ▲ WEB-INF
 - ▷ classes
 - ▷ config
 - ▲ lib
 - angular-ui-bootstrap-0.11.0.jar
 - angularjs-1.5.5.jar
 - aopalliance-1.0.jar
 - bootstrap-2.3.2.jar
 - commons-dbcp-1.4.jar
 - commons-logging-1.1.1.jar
 - commons-pool-1.5.4.jar
 - jackson-annotations-2.2.3.jar
 - jackson-core-2.2.3.jar
 - jackson-databind-2.2.3.jar
 - jquery-1.9.0.jar
 - jstl-1.2.jar
 - jstl-api-1.2.jar
 - jstl-impl-1.2.jar
 - spring-aop-3.2.2.RELEASE.jar
 - spring-beans-3.2.2.RELEASE.jar
 - spring-context-3.2.2.RELEASE.jar
 - spring-context-support-3.2.2.RELEASE.jar
 - spring-core-3.2.2.RELEASE.jar
 - spring-expression-3.2.2.RELEASE.jar
 - spring-jdbc-3.2.2.RELEASE.jar
 - spring-tx-3.2.2.RELEASE.jar
 - spring-web-3.2.2.RELEASE.jar
 - spring-webmvc-3.2.2.RELEASE.jar

Distribuzione del software – File .ear (3/3)

Pagine HTML e Angular

- ▲ CorsoAngularJsAndSpringMVC-0.0.1-SNAPSHOT
 - ▷ js
 - ▷ META-INF
 - ▲ pages
 - ▷ 0_Esempio_1
 - ▷ 01_databinding
 - ▲ 15_esempioChiamataSpringMvc
 - EsempioIntegrazioneMvc.jsp
 - EsempioIntegrazioneMvcController.js
 - EsempioIntegrazioneMvcService.js

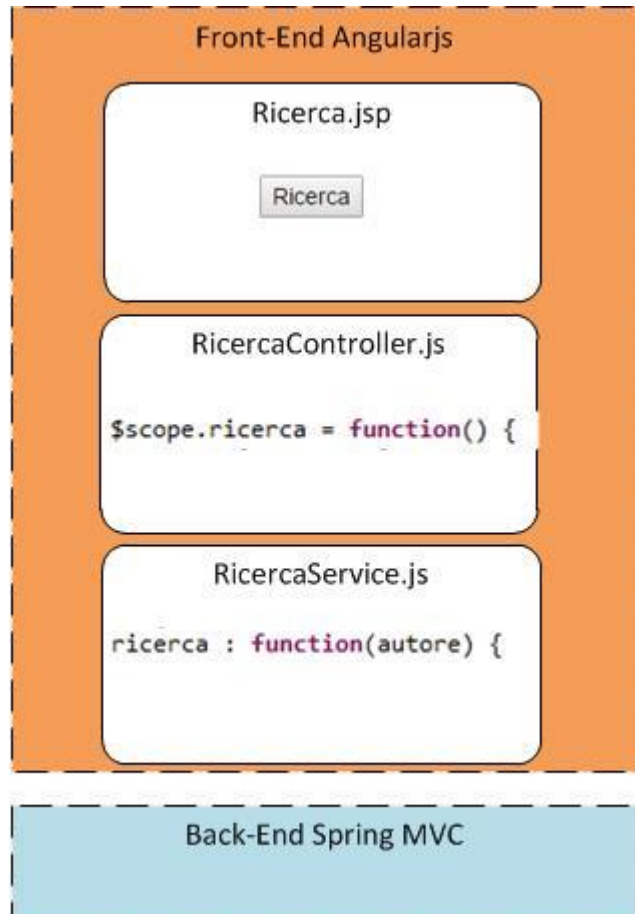
Resources

- ▲ CorsoAngularJsAndSpringMVC-0.0.1-SNAPSHOT
 - ▷ js
 - ▷ META-INF
 - ▷ pages
 - ▲ resources
 - FileTestProva.txt

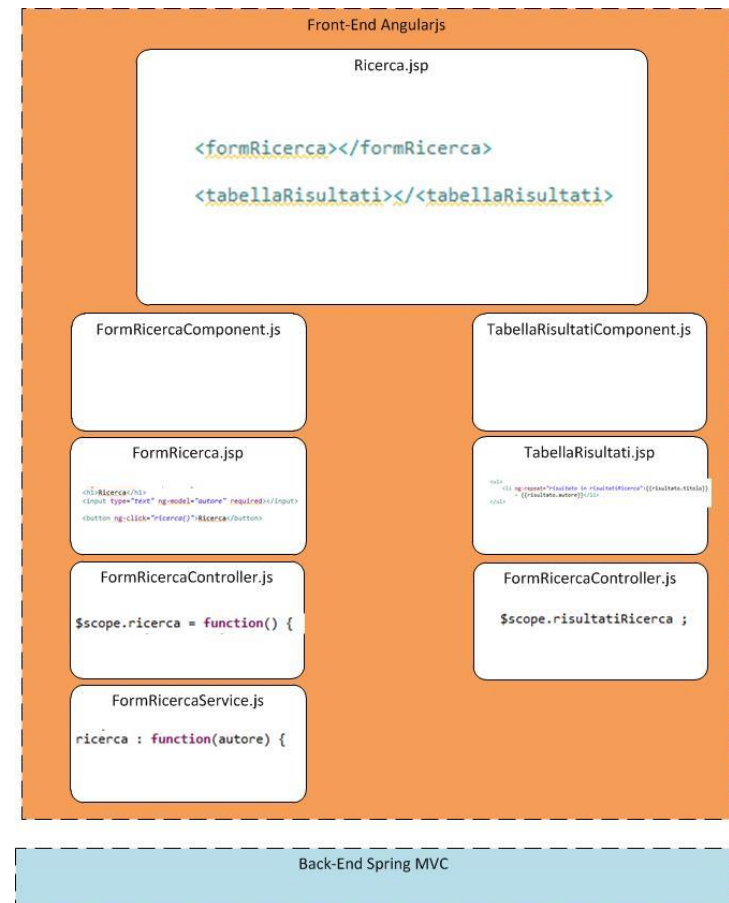
Configurazione

- ▲ CorsoAngularJsAndSpringMVC-0.0.1-SNAPSHOT
 - ▷ js
 - ▷ META-INF
 - ▷ pages
 - ▷ resources
 - ▲ WEB-INF
 - ▷ classes
 - ▲ config
 - mvc-application-config.xml
 - web-application-config.xml
 - ▷ lib
 - web.xml

Utilizzo di AngularJs con «componenti»



VS



AngularJs VS JSF

JSF	AngularJs
Lo stato e il valore delle variabili è mantenuto lato server (Backing Bean)	Lo stato e il valore delle variabili è mantenuto lato client (Controller AngularJs)
Continua iterazione con il server (getter and setter method)	Iterazione del server solamente per risorse/chiamata a servizi
Vincolato all'utilizzo del linguaggio Java	Qualsiasi linguaggio può essere utilizzato per implementare la parte server, purché esponga servizi REST

```
<h:panelgrid columns="2">
  <h:outputText value="Your name:" />
  <h:input id="commentatorsNameID" type="text"
    value="#{commentBean.yourName}"
    placeholder="Anonymous" />
  <h:outputText value="Your website:" />
  <h:input id="commentatorsWebSiteID" type="text"
    value="#{commentBean.website}"
    placeholder="advertise your website here"/>
  .
</h:panelgrid>
```

```
<table>
  <tr>
    <td>Your name:</td>
    <td>
      <input id="commentatorsNameID" type="text"
        ng-model="yourName"
        placeholder="Anonymous" />
    </td>
  </tr>
  <tr>
    <td>Your website:</td>
    <td>
      <input id="commentatorsWebSiteID" type="text"
        ng-model="website"
        placeholder="advertise your website here"/>
    </td>
  </tr>
</table>
```




Via Morgagni, 22
Tel. +39 045 8282711
Fax +39 045 8282712
37135 Verona

www.addvalue.it