EBook Gratuito

APPRENDIMENTO AngularJS

Free unaffiliated eBook created from
Stack Overflow contributors.

#angularjs

Sommario

Di

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: angularjs

It is an unofficial and free AngularJS ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official AngularJS.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capitolo 1: Iniziare con AngularJS

Osservazioni

AngularJS è un framework di applicazioni Web progettato per semplificare lo sviluppo di applicazioni client avanzate. Questa documentazione è per Angular 1.x , il predecessore del più moderno Angular 2 o consultare la documentazione Stack Overflow per Angular 2 .

Versioni

Versione	Data di rilascio
1.6.5	2017/07/03
1.6.4	2017/03/31
1.6.3	2017/03/08
1.6.2	2017/02/07
1.5.11	2017/01/13
1.6.1	2016/12/23
1.5.10	2016/12/15
1.6.0	2016/12/08
1.6.0-rc.2	2016/11/24
1.5.9	2016/11/24
1.6.0-rc.1	2016/11/21
1.6.0-rc.0	2016/10/26
1.2.32	2016/10/11
1.4.13	2016/10/10
1.2.31	2016/10/10
1.5.8	2016/07/22
1.2.30	2016/07/21
1.5.7	2016/06/15

Versione	Data di rilascio
1.4.12	2016/06/15
1.5.6	2016/05/27
1.4.11	2016/05/27
1.5.5	2016/04/18
1.5.4	2016/04/14
1.5.3	2016/03/25
1.5.2	2016/03/19
1.4.10	2016/03/16
1.5.1	2016/03/16
1.5.0	2016/02/05
1.5.0-rc.2	2016/01/28
1.4.9	2016/01/21
1.5.0-rc.1	2016/01/16
1.5.0-rc.0	2015/12/09
1.4.8	2015/11/20
1.5.0-beta.2	2015/11/18
1.4.7	2015/09/30
1.3.20	2015/09/30
1.2.29	2015/09/30
1.5.0-beta.1	2015/09/30
1.5.0-beta.0	2015/09/17
1.4.6	2015/09/17
1.3.19	2015/09/17
1.4.5	2015/08/28
1.3.18	2015/08/19

Versione	Data di rilascio
1.4.4	2015/08/13
1.4.3	2015/07/15
1.3.17	2015/07/07
1.4.2	2015/07/07
1.4.1	2015/06/16
1.3.16	2015/06/06
1.4.0	2015/05/27
1.4.0-rc.2	2015/05/12
1.4.0-rc.1	2015/04/24
1.4.0-rc.0	2015/04/10
1.3.15	2015/03/17
1.4.0-beta.6	2015/03/17
1.4.0-beta.5	2015/02/24
1.3.14	2015/02/24
1.4.0-beta.4	2015/02/09
1.3.13	2015/02/09
1.3.12	2015/02/03
1.4.0-beta.3	2015/02/03
1.3.11	2015/01/27
1.4.0-beta.2	2015/01/27
1.4.0-beta.1	2015/01/20
1.3.10	2015/01/20
1.3.9	2015/01/15
1.4.0-beta.0	2015/01/14
1.3.8	2014/12/19

Versione	Data di rilascio
1.2.28	2014/12/16
1.3.7	2014/12/15
1.3.6	2014/12/09
1.3.5	2014/12/02
1.3.4	2014/11/25
1.2.27	2014/11/21
1.3.3	2014/11/18
1.3.2	2014/11/07
1.3.1	2014/10/31
1.3.0	2014/10/14
1.3.0-rc.5	2014/10/09
1.2.26	2014/10/03
1.3.0-rc.4	2014/10/02
1.3.0-rc.3	2014/09/24
1.2.25	2014/09/17
1.3.0-rc.2	2014/09/17
1.2.24	2014/09/10
1.3.0-rc.1	2014/09/10
1.3.0-rc.0	2014/08/30
1.2.23	2014/08/23
1.3.0-beta.19	2014/08/23
1.2.22	2014/08/12
1.3.0-beta.18	2014/08/12
1.2.21	2014/07/25
1.3.0-beta.17	2014/07/25

Versione	Data di rilascio
1.3.0-beta.16	2014/07/18
1.2.20	2014/07/11
1.3.0-beta.15	2014/07/11
1.2.19	2014/07/01
1.3.0-beta.14	2014/07/01
1.3.0-beta.13	2014/06/16
1.3.0-beta.12	2014/06/14
1.2.18	2014/06/14
1.3.0-beta.11	2014/06/06
1.2.17	2014/06/06
1.3.0-beta.10	2014/05/24
1.3.0-beta.9	2014/05/17
1.3.0-beta.8	2014/05/09
1.3.0-beta.7	2014/04/26
1.3.0-beta.6	2014/04/22
1.2.16	2014/04/04
1.3.0-beta.5	2014/04/04
1.3.0-beta.4	2014/03/28
1.2.15	2014/03/22
1.3.0-beta.3	2014/03/21
1.3.0-beta.2	2014/03/15
1.3.0-beta.1	2014/03/08
1.2.14	2014/03/01
1.2.13	2014/02/15
1.2.12	2014/02/08

Versione	Data di rilascio
1.2.11	2014/02/03
1.2.10	2014/01/25
1.2.9	2014/01/15
1.2.8	2014/01/10
1.2.7	2014/01/03
1.2.6	2013/12/20
1.2.5	2013/12/13
1.2.4	2013/12/06
1.2.3	2013/11/27
1.2.2	2013/11/22
1.2.1	2013/11/15
1.2.0	2013/11/08
1.2.0-rc.3	2013/10/14
1.2.0-rc.2	2013/09/04
1.0.8	2013/08/22
1.2.0rc1	2013/08/13
1.0.7	2013/05/22
1.1.5	2013/05/22
1.0.6	2013/04/04
1.1.4	2013/04/04
1.0.5	2013/02/20
1.1.3	2013/02/20
1.0.4	2013/01/23
1.1.2	2013/01/23
1.1.1	2012/11/27

Versione	Data di rilascio
1.0.3	2012/11/27
1.1.0	2012/09/04
1.0.2	2012/09/04
1.0.1	2012-06-25
1.0.0	2012-06-14
v1.0.0rc12	2012-06-12
v1.0.0rc11	2012-06-11
v1.0.0rc10	2012-05-24
v1.0.0rc9	2012-05-15
v1.0.0rc8	2012-05-07
v1.0.0rc7	2012-05-01
v1.0.0rc6	2012-04-21
v1.0.0rc5	2012-04-12
v1.0.0rc4	2012-04-05
v1.0.0rc3	2012-03-30
v1.0.0rc2	2012-03-21
g3-v1.0.0rc1	2012-03-14
g3-v1.0.0-RC2	2012-03-16
1.0.0rc1	2012-03-14
0.10.6	2012-01-17
0.10.5	2011-11-08
0.10.4	2011-10-23
0.10.3	2011-10-14
0.10.2	2011-10-08
0.10.1	2011-09-09

Versione	Data di rilascio
0.10.0	2011-09-02
0.9.19	2011-08-21
0.9.18	2011-07-30
0.9.17	2011-06-30
0.9.16	2011-06-08
0.9.15	2011-04-12
0.9.14	2011-04-01
0.9.13	2011-03-14
0.9.12	2011-03-04
0.9.11	2011-02-09
0.9.10	2011-01-27
0.9.9	2011-01-14
0.9.7	2010-12-11
0.9.6	2010-12-07
0.9.5	2010-11-25
0.9.4	2010-11-19
0.9.3	2010-11-11
0.9.2	2010-11-03
0.9.1	2010-10-27
0.9.0	2010-10-21

Examples

Iniziare

Crea un nuovo file HTML e incolla il seguente contenuto:

```
<!DOCTYPE html>
<html ng-app>
```

```
<head>
  <title>Hello, Angular</title>
  <script src="https://code.angularjs.org/1.5.8/angular.min.js"></script>
</head>
<body ng-init="name='World'">
  <label>Name</label>
  <input ng-model="name" />
  <span>Hello, {{ name }}!</span>
  <p ng-bind="name"></p>
</body>
</html>
```

Dimostrazione dal vivo

Quando apri il file con un browser, vedrai un campo di input seguito dal testo `Hello, World!` . La modifica del valore nell'input aggiornerà il testo in tempo reale, senza la necessità di aggiornare l'intera pagina.

Spiegazione:

1. Carica il framework angolare da una rete di Content Delivery.

   ```
   <script src="https://code.angularjs.org/1.5.8/angular.min.js"></script>
   ```

2. Definire il documento HTML come un'applicazione Angolare con la direttiva `ng-app`

   ```
   <html ng-app>
   ```

3. Inizializza la variabile del `name` usando `ng-init`

   ```
   <body ng-init=" name = 'World' ">
   ```

   Si noti che ng-init deve essere usato solo a scopi dimostrativi e di test. Quando si crea un'applicazione vera e propria, i controller devono inizializzare i dati.

4. Associare i dati dal modello alla vista sui controlli HTML. Associare un `<input>` alla proprietà `name` con `ng-model`

   ```
   <input ng-model="name" />
   ```

5. Mostra contenuti dal modello usando doppie parentesi `{{ }}`

   ```
   <span>Hello, {{ name }}</span>
   ```

6. Un altro modo per legare la proprietà `name` è usare `ng-bind` invece dei manubri `"{{ }}"`

   ```
    <span ng-bind="name"></span>
   ```

Gli ultimi tre passaggi stabiliscono il *collegamento dati bidirezionale* . Le modifiche apportate

all'input aggiornano il *modello* , che si riflette nella *vista* .

C'è una differenza tra l'uso di manubri e `ng-bind` . Se usi i manubri, potresti vedere il vero `Hello, {{name}}` mentre la pagina viene caricata prima che l'espressione sia risolta (prima che i dati siano caricati) mentre se usi `ng-bind` , mostrerà solo i dati quando il nome è risolto. In alternativa, la direttiva `ng-cloak` può essere utilizzata per impedire che i manubri vengano visualizzati prima di essere compilati.

Mostrare tutti i costrutti angolari comuni

L'esempio seguente mostra i comuni costrutti AngularJS in un file:

```
<!DOCTYPE html>
<html ng-app="myDemoApp">
  <head>
    <style>.started { background: gold; }</style>
    <script src="https://code.angularjs.org/1.5.8/angular.min.js"></script>
    <script>
      function MyDataService() {
        return {
          getWorlds: function getWorlds() {
            return ["this world", "another world"];
          }
        };
      }

      function DemoController(worldsService) {
        var vm = this;
        vm.messages = worldsService.getWorlds().map(function(w) {
          return "Hello, " + w + "!";
        });
      }

      function startup($rootScope, $window) {
        $window.alert("Hello, user! Loading worlds...");
        $rootScope.hasStarted = true;
      }

      angular.module("myDemoApp", [/* module dependencies go here */])
        .service("worldsService", [MyDataService])
        .controller("demoController", ["worldsService", DemoController])
        .config(function() {
          console.log('configuring application');
        })
        .run(["$rootScope", "$window", startup]);
    </script>
  </head>
  <body ng-class="{ 'started': hasStarted }" ng-cloak>
    <div ng-controller="demoController as vm">
      <ul>
        <li ng-repeat="msg in vm.messages">{{ msg }}</li>
      </ul>
    </div>
  </body>
</html>
```

Di seguito viene spiegata ogni riga del file:

Dimostrazione dal vivo

1. `ng-app="myDemoApp"` , la direttiva ngApp che avvia l'applicazione e dice angolare che un elemento DOM è controllato da uno specifico `angular.module` denominato `"myDemoApp"` ;
2. `<script src="angular.min.js">` è il primo passo per avviare la libreria AngularJS ;

Vengono dichiarate tre funzioni (`MyDataService` , `DemoController` e `startup`), che vengono utilizzate (e spiegate) di seguito.

3. `angular.module(...)` utilizzato con un array come secondo argomento crea un nuovo modulo. Questo array viene utilizzato per fornire un elenco di dipendenze del modulo. In questo esempio chiamiamo a catena il risultato della funzione `module(...)` ;

4. `.service(...)` crea un servizio angolare e restituisce il modulo per il concatenamento;

5. `.controller(...)` crea un controller angolare e restituisce il modulo per il concatenamento;

6. `.config(...)` Utilizzare questo metodo per registrare il lavoro che deve essere eseguito sul caricamento del modulo.

7. `.run(...)` assicura che il codice venga eseguito all'avvio e assuma una serie di elementi come parametro. Utilizzare questo metodo per registrare il lavoro che deve essere eseguito quando l'iniettore ha finito di caricare tutti i moduli.

    - il primo elemento sta permettendo a Angular di sapere che la funzione di `startup` richiede che il servizio `$rootScope` sia iniettato come argomento;
    - il secondo elemento sta permettendo a Angular di sapere che la funzione di `startup` richiede che il `startup` incorporato `$window` venga iniettato come argomento;
    - l' *ultimo* elemento nell'array, l' `startup` , è la funzione effettiva da eseguire all'avvio;

8. `ng-class` è la direttiva ngClass per impostare una `class` dinamica, e in questo esempio utilizza `hasStarted` sul `$rootScope` modo dinamico

9. `ng-cloak` è una direttiva per impedire che il template html Angolare non restituito (ad esempio " `{{ msg }}` ") venga mostrato brevemente prima che Angular abbia caricato completamente l'applicazione.

10. `ng-controller` è la direttiva che chiede ad Angular di creare un nuovo controller di nome specifico per orchestrare quella parte del DOM;

11. `ng-repeat` è la direttiva per rendere l'iterazione angolare su una raccolta e clonare un modello DOM per ciascun elemento;

12. `{{ msg }}` mostra l' interpolazione : rendering in loco di una parte dell'ambito o del controller;

L'importanza dell'ambito

Poiché Angular utilizza l'HTML per estendere una pagina Web e un semplice Javascript per aggiungere la logica, semplifica la creazione di una pagina Web utilizzando **ng-app** , **ng-controller** e alcune direttive incorporate come **ng-if** , **ng-repeat** , ecc. Con la nuova sintassi

controllerAs , i nuovi utenti Angular possono associare funzioni e dati al proprio controller invece di utilizzare `$scope` .

Tuttavia, prima o poi, è importante capire che cosa sia esattamente questa cosa di `$scope` . Continuerà a mostrarsi negli esempi, quindi è importante avere una certa comprensione.

La buona notizia è che è un concetto semplice ma potente.

Quando crei il seguente:

```
<div ng-app="myApp">
 <h1>Hello {{ name }}</h1>
</div>
```

Dove vive il **nome** ?

La risposta è che Angular crea un oggetto `$rootScope` . Questo è semplicemente un normale oggetto Javascript e quindi **name** è una proprietà `$rootScope` :

```
angular.module("myApp", [])
  .run(function($rootScope) {
    $rootScope.name = "World!";
  });
```

E proprio come con lo scope globale in Javascript, di solito non è una buona idea aggiungere elementi all'ambito globale o `$rootScope` .

Naturalmente, la maggior parte delle volte, creiamo un controller e inseriamo le funzionalità richieste in quel controller. Ma quando creiamo un controller, Angular fa la magia e crea un oggetto `$scope` per quel controller. Questo è a volte indicato come **ambito locale** .

Quindi, creando il seguente controller:

```
<div ng-app="myApp">
  <div ng-controller="MyController">
    <h1>Hello {{ name }}</h1>
  </div>
</div>
```

consentirebbe all'ambito locale di essere accessibile tramite il parametro `$scope` .

```
angular.module("myApp", [])
  .controller("MyController", function($scope) {
    $scope.name = "Mr Local!";
  });
```

Un controller senza un parametro `$scope` potrebbe semplicemente non averne bisogno per qualche motivo. Ma è importante rendersi conto che, **anche con la sintassi del controllerAs** , esiste l'ambito locale.

Poiché `$scope` è un oggetto JavaScript, Angular lo imposta magicamente per ereditare

prototipicamente da `$rootScope` . E come puoi immaginare, ci può essere una catena di scopi. Ad esempio, è possibile creare un modello in un controllore principale e collegarlo all'ambito del controllore genitore come `$scope.model` .

Quindi tramite la catena di prototipi, un controller figlio poteva accedere localmente allo stesso modello con `$scope.model` .

Niente di tutto questo è inizialmente evidente, poiché è solo Angular che fa la sua magia in background. Ma capire `$scope` è un passo importante per conoscere come funziona Angular.

Il più semplice possibile Hello World angolare.

Angular 1 è il cuore di un compilatore DOM. Possiamo passarlo in HTML, sia come modello o semplicemente come una normale pagina web, e poi averlo compilare un'app.

Possiamo dire ad Angular di trattare una regione della pagina come *un'espressione* usando la sintassi in stile `{{ }}` manubri. Qualunque cosa tra le parentesi graffe verrà compilata, in questo modo:

```
{{ 'Hello' + 'World' }}
```

Questo produrrà:

```
HelloWorld
```

ng-app

Indichiamo ad Angular quale parte del nostro DOM viene trattata come modello principale utilizzando la *direttiva* `ng-app` . Una direttiva è un attributo o elemento personalizzato che il compilatore di template Angolare sa come gestire. Aggiungiamo ora una direttiva ng-app:

```
<html>
  <head>
    <script src="/angular.js"></script>
  </head>
  <body ng-app>
    {{ 'Hello' + 'World' }}
  </body>
</html>
```

Ora ho detto all'elemento del corpo di essere il modello radice. Tutto sarà compilato.

direttive

Le direttive sono direttive del compilatore. Estendono le funzionalità del compilatore Angular DOM. Ecco perché **Misko** , il creatore di Angular, descrive Angular come:

> "Che sarebbe stato un browser Web se fosse stato creato per le applicazioni web.

Creiamo letteralmente nuovi attributi ed elementi HTML e Angular li compila in un'app. `ng-app` è una direttiva che semplicemente attiva il compilatore. Altre direttive includono:

- `ng-click` , che aggiunge un gestore di clic,
- `ng-hide` , che nasconde in modo condizionale un elemento, e
- `<form>` , che aggiunge un comportamento aggiuntivo a un elemento del modulo HTML standard.

Angular viene fornito con circa 100 direttive incorporate che consentono di eseguire le attività più comuni. Possiamo anche scrivere il nostro, e questi saranno trattati allo stesso modo delle direttive incorporate.

Costruiamo un'app Angular con una serie di direttive, cablate insieme con HTML.

Minificazione in Angolare

Cos'è il Minification?

È il processo di rimozione di tutti i caratteri non necessari dal codice sorgente senza modificarne la funzionalità.

Sintassi normale

Se usiamo la normale sintassi angolare per scrivere un controller, dopo aver ridotto i nostri file interromperà la nostra funzionalità.

Controller (prima della minificazione):

```
var app = angular.module('mainApp', []);
app.controller('FirstController', function($scope) {
    $scope.name= 'Hello World !';
});
```

Dopo aver usato lo strumento di minificazione, verrà ridotto come di seguito.

```
var app=angular.module("mainApp",[]);app.controller("FirstController",function(e){e.name=
'Hello World !'})
```

Qui, la minimizzazione ha rimosso spazi non necessari e la variabile $ scope dal codice. Quindi, quando usiamo questo codice miniato, non stamperà nulla sulla vista. Perché $ scope è una parte cruciale tra controller e view, che ora è sostituita dalla piccola variabile 'e'. Pertanto, quando si esegue l'applicazione, viene generato un errore di dipendenza "e" del provider sconosciuto.

Esistono due modi per annotare il codice con le informazioni sul nome del servizio che sono minime:

Sintassi di annotazione in linea

```
var app = angular.module('mainApp', []);
app.controller('FirstController', ['$scope', function($scope) {
```

```
    $scope.message = 'Hello World !';
}]);
```

$ inject Sintassi di annotazione di proprietà

```
FirstController.$inject = ['$scope'];
var FirstController = function($scope) {
    $scope.message = 'Hello World !';
}

var app = angular.module('mainApp', []);
app.controller('FirstController', FirstController);
```

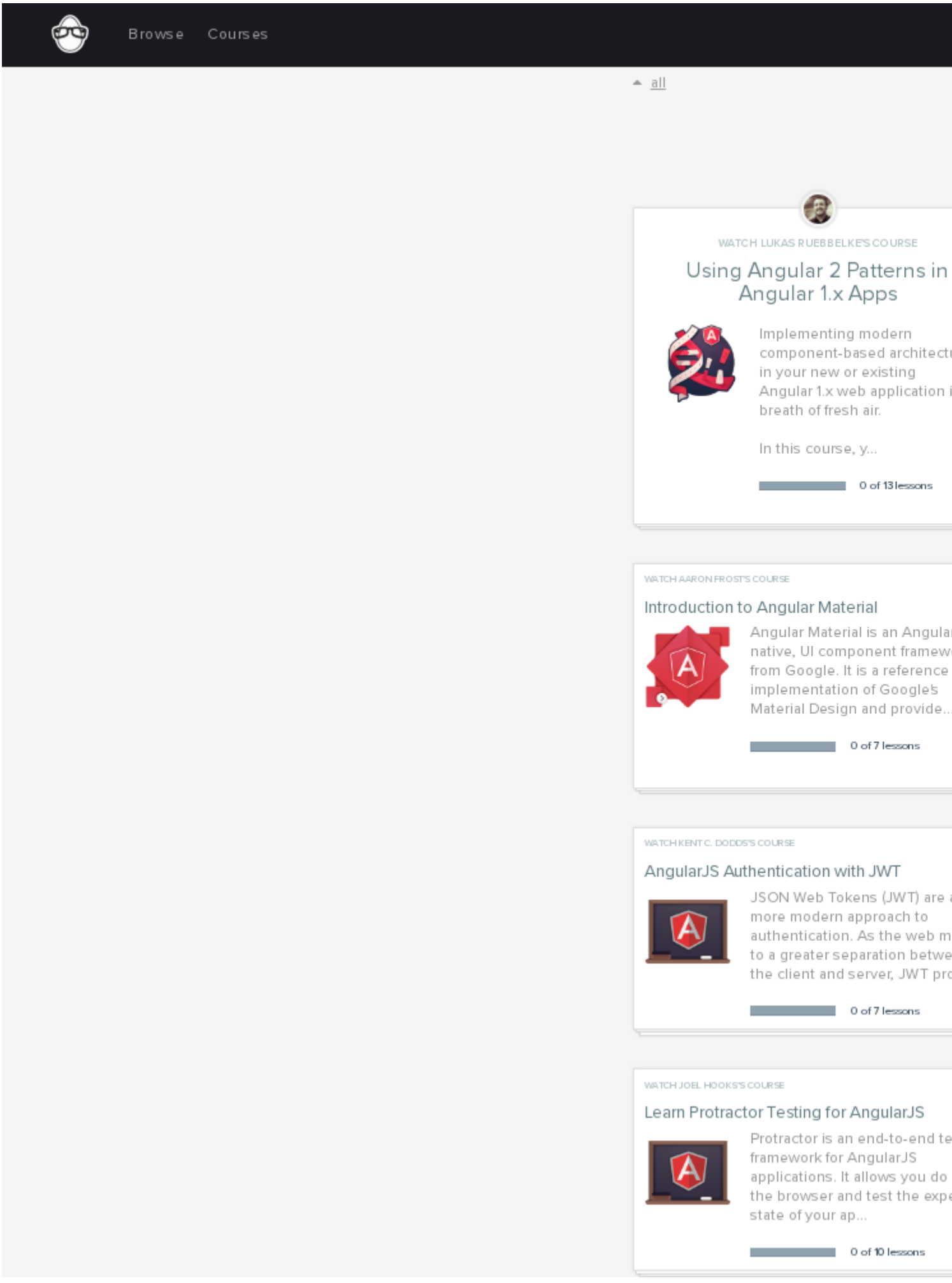
Dopo la minificazione, questo codice sarà

```
var
app=angular.module("mainApp",[]);app.controller("FirstController",["$scope",function(a){a.message="Hel
World !"}]);
```

Qui, angolare considererà la variabile 'a' da trattare come $ scope, e mostrerà l'output come 'Hello World!'.

AngularJS Guida introduttiva Tutorial video

Ci sono molti buoni video tutorial per il framework AngularJS su egghead.io



- https://egghead.io/courses/angularjs-app-from-scratch-getting-started

- https://egghead.io/courses/angularjs-application-architecture
- https://egghead.io/courses/angular-material-introduction
- https://egghead.io/courses/building-an-angular-1-x-ionic-application
- https://egghead.io/courses/angular-and-webpack-for-modular-applications
- https://egghead.io/courses/angularjs-authentication-with-jwt
- https://egghead.io/courses/angularjs-data-modeling
- https://egghead.io/courses/angular-automation-with-gulp
- https://egghead.io/courses/learn-protractor-testing-for-angularjs
- https://egghead.io/courses/ionic-quickstart-for-windows
- https://egghead.io/courses/build-angular-1-x-apps-with-redux
- https://egghead.io/courses/using-angular-2-patterns-in-angular-1-x-apps

Leggi Iniziare con AngularJS online: https://riptutorial.com/it/angularjs/topic/295/iniziare-con-angularjs

Capitolo 2: $ richiesta http

Examples

Usando $ http in un controller

Il servizio `$http` è una funzione che genera una richiesta HTTP e restituisce una promessa.

Uso generale

```
// Simple GET request example:
$http({
  method: 'GET',
  url: '/someUrl'
}).then(function successCallback(response) {
    // this callback will be called asynchronously
    // when the response is available
  }, function errorCallback(response) {
    // called asynchronously if an error occurs
    // or server returns response with an error status.
  });
```

Utilizzo all'interno del controller

```
appName.controller('controllerName',
    ['$http', function($http){

    // Simple GET request example:
    $http({
        method: 'GET',
        url: '/someUrl'
    }).then(function successCallback(response) {
        // this callback will be called asynchronously
        // when the response is available
    }, function errorCallback(response) {
        // called asynchronously if an error occurs
        // or server returns response with an error status.
    });
}])
```

Metodi di scelta rapida

`$http` servizio `$http` ha anche metodi di scelta rapida. Leggi i metodi http qui

Sintassi

```
$http.get('/someUrl', config).then(successCallback, errorCallback);
$http.post('/someUrl', data, config).then(successCallback, errorCallback);
```

Metodi di scelta rapida

- $ http.get

- $ http.head
- $ http.post
- $ http.put
- $ http.delete
- $ http.jsonp
- $ http.patch

Utilizzo della richiesta $ http in un servizio

Le richieste HTTP sono ampiamente utilizzate ripetutamente in ogni app Web, quindi è consigliabile scrivere un metodo per ogni richiesta comune e quindi utilizzarlo in più punti all'interno dell'app.

Creare un `httpRequestsService.js`

httpRequestsService.js

```
appName.service('httpRequestsService', function($q, $http){

    return {
        // function that performs a basic get request
        getName: function(){
            // make sure $http is injected
            return $http.get("/someAPI/names")
                .then(function(response) {
                    // return the result as a promise
                    return response;
                }, function(response) {
                    // defer the promise
                    return $q.reject(response.data);
                });
        },

        // add functions for other requests made by your app
        addName: function(){
            // some code...
        }
    }
})
```

Il servizio di cui sopra eseguirà una richiesta di ottenere all'interno del servizio. Questo sarà disponibile per qualsiasi controller in cui il servizio è stato iniettato.

Esempio di utilizzo

```
appName.controller('controllerName',
    ['httpRequestsService', function(httpRequestsService){

        // we injected httpRequestsService service on this controller
        // that made the getName() function available to use.
        httpRequestsService.getName()
            .then(function(response){
                // success
            }, function(error){
```

```
                // do something with the error
            })
    }])
```

Utilizzando questo approccio possiamo ora utilizzare **httpRequestsService.js in** qualsiasi momento e in qualsiasi controller.

Tempistica di una richiesta $ http

Le richieste $ http richiedono tempi variabili a seconda del server, alcuni possono richiedere alcuni millisecondi e alcuni possono richiedere alcuni secondi. Spesso il tempo richiesto per recuperare i dati da una richiesta è fondamentale. Supponendo che il valore di risposta sia un array di nomi, considera il seguente esempio:

scorretto

```
$scope.names = [];

$http({
    method: 'GET',
    url: '/someURL'
}).then(function successCallback(response) {
            $scope.names = response.data;
        },
        function errorCallback(response) {
            alert(response.status);
        });

alert("The first name is: " + $scope.names[0]);
```

L'accesso a `$scope.names[0]` proprio sotto la richiesta $ http genera spesso un errore - questa riga di codice viene eseguita prima che la risposta sia ricevuta dal server.

Corretta

```
$scope.names = [];

$scope.$watch('names', function(newVal, oldVal) {
    if(!(newVal.length == 0)) {
        alert("The first name is: " + $scope.names[0]);
    }
});

$http({
    method: 'GET',
    url: '/someURL'
}).then(function successCallback(response) {
            $scope.names = response.data;
        },
        function errorCallback(response) {
            alert(response.status);
        });
```

Usando il servizio $ watch `$scope.names` all'array `$scope.names` solo quando viene ricevuta la

risposta. Durante l'inizializzazione, la funzione viene chiamata anche se `$scope.names` stato inizializzato in precedenza, pertanto è necessario verificare se `newVal.length` è diverso da 0. Attenzione: qualsiasi modifica apportata a `$scope.names` attiverà la funzione di visualizzazione.

Leggi $ richiesta http online: https://riptutorial.com/it/angularjs/topic/3620/--richiesta-http

Capitolo 3: Ambiti $ angolari

Osservazioni

Angular utilizza un **albero** di ambiti per collegare la logica (da controller, direttive, ecc.) Alla vista e costituisce il meccanismo principale alla base del rilevamento dei cambiamenti in AngularJS. Un riferimento più dettagliato per gli ambiti può essere trovato su docs.angularjs.org

La radice dell'albero è accessibile tramite il servizio **inject-$ $ rootScope** . Tutti gli ambiti child $ ereditano i metodi e le proprietà del proprio scope $ scope, consentendo ai bambini l'accesso ai metodi senza l'uso di Angular Services.

Examples

Esempio di base dell'eredità $ scope

```
angular.module('app', [])
.controller('myController', ['$scope', function($scope){
    $scope.person = { name: 'John Doe' };
}]);

<div ng-app="app" ng-conroller="myController">
    <input ng-model="person.name" />
    <div ng-repeat="number in [0,1,2,3]">
        {{person.name}} {{number}}
    </div>
</div>
```

In questo esempio, la direttiva ng-repeat crea un nuovo ambito per ciascuno dei suoi figli appena creati.

Questi ambiti creati sono figli del loro ambito genitore (in questo caso l'ambito creato da myController) e, pertanto, ereditano tutte le sue proporzioni, come la persona.

Evita di ereditare i valori primitivi

In javascript, assegnando un valore non primitivo (come Oggetto, Matrice, Funzione e molti altri), mantiene un riferimento (un indirizzo in memoria) al valore assegnato.

Assegnare un valore primitivo (String, Number, Boolean o Symbol) a due variabili diverse e cambiarne uno, non cambierà entrambi:

```
var x = 5;
var y = x;
y = 6;
console.log(y === x, x, y); //false, 5, 6
```

Ma con un valore non primitivo, poiché entrambe le variabili mantengono semplicemente i

riferimenti allo stesso oggetto, cambiando una variabile **si** cambia l'altra:

```
var x = { name : 'John Doe' };
var y = x;
y.name = 'Jhon';
console.log(x.name === y.name, x.name, y.name); //true, John, John
```

In angolare, quando viene creato un ambito, vengono assegnate tutte le proprietà del suo genitore. Tuttavia, cambiando le proprietà in seguito, si influirà solo sull'ambito genitore se si tratta di un valore non primitivo:

```
angular.module('app', [])
.controller('myController', ['$scope', function($scope){
    $scope.person = { name: 'John Doe' }; //non-primitive
    $scope.name = 'Jhon Doe'; //primitive
}])
.controller('myController1', ['$scope', function($scope){}]);

<div ng-app="app" ng-controller="myController">
    binding to input works: {{person.name}}<br/>
    binding to input does not work: {{name}}<br/>
    <div ng-controller="myController1">
        <input ng-model="person.name" />
        <input ng-model="name" />
    </div>
</div>
```

Ricorda: in Gli ambiti angolari possono essere creati in molti modi (come le direttive incorporate o personalizzate, o la funzione `$scope.$new()`) e tenere traccia della struttura ad albero è probabilmente impossibile.

Usando solo valori non primitivi come proprietà dell'ambito ti manterrai al sicuro (a meno che tu non abbia bisogno di una proprietà da non ereditare, o di altri casi in cui sei a conoscenza dell'ereditarietà dell'ambito).

Una funzione disponibile nell'intera app

Attenzione, questo approccio potrebbe essere considerato come una cattiva progettazione per le app angolari, dal momento che richiede ai programmatori di ricordare sia dove le funzioni sono posizionate nella struttura ad albero, sia a conoscenza dell'ereditarietà dell'ambito. In molti casi sarebbe preferibile iniettare un servizio (pratica angolare - utilizzando l'ereditarietà dell'ambito rispetto all'iniezione) .

Questo esempio mostra solo come l'ereditarietà dell'ambito può essere utilizzata per le nostre esigenze e il modo in cui puoi trarne vantaggio, e non le migliori pratiche di progettazione di un'intera app.

In alcuni casi, potremmo sfruttare l'ereditarietà dell'ambito e impostare una funzione come proprietà di rootScope. In questo modo, tutti gli ambiti nell'app (tranne gli ambiti isolati) erediteranno questa funzione e potranno essere richiamati da qualsiasi punto dell'app.

```
angular.module('app', [])
.run(['$rootScope', function($rootScope){
    var messages = []
    $rootScope.addMessage = function(msg){
        messages.push(msg);
    }
}]);

<div ng-app="app">
    <a ng-click="addMessage('hello world!')">it could be accsessed from here</a>
    <div ng-include="inner.html"></div>
</div>
```

inner.html:

```
<div>
    <button ng-click="addMessage('page')">and from here to!</button>
</div>
```

Creazione di eventi $ scope personalizzati

Come i normali elementi HTML, è possibile che gli ambiti $ abbiano i propri eventi. Gli eventi $ scope possono essere sottoscritti utilizzando la seguente modalità:

```
 $scope.$on('my-event', function(event, args) {
    console.log(args); // { custom: 'data' }
});
```

Se è necessario annullare la registrazione di un listener di eventi, la funzione **$ on** restituirà una funzione non vincolante. Per continuare con l'esempio precedente:

```
var unregisterMyEvent = $scope.$on('my-event', function(event, args) {
    console.log(args); // { custom: 'data' }
    unregisterMyEvent();
});
```

Esistono due modi per attivare il proprio evento **$** scope **$ broadcast** e **$ emit personalizzati** . Per notificare il / i genitore / i di un ambito di un evento specifico, usa **$ emit**

```
$scope.$emit('my-event', { custom: 'data' });
```

L'esempio precedente attiverà qualsiasi listener di eventi per il `my-event` nello scope parent e continuerà la struttura dell'oscilloscopio su **$ rootScope a** meno che un listener non chiami `stopPropagation` sull'evento. Solo gli eventi attivati con **$ emit** possono chiamare `stopPropagation`

Il retro di **$ emit** è **$ broadcast** , che attiverà qualsiasi listener di eventi su tutti gli ambiti figlio nella struttura ad albero che sono figli dello scope che ha chiamato **$ broadcast** .

```
$scope.$broadcast('my-event', { custom: 'data' });
```

Gli eventi attivati con **$ broadcast** non possono essere annullati.

Usando le funzioni $ scope

Mentre dichiarare una funzione in $ rootcope ha i suoi vantaggi, possiamo anche dichiarare una funzione $ scope qualsiasi parte del codice che viene iniettata dal servizio $ scope. Controller, per esempio.

controllore

```
myApp.controller('myController', ['$scope', function($scope){
    $scope.myFunction = function () {
        alert("You are in myFunction!");
    };
}]);
```

Ora puoi chiamare la tua funzione dal controller usando:

```
$scope.myfunction();
```

O tramite HTML che si trova sotto quel controller specifico:

```
<div ng-controller="myController">
    <button ng-click="myFunction()"> Click me! </button>
</div>
```

Direttiva

Una direttiva angolare è un altro punto in cui puoi utilizzare il tuo ambito:

```
myApp.directive('triggerFunction', function() {
    return {
        scope: {
            triggerFunction: '&'
        },
        link: function(scope, element) {
            element.bind('mouseover', function() {
                scope.triggerFunction();
            });
        }
    };
});
```

E nel tuo codice HTML sotto lo stesso controller:

```
<div ng-controller="myController">
    <button trigger-function="myFunction()"> Hover over me! </button>
</div>
```

Naturalmente, è possibile utilizzare ngMouseover per la stessa cosa, ma ciò che è speciale delle direttive è che è possibile personalizzarle nel modo desiderato. E ora sai come usare le tue funzioni $ scope al loro interno, sii creativo!

Come puoi limitare l'ambito di applicazione a una direttiva e perché dovresti farlo?

Scope è usato come "colla" che usiamo per comunicare tra il controllore genitore, la direttiva e il modello di direttiva. Ogni volta che l'applicazione AngularJS viene sottoposta a boot, viene creato un oggetto rootScope. Ogni ambito creato da controllori, direttive e servizi è prototipicamente ereditato da rootScope.

Sì, possiamo limitare l'ambito su una direttiva. Possiamo farlo creando un ambito isolato per la direttiva.

> Esistono 3 tipi di ambiti direttive:
>
> 1. Ambito: False (la direttiva usa il suo ambito genitore)
> 2. Ambito: True (la direttiva ottiene un nuovo ambito)
> 3. Ambito: {} (la direttiva ottiene un nuovo ambito isolato)

Direttive con il nuovo ambito isolato: quando creiamo un nuovo ambito isolato, esso non verrà ereditato dall'ambito principale. Questo nuovo ambito si chiama ambito isolato perché è completamente separato dall'ambito principale. Perché? dovremmo usare un ambito isolato: dovremmo usare un ambito isolato quando vogliamo creare una direttiva personalizzata perché farà in modo che la nostra direttiva sia generica e posizionata ovunque all'interno dell'applicazione. L'ambito genitore non interferirà con l'ambito della direttiva.

Esempio di ambito isolato:

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

app.controller("Ctrl1",function($scope){
    $scope.name = "Prateek";
    $scope.reverseName = function(){
        $scope.name = $scope.name.split('').reverse().join('');
    };
});
app.directive("myDirective", function(){
    return {
        restrict: "EA",
        scope: {},
        template: "<div>Your name is : {{name}}</div>"+
        "Change your name : <input type='text' ng-model='name'/>"
    };
});
```

Esistono 3 tipi di prefissi che AngularJS fornisce per l'ambito isolato:

1. "@" (Rilegatura del testo / rilegatura unidirezionale)
2. "=" (Rilegatura diretta del modello / rilegatura bidirezionale)
3. "&" (Binding del comportamento / binding del metodo)

Tutti questi prefissi ricevono i dati dagli attributi dell'elemento direttiva come:

```
<div my-directive
  class="directive"
  name="{{name}}"
  reverse="reverseName()"
  color="color" >
</div>
```

Leggi Ambiti $ angolari online: https://riptutorial.com/it/angularjs/topic/3157/ambiti---angolari

Capitolo 4: angularjs con filtro dati, impaginazione ecc

introduzione

Esempio di provider e query sui dati di visualizzazione con filtro, impaginazione, ecc in Angularjs.

Examples

Angularjs visualizza i dati con filtro, paginazione

```
<div ng-app="MainApp" ng-controller="SampleController">
    <input ng-model="dishName" id="search" class="form-control" placeholder="Filter text">
    <ul>
        <li dir-paginate="dish in dishes |  filter : dishName | itemsPerPage: pageSize"
pagination-id="flights">{{dish}}</li>
    </ul>
    <dir-pagination-controls boundary-links="true" on-page-
change="changeHandler(newPageNumber)" pagination-id="flights"></dir-pagination-controls>
</div>
<script type="text/javascript" src="angular.min.js"></script>
<script type="text/javascript" src="pagination.js"></script>
<script type="text/javascript">

var MainApp = angular.module('MainApp', ['angularUtils.directives.dirPagination'])
MainApp.controller('SampleController', ['$scope', '$filter', function ($scope, $filter) {

    $scope.pageSize = 5;

    $scope.dishes = [
    'noodles',
    'sausage',
    'beans on toast',
    'cheeseburger',
    'battered mars bar',
    'crisp butty',
    'yorkshire pudding',
    'wiener schnitzel',
    'sauerkraut mit ei',
    'salad',
    'onion soup',
    'bak choi',
    'avacado maki'
    ];

    $scope.changeHandler = function (newPage) { };
}]);
</script>
```

Leggi angularjs con filtro dati, impaginazione ecc online:
https://riptutorial.com/it/angularjs/topic/10821/angularjs-con-filtro-dati--impaginazione-ecc

Capitolo 5: Archiviazione delle sessioni

Examples

Gestione dello storage di sessione tramite il servizio utilizzando angularjs

Servizio di archiviazione di sessione:

Servizio di fabbrica comune che salverà e restituirà i dati della sessione salvati in base alla chiave.

```
 'use strict';

/**
 * @ngdoc factory
 * @name app.factory:storageService
 * @description This function will communicate with HTML5 sessionStorage via Factory Service.
 */

app.factory('storageService', ['$rootScope', function($rootScope) {

    return {
        get: function(key) {
            return sessionStorage.getItem(key);
        },
        save: function(key, data) {
            sessionStorage.setItem(key, data);
        }
    };
}]);
```

Nel controller:

Inietta la dipendenza storageService nel controller per impostare e ottenere i dati dalla memoria di sessione.

```
app.controller('myCtrl',['storageService',function(storageService) {

  // Save session data to storageService
  storageService.save('key', 'value');

  // Get saved session data from storageService
  var sessionData = storageService.get('key');

});
```

Leggi Archiviazione delle sessioni online:
https://riptutorial.com/it/angularjs/topic/8201/archiviazione-delle-sessioni

Capitolo 6: Caricamento lento

Osservazioni

1. Se le tue dipendenze caricate pigre richiedono altre dipendenze caricate pigre assicurati di caricarle nell'ordine corretto!

```
angular.module('lazy', [
 'alreadyLoadedDependency1',
 'alreadyLoadedDependency2',
 ...
 {
  files: [
  'path/to/lazily/loaded/dependency1.js',
  'path/to/lazily/loaded/dependency2.js', //<--- requires lazily loaded dependency1
  'path/to/lazily/loaded/dependency.css'
  ],
  serie: true //Sequential load instead of parallel
 }
]);
```

Examples

Preparare il tuo progetto per il caricamento lento

Dopo aver incluso `oclazyload.js` nel tuo file di indice, dichiara `ocLazyLoad` come dipendenza in `app.js`

```
//Make sure you put the correct dependency! it is spelled different than the service!
angular.module('app', [
 'oc.lazyLoad',
 'ui-router'
])
```

uso

Per caricare pigramente i file, iniettare il servizio `$ocLazyLoad` in un controller o in un altro servizio

```
.controller('someCtrl', function($ocLazyLoad) {
 $ocLazyLoad.load('path/to/file.js').then(...);
});
```

I moduli angolari verranno automaticamente caricati in angolari.

Altre varianti:

```
$ocLazyLoad.load([
  'bower_components/bootstrap/dist/js/bootstrap.js',
  'bower_components/bootstrap/dist/css/bootstrap.css',
```

```
  'partials/template1.html'
]);
```

Per un elenco completo delle varianti visita la documentazione ufficiale

Utilizzo con router

UI-Router:

```
.state('profile', {
 url: '/profile',
 controller: 'profileCtrl as vm'
 resolve: {
  module: function($ocLazyLoad) {
   return $ocLazyLoad.load([
    'path/to/profile/module.js',
    'path/to/profile/style.css'
   ]);
  }
 }
});
```

ngRoute:

```
.when('/profile', {
     controller: 'profileCtrl as vm'
     resolve: {
      module: function($ocLazyLoad) {
       return $ocLazyLoad.load([
        'path/to/profile/module.js',
        'path/to/profile/style.css'
       ]);
      }
     }
    });
```

Utilizzando l'iniezione di dipendenza

La seguente sintassi consente di specificare le dipendenze nel `module.js` anziché nelle specifiche esplicite quando si utilizza il servizio

```
//lazy_module.js
angular.module('lazy', [
 'alreadyLoadedDependency1',
 'alreadyLoadedDependency2',
 ...
 [
  'path/to/lazily/loaded/dependency.js',
  'path/to/lazily/loaded/dependency.css'
 ]
]);
```

Nota : questa sintassi funzionerà solo con moduli caricati pigramente!

Usando la direttiva

```
<div oc-lazy-load="['path/to/lazy/loaded/directive.js',
'path/to/lazy/loaded/directive.html']">

<!-- myDirective available here -->
<my-directive></my-directive>

</div>
```

Leggi Caricamento lento online: https://riptutorial.com/it/angularjs/topic/6400/caricamento-lento

Capitolo 7: Come funziona il collegamento dei dati

Osservazioni

Quindi, mentre questo concetto di Data Binding nel suo complesso è facile per lo sviluppatore, è piuttosto pesante sul browser poiché Angular ascolta ogni cambiamento di evento ed esegue il ciclo di digestione. Per questo motivo, ogni volta che associamo un modello alla vista, assicurati che Scope sia il più ottimizzato possibile

Examples

Esempio di associazione dati

```
<p ng-bind="message"></p>
```

Questo 'messaggio' deve essere collegato all'ambito del controller degli elementi corrente.

```
$scope.message = "Hello World";
```

In un secondo momento, anche se il modello del messaggio viene aggiornato, il valore aggiornato si riflette nell'elemento HTML. Quando compila angolare il modello "Hello World" sarà collegato al innerHTML del mondo attuale. Angolare mantiene un meccanismo di osservazione di tutte le direttive attaccate alla vista. Ha un meccanismo del ciclo del digest in cui scorre l'array di Watchers, aggiornando l'elemento DOM se c'è un cambiamento nel valore precedente del modello.

Non è previsto un controllo periodico dell'ambito di applicazione in caso di modifica degli oggetti ad esso associati. Non tutti gli oggetti collegati all'ambito vengono guardati. L'ambito mantiene prototipicamente un **WatchersArray $$** . L'ambito esegue solo iterazioni attraverso questo WatchersArray quando viene chiamato $ digest.

Angular aggiunge un osservatore a WatchersArray per ognuno di questi

1. {{espressione}} - Nei tuoi modelli (e in qualsiasi altro luogo in cui è presente un'espressione) o quando definiamo ng-model.
2. $ scope. $ watch ('expression / function') - Nel tuo JavaScript possiamo solo collegare un oggetto scope per il controllo angolare.

La funzione **$ watch** accetta tre parametri:

1. Il primo è una funzione di watcher che restituisce appena l'oggetto o possiamo semplicemente aggiungere un'espressione.

2. Il secondo è una funzione di ascolto che verrà chiamata quando c'è un cambiamento nell'oggetto. Tutte le cose come le modifiche DOM verranno implementate in questa funzione.

3. Il terzo è un parametro opzionale che accetta un valore booleano. Se il suo vero, profondo angolare guarda l'oggetto e se il suo falso Angolare fa semplicemente un riferimento guardando sull'oggetto. L'implementazione approssimativa di $ watch appare così

```
Scope.prototype.$watch = function(watchFn, listenerFn) {
   var watcher = {
       watchFn: watchFn,
       listenerFn: listenerFn || function() { },
       last: initWatchVal  // initWatchVal is typically undefined
   };
   this.$$watchers.push(watcher); // pushing the Watcher Object to Watchers
};
```

C'è una cosa interessante in Angular chiamato Digest Cycle. Il ciclo $ digest inizia come risultato di una chiamata a $ scope. $ Digest (). Si supponga di modificare un modello $ scope in una funzione di gestione tramite la direttiva ng-click. In questo caso, AngularJS attiva automaticamente un ciclo $ digest chiamando $ digest (). Oltre a ng-click, ci sono molte altre direttive / servizi incorporati che consentono di modificare i modelli (ad esempio ng-model, $ timeout, ecc.) e attiva automaticamente un ciclo $ digest. L'implementazione approssimativa di $ digest appare così.

```
Scope.prototype.$digest = function() {
      var dirty;
      do {
          dirty = this.$$digestOnce();
      } while (dirty);
}
Scope.prototype.$$digestOnce = function() {
   var self = this;
   var newValue, oldValue, dirty;
   _.forEach(this.$$watchers, function(watcher) {
          newValue = watcher.watchFn(self);
          oldValue = watcher.last;   // It just remembers the last value for dirty checking
          if (newValue !== oldValue) { //Dirty checking of References
   // For Deep checking the object , code of Value
   // based checking of Object should be implemented here
              watcher.last = newValue;
              watcher.listenerFn(newValue,
                   (oldValue === initWatchVal ? newValue : oldValue),
                    self);
          dirty = true;
          }
     });
   return dirty;
 };
```

Se usiamo la funzione **setTimeout ()** di JavaScript per aggiornare un modello di scope, Angular non ha modo di sapere cosa potresti cambiare. In questo caso è nostra responsabilità chiamare $ apply () manualmente, che attiva un ciclo $ digest. Allo stesso modo, se si dispone di una direttiva che imposta un listener di eventi DOM e modifica alcuni modelli all'interno della funzione di

gestione, è necessario chiamare $ apply () per garantire che le modifiche abbiano effetto. La grande idea di $ apply è che possiamo eseguire un codice che non è a conoscenza di Angular, che il codice potrebbe ancora cambiare le cose sull'ambito. Se avvolgiamo quel codice in $ apply, si prenderà cura di chiamare $ digest (). Implementazione approssimativa di $ apply ().

```
Scope.prototype.$apply = function(expr) {
       try {
         return this.$eval(expr); //Evaluating code in the context of Scope
       } finally {
         this.$digest();
       }
};
```

Leggi Come funziona il collegamento dei dati online:
https://riptutorial.com/it/angularjs/topic/2342/come-funziona-il-collegamento-dei-dati

Capitolo 8: Compiti grunt

Examples

Esegui l'applicazione localmente

> L'esempio seguente richiede che node.js sia installato e che npm sia disponibile.
> Il codice di lavoro completo può essere biforcato da GitHub @
> https://github.com/mikkoviitala/angular-grunt-run-local

Di solito una delle prime cose che vuoi fare quando si sviluppa una nuova applicazione web è farla girare localmente.

Qui di seguito troverete un esempio completo che raggiunge proprio questo, usando grunt (javascript task runner), npm (node package manager) e bower (ancora un altro gestore di pacchetti).

Oltre ai tuoi file di applicazione effettivi dovrai installare alcune dipendenze di terze parti utilizzando gli strumenti sopra menzionati. Nella directory del progetto, **preferibilmente root** , avrai bisogno di tre (3) file.

- package.json (dipendenze gestite da npm)
- bower.json (dipendenze gestite da bower)
- gruntfile.js (compiti grunt)

Quindi la directory del tuo progetto è così:

---

package.json

Installeremo il grunt stesso, **matchdep** per semplificarci la vita permettendoci di filtrare le dipendenze per nome, **grunt-express** usato per avviare express web server via grunt e **grunt-open** per aprire url / file da un grunt task.

Quindi questi pacchetti sono tutti su "infrastruttura" e aiutanti su cui creeremo la nostra applicazione.

```
{
  "name": "app",
  "version": "1.0.0",
  "dependencies": {},
  "devDependencies": {
```

```
    "grunt": "~0.4.1",
    "matchdep": "~0.1.2",
    "grunt-express": "~1.0.0-beta2",
    "grunt-open": "~0.2.1"
  },
  "scripts": {
    "postinstall": "bower install"
  }
}
```

bower.json

Bower è (o almeno dovrebbe essere) tutto sul front-end e lo useremo per installare **angolare** .

```
{
  "name": "app",
  "version": "1.0.0",
  "dependencies": {
    "angular": "~1.3.x"
  },
  "devDependencies": {}
}
```

gruntfile.js

All'interno di gruntfile.js avremo l'effettiva "applicazione in esecuzione" magica, che apre la nostra applicazione in una nuova finestra del browser, in esecuzione su http: // localhost: 9000 /

```
'use strict';

// see http://rhumaric.com/2013/07/renewing-the-grunt-livereload-magic/

module.exports = function(grunt) {
  require('matchdep').filterDev('grunt-*').forEach(grunt.loadNpmTasks);

  grunt.initConfig({
    express: {
      all: {
        options: {
          port: 9000,
          hostname: 'localhost',
          bases: [__dirname]
        }
      }
    },

    open: {
      all: {
        path: 'http://localhost:<%= express.all.options.port%>'
      }
    }
  });

  grunt.registerTask('app', [
    'express',
    'open',
    'express-keepalive'
  ]);
```

```
};
```

uso

Per far funzionare la tua applicazione da zero, salva i file sopra nella directory principale del tuo progetto (qualsiasi cartella vuota lo farà). Quindi avviare la console / riga di comando e digitare quanto segue per installare tutte le dipendenze richieste.

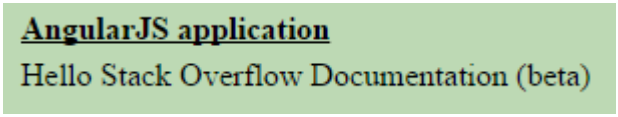
```
npm install -g grunt-cli bower
npm install
```

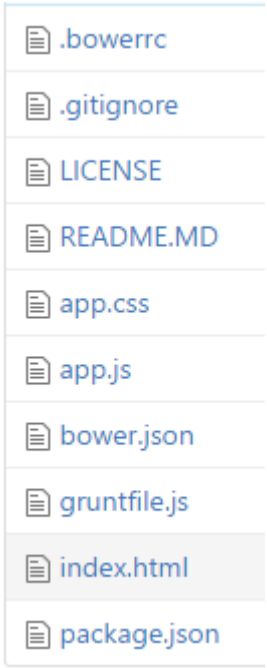
E poi esegui la tua applicazione usando

```
grunt app
```

Nota che sì, avrai bisogno anche dei tuoi file di applicazione reali.
Per un esempio quasi minimale, consulta il repository GitHub menzionato all'inizio di questo esempio.

La struttura non è così diversa. C'è solo il template `index.html` , il codice angolare in `app.js` e alcuni stili in `app.css` . Altri file sono per la configurazione Git e editor e alcune cose generiche. Provaci!





Leggi Compiti grunt online: https://riptutorial.com/it/angularjs/topic/6077/compiti-grunt

Capitolo 9: componenti

Parametri

Parametro	Dettagli
=	Per utilizzare l'associazione dati bidirezionale. Ciò significa che se si aggiorna quella variabile nell'ambito del componente, la modifica si rifletterà sull'ambito principale.
<	Associazioni unidirezionali quando vogliamo solo leggere un valore da un ambito genitore e non aggiornarlo.
@	Parametri di stringa.
&	Per i callback nel caso in cui il tuo componente abbia bisogno di generare qualcosa per il suo genitore.
-	-
LifeCycle Hooks	**Dettagli** (richiede angular.version> = 1.5.3)
$ onInit ()	Chiamato su ciascun controller dopo che tutti i controller su un elemento sono stati costruiti e hanno inizializzato i loro binding. Questo è un buon posto dove mettere il codice di inizializzazione per il tuo controller.
$ onChanges (changesObj)	Chiamato quando vengono aggiornati i binding unidirezionali. Le `changesObj` è un hash le cui chiavi sono i nomi delle proprietà associate che sono state modificate ei valori sono un oggetto della forma `{ currentValue, previousValue, isFirstChange() }` .
$ OnDestroy ()	Chiamato su un controller quando il suo ambito di contenimento viene distrutto. Utilizza questo hook per rilasciare risorse esterne, orologi e gestori di eventi.
$ postLink ()	Chiamato dopo l'elemento di questo controller e i suoi figli sono stati collegati. Questo hook può essere considerato analogo ai ganci ngAfterViewInit e ngAfterContentInit in Angular 2.
$ doCheck ()	Chiamato ad ogni turno del ciclo di digestione. Offre l'opportunità di rilevare e agire sui cambiamenti. Qualsiasi azione che si desidera eseguire in risposta alle modifiche rilevate deve essere invocata da questo hook; l'implementazione di questo non ha alcun effetto su quando viene chiamato $ onChanges.

Osservazioni

Component è un tipo speciale di direttiva che utilizza una configurazione più semplice che è adatta per una struttura applicativa basata su componenti. I componenti sono stati introdotti in Angular 1.5, gli esempi in questa sezione **non funzioneranno** con le versioni precedenti di AngularJS.

Una guida completa per sviluppatori su Componenti è disponibile su https://docs.angularjs.org/guide/component

Examples

Componenti di base e ganci LifeCycle

Cos'è un componente?

- Un componente è fondamentalmente una direttiva che utilizza una configurazione più semplice e che è adatta per un'architettura basata su componenti, che è ciò che riguarda Angular 2. Pensa a un componente come a un widget: un pezzo di codice HTML che puoi riutilizzare in diversi punti della tua applicazione web.

Componente

```
angular.module('myApp', [])
    .component('helloWorld', {
        template: '<span>Hello World!</span>'
    });
```

markup

```
<div ng-app="myApp">
  <hello-world> </hello-world>
</div>
```

Dimostrazione dal vivo

Utilizzo di dati esterni in Component:

Potremmo aggiungere un parametro per passare un nome al nostro componente, che verrebbe utilizzato come segue:

```
angular.module("myApp", [])
  .component("helloWorld",{
    template: '<span>Hello {{$ctrl.name}}!</span>',
    bindings: { name: '@' }
```

```
  });
```

markup

```
<div ng-app="myApp">
  <hello-world name="'John'" > </hello-world>
</div>
```

Dimostrazione dal vivo

Utilizzo dei controller nei componenti

Diamo un'occhiata a come aggiungere un controller ad esso.

```
angular.module("myApp", [])
  .component("helloWorld",{
      template: "Hello {{$ctrl.name}}, I'm {{$ctrl.myName}}!",
      bindings: { name: '@' },
      controller: function(){
        this.myName = 'Alain';
      }
  });
```

markup

```
<div ng-app="myApp">
  <hello-world name="John"> </hello-world>
</div>
```

Demo CodePen

I parametri passati al componente sono disponibili nell'ambito del controller appena prima che la funzione `$onInit` venga chiamata da Angular. Considera questo esempio:

```
angular.module("myApp", [])
  .component("helloWorld",{
      template: "Hello {{$ctrl.name}}, I'm {{$ctrl.myName}}!",
      bindings: { name: '@' },
      controller: function(){
        this.$onInit = function() {
          this.myName = "Mac" + this.name;
        }
      }
  });
```

Nel modello dall'alto, ciò renderebbe "Ciao John, sono MacJohn!".

Si noti che `$ctrl` è il valore predefinito Angolare per `controllerAs` se non ne viene specificato uno.

Dimostrazione dal vivo

Usare "require" come oggetto

In alcuni casi potrebbe essere necessario accedere ai dati da un componente principale all'interno del componente.

Questo può essere ottenuto specificando che il nostro componente richiede quel componente genitore, il fabbisogno ci fornirà il riferimento al controller componente richiesto, che può quindi essere utilizzato nel nostro controller come mostrato nell'esempio seguente:

> Si noti che i controller richiesti sono garantiti per essere pronti solo dopo il hook $ onInit.

```
angular.module("myApp", [])
  .component("helloWorld",{
      template: "Hello {{$ctrl.name}}, I'm {{$ctrl.myName}}!",
      bindings: { name: '@' },
      require: {
        parent: '^parentComponent'
      },
      controller: function () {
        // here this.parent might not be initiated yet

        this.$onInit = function() {
           // after $onInit, use this.parent to access required controller
           this.parent.foo();
        }

      }
  });
```

Tenete presente, tuttavia, che ciò crea un accoppiamento stretto tra il bambino e il genitore.

Componenti in JS angolare

I componenti in angularJS possono essere visualizzati come una direttiva personalizzata (<html> questo in una direttiva HTML, e qualcosa di simile sarà una direttiva personalizzata <ANYTHING>). Un componente contiene una vista e un controller. Il controller contiene la logica di business che è associata a una vista che l'utente vede. Il componente differisce da una direttiva angolare perché contiene meno configurazione. Una componente angolare può essere definita in questo modo.

```
angular.module("myApp",[]).component("customer", {})
```

I componenti sono definiti sui moduli angolari. Contengono due argomenti, Uno è il nome del componente e il secondo è un oggetto che contiene una coppia di valori chiave, che definisce quale vista e quale controller verrà utilizzato in questo modo.

```
angular.module("myApp",[]).component("customer", {
    templateUrl : "customer.html", // your view here
    controller: customerController, //your controller here
```

```
    controllerAs: "cust"        //alternate name for your controller
})
```

"myApp" è il nome dell'app che stiamo costruendo e il cliente è il nome del nostro componente. Ora per chiamarlo nel file html principale lo metteremo semplicemente così

```
<customer></customer>
```

Ora questa direttiva sarà sostituita dalla vista che hai specificato e dalla logica di business che hai scritto nel tuo controller.

NOTA: Ricorda componente prendere un oggetto come secondo argomento mentre direttiva prendere una funzione di fabbrica come argomento.

Leggi componenti online: https://riptutorial.com/it/angularjs/topic/892/componenti

Capitolo 10: Condivisione dei dati

Osservazioni

Una domanda molto comune quando si lavora con Angular è come condividere i dati tra i controller. L'utilizzo di un servizio è la risposta più frequente e questo è un semplice esempio che dimostra un modello factory per condividere qualsiasi tipo di oggetto dati tra due o più controller. Poiché si tratta di un riferimento a oggetti condivisi, un aggiornamento in un controller sarà immediatamente disponibile in tutti gli altri controller che utilizzano il servizio. Si noti che sia il servizio che la fabbrica e entrambi i provider .

Examples

Utilizzo di ngStorage per condividere i dati

In primo luogo, includi l'origine **ngStorage** nel tuo index.html.

Un esempio di iniezione di `ngStorage` src potrebbe essere:

```
<head>
    <title>Angular JS ngStorage</title>
    <script src =
"http://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>
    <script src="https://rawgithub.com/gsklee/ngStorage/master/ngStorage.js"></script>
</head>
```

`ngStorage` ti dà 2 storage vale a dire: `$localStorage` e `$sessionStorage` . È necessario richiedere ngStorage e Inject i servizi.

Supponiamo che se `ng-app="myApp"` , si stia iniettando `ngStorage` come segue:

```
var app = angular.module('myApp', ['ngStorage']);
        app.controller('controllerOne', function($localStorage,$sessionStorage) {
        // an object to share
        var sampleObject = {
            name: 'angularjs',
            value: 1
        };
       $localStorage.valueToShare = sampleObject;
       $sessionStorage.valueToShare = sampleObject;
    })
.controller('controllerTwo', function($localStorage,$sessionStorage) {
    console.log('localStorage: '+ $localStorage +'sessionStorage: '+$sessionStorage);
})
```

`$localStorage` e `$sessionStorage` sono accessibili a livello globale attraverso qualsiasi controller fintanto che si iniettano tali servizi nei controller.

Puoi anche utilizzare `localStorage` e `sessionStorage` di `HTML5` . Tuttavia, l'uso di `HTML5 localStorage`

richiede la serializzazione e la deserializzazione degli oggetti prima di utilizzarli o salvarli.

Per esempio:

```
var myObj = {
    firstname: "Nic",
    lastname: "Raboy",
    website: "https://www.google.com"
}
//if you wanted to save into localStorage, serialize it
window.localStorage.set("saved", JSON.stringify(myObj));

//unserialize to get object
var myObj = JSON.parse(window.localStorage.get("saved"));
```

Condivisione dei dati da un controller a un altro tramite il servizio

Possiamo creare un `service` per `set` e `get` i dati tra i `controllers` e quindi iniettare quel servizio nella funzione controller dove vogliamo usarlo.

Servizio :

```
app.service('setGetData', function() {
  var data = '';
    getData: function() { return data; },
    setData: function(requestData) { data = requestData; }
});
```

Controller:

```
app.controller('myCtrl1', ['setGetData',function(setGetData) {

  // To set the data from the one controller
  var data = 'Hello World !!';
  setGetData.setData(data);

}]);

app.controller('myCtrl2', ['setGetData',function(setGetData) {

  // To get the data from the another controller
  var res = setGetData.getData();
  console.log(res); // Hello World !!

}]);
```

Qui, possiamo vedere che `myCtrl1` è usato per `setting` i dati e `myCtrl2` è usato per `getting` i dati. Quindi, possiamo condividere i dati da un controller a un altro controller come questo.

Leggi Condivisione dei dati online: https://riptutorial.com/it/angularjs/topic/1923/condivisione-dei-dati

Capitolo 11: Controller

Sintassi

- <htmlElement ng-controller = "controllerName"> ... </ htmlElement>
- <script> app.controller ('controllerName', controllerFunction); </ Script>

Examples

Il tuo primo controller

Un controller è una struttura di base utilizzata in Angular per preservare l'ambito e gestire determinate azioni all'interno di una pagina. Ogni controller è accoppiato con una vista HTML.

Di seguito è riportato uno standard di base per un'app Angular:

```
<!DOCTYPE html>

<html lang="en" ng-app='MyFirstApp'>
    <head>
        <title>My First App</title>

        <!-- angular source -->
        <script src="https://code.angularjs.org/1.5.3/angular.min.js"></script>

        <!-- Your custom controller code -->
        <script src="js/controllers.js"></script>
    </head>
    <body>
        <div ng-controller="MyController as mc">
            <h1>{{ mc.title }}</h1>
            <p>{{ mc.description }}</p>
            <button ng-click="mc.clicked()">
                Click Me!
            </button>
        </div>
    </body>
</html>
```

Ci sono alcune cose da notare qui:

```
<html ng-app='MyFirstApp'>
```

L'impostazione del nome dell'app con `ng-app` ti consente di accedere all'applicazione in un file Javascript esterno, che verrà trattato di seguito.

```
<script src="js/controllers.js"></script>
```

Avremo bisogno di un file Javascript dove definisci i tuoi controller e le loro azioni / dati.

```
<div ng-controller="MyController as mc">
```

L'attributo `ng-controller` imposta il controller per quell'elemento DOM e tutti gli elementi che sono figli (ricorsivamente) sotto di esso.

Puoi avere più di uno stesso controller (in questo caso, `MyController`) dicendo `... as mc` , stiamo dando a questo istanza del controller un alias.

```
<h1>{{ mc.title }}</h1>
```

La notazione `{{ ... }}` è un'espressione angolare. In questo caso, questo imposterà il testo interno di quell'elemento `<h1>` a qualunque sia il valore di `mc.title` .

Nota: Angular utilizza l'associazione dati bidirezionale, il che significa che indipendentemente dal modo in cui si aggiorna il valore `mc.title` , si rifletterà sia sul controller che sulla pagina.

Si noti inoltre che le espressioni angolari *non* devono fare riferimento a un controller. Un'espressione Angolare può essere semplice come `{{ 1 + 2 }}` o `{{ "Hello " + "World" }}` .

```
<button ng-click="mc.clicked()">
```

`ng-click` è una direttiva Angular, in questo caso vincola l'evento click per il pulsante per attivare la funzione `clicked()` dell'istanza `MyController` .

Con queste cose in mente, scriviamo un'implementazione del controller `MyController` . Con l'esempio sopra, dovresti scrivere questo codice in `js/controller.js` .

Innanzitutto, devi istanziare l'app Angolare nel tuo Javascript.

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

Nota che il nome che passiamo qui è lo stesso del nome che hai impostato nel tuo HTML con la direttiva `ng-app` .

Ora che abbiamo l'oggetto app, possiamo usarlo per creare controller.

```
app.controller('MyController', function(){
    var ctrl = this;

    ctrl.title = "My First Angular App";
    ctrl.description = "This is my first Angular app!";

    ctrl.clicked = function(){
        alert("MyController.clicked()");
    };
});
```

Nota: per tutto ciò che vogliamo far parte dell'istanza del controllore, usiamo `this` parola chiave.

Questo è tutto ciò che è necessario per costruire un controller semplice.

Creazione di controller

```
angular
    .module('app')
    .controller('SampleController', SampleController)

SampleController.$inject = ['$log', '$scope'];
function SampleController($log, $scope){
    $log.debug('*****SampleController******');

    /* Your code below */
}
```

Nota: l' `.$inject` assicurerà che le tue dipendenze non vengano rimescolate dopo il minification. Inoltre, assicurarsi che sia in ordine con la funzione denominata.

Creazione di controllori, Minificazione sicura

Ci sono un paio di modi diversi per proteggere la creazione del tuo controller da minification.

Il primo è chiamato annotazione array in linea. Sembra il seguente:

```
var app = angular.module('app');
app.controller('sampleController', ['$scope', '$http', function(a, b){
    //logic here
}]);
```

Il secondo parametro del metodo controller può accettare una matrice di dipendenze. Come puoi vedere, ho definito `$scope` e `$http` che dovrebbero corrispondere ai parametri della funzione controller in cui `a` sarà `$scope` , e `b` sarebbe `$http` . Prendi nota che l'ultimo elemento dell'array dovrebbe essere la funzione del tuo controller.

La seconda opzione utilizza la proprietà `$inject` . Sembra il seguente:

```
var app = angular.module('app');
app.controller('sampleController', sampleController);
sampleController.$inject = ['$scope', '$http'];
function sampleController(a, b) {
    //logic here
}
```

Questo fa la stessa cosa dell'annotazione dell'array inline ma offre uno stile diverso per quelli che preferiscono un'opzione all'altra.

L'ordine delle dipendenze iniettate è importante

Quando si iniettano dipendenze utilizzando il modulo matrice, assicurarsi che l'elenco delle dipendenze corrisponda al corrispondente elenco di argomenti passati alla funzione controller.

Si noti che nell'esempio seguente, `$scope` e `$http` sono invertiti. Ciò causerà un problema nel codice.

```
// Intentional Bug: injected dependencies are reversed which will cause a problem
app.controller('sampleController', ['$scope', '$http',function($http, $scope) {
    $http.get('sample.json');
}]);
```

Utilizzo di ControllerAs in Angular JS

In `$scope` angolare è la colla tra il controller e la vista che aiuta con tutte le nostre esigenze di associazione dei dati. Controller As è un altro modo di collegare controller e view ed è quasi sempre consigliato. Fondamentalmente questi sono i due costrutti del controller in Angular (cioè $ scope e Controller As).

Diversi modi di usare Controller As are -

controllerAs Visualizza sintassi

```
<div ng-controller="CustomerController as customer">
    {{ customer.name }}
</div>
```

controllerAs Sintassi del controller

```
function CustomerController() {
    this.name = {};
    this.sendMessage = function() { };
}
```

controllerAs con VM

```
function CustomerController() {
    /*jshint validthis: true */
    var vm = this;
    vm.name = {};
    vm.sendMessage = function() { };
}
```

`controllerAs` è zucchero sintattico su `$scope` . È ancora possibile associare alla vista e accedere ancora ai metodi `$scope` . L'utilizzo di `controllerAs` è una delle migliori pratiche suggerite dal team di base angolare. Ci sono molte ragioni per questo, alcune di queste sono -

- `$scope` espone i membri dal controller alla vista tramite un oggetto intermedio. Impostando `this.*` , possiamo esporre solo ciò che vogliamo esporre dal controller alla vista. Inoltre segue il modo standard JavaScript di utilizzare questo.
- usando la sintassi `controllerAs` , abbiamo un codice più leggibile e la proprietà genitore può essere letta usando il nome alias del controllore genitore invece di usare la sintassi `$parent` .
- Promuove l'uso del binding a un oggetto "tratteggiato" nella View (ad esempio,

customer.name anziché name), che è più contestuale, più facile da leggere ed evita qualsiasi problema di riferimento che possa verificarsi senza "punteggiatura".

- Aiuta a evitare l'uso `$parent` chiamate `$parent` in Views con i controller nidificati.
- Utilizzare una variabile di cattura per questo quando si utilizza la sintassi del `controllerAs` . Scegli un nome di variabile consistente come `vm` , che sta per ViewModel. Perché `this` parola chiave è contestuale e se utilizzata all'interno di una funzione all'interno di un controller può cambiare il suo contesto. Catturare il contesto di questo evita di incontrare questo problema.

NOTA: usando la sintassi `controllerAs` aggiungere al riferimento corrente dello scope al controller corrente, quindi disponibile come campo

```
<div ng-controller="Controller as vm>...</div>
```

`vm` è disponibile come `$scope.vm` .

Creazione di controller angolari sicuri per la sicurezza

Per creare controllori angolari sicuri per minificazione, cambierai i parametri della funzione del `controller` .

Il secondo argomento nella funzione `module.controller` deve essere passato a un **array** , dove l' **ultimo parametro** è la **funzione controller** e ogni parametro prima è il **nome** di ciascun valore iniettato.

Questo è diverso dal normale paradigma; che prende la **funzione** del **controller** con gli argomenti iniettati.

Dato:

```
var app = angular.module('myApp');
```

Il controller dovrebbe assomigliare a questo:

```
app.controller('ctrlInject',
    [
        /* Injected Parameters */
        '$Injectable1',
        '$Injectable2',
        /* Controller Function */
        function($injectable1Instance, $injectable2Instance) {
            /* Controller Content */
        }
    ]
);
```

Nota: *non è necessario che i nomi dei parametri iniettati corrispondano, ma verranno associati in ordine.*

Questo ridurrà a qualcosa di simile a questo:

```
var
a=angular.module('myApp');a.controller('ctrlInject',['$Injectable1','$Injectable2',function(b,c){/*
Controller Content */}]);
```

Il processo di minificazione sostituirà ogni istanza di `app` con `a` istanza di `$Injectable1Instance` con `b` e ogni istanza di `$Injectable2Instance` con `c` .

Controller annidati

Anche i controller di nidificazione incatenano il `$scope` . La modifica di una variabile `$scope` nel controller nidificato modifica la stessa variabile `$scope` nel controller principale.

```
.controller('parentController', function ($scope) {
    $scope.parentVariable = "I'm the parent";
});

.controller('childController', function ($scope) {
    $scope.childVariable = "I'm the child";

    $scope.childFunction = function () {
        $scope.parentVariable = "I'm overriding you";
    };
});
```

Ora proviamo a gestirli entrambi, annidati.

```
<body ng-controller="parentController">
    What controller am I? {{parentVariable}}
    <div ng-controller="childController">
        What controller am I? {{childVariable}}
        <button ng-click="childFunction()"> Click me to override! </button>
    </div>
</body>
```

I controller di nidificazione possono avere i suoi benefici, ma una cosa deve essere tenuta a mente quando si fa così. La chiamata alla direttiva `ngController` crea una nuova istanza del controller, che può spesso generare confusione e risultati imprevisti.

Leggi Controller online: https://riptutorial.com/it/angularjs/topic/601/controller

Capitolo 12: Controller con ES6

Examples

controllore

è molto facile scrivere un controller angularJS con ES6 se si è familiarizzati con la **programmazione orientata agli oggetti** :

```
class exampleContoller{

       constructor(service1,service2,...serviceN){
          let ctrl=this;
              ctrl.service1=service1;
              ctrl.service2=service2;
               .
               .
               .
              ctrl.service1=service1;
              ctrl.controllerName = 'Example Controller';
              ctrl.method1(controllerName)

       }

       method1(param){
          let ctrl=this;
               ctrl.service1.serviceFunction();
              .
              .
              ctrl.scopeName=param;
       }
       .
       .
       .
       methodN(param){
          let ctrl=this;
               ctrl.service1.serviceFunction();
              .
              .
       }


   }
   exampleContoller.$inject = ['service1','service2',...,'serviceN'];
   export default exampleContoller;
```

Leggi Controller con ES6 online: https://riptutorial.com/it/angularjs/topic/9419/controller-con-es6

Capitolo 13: Convalida del modulo

Examples

Convalida del modulo di base

Uno dei punti di forza di Angular è la convalida della forma lato client.

Affrontare gli input di form tradizionali e dover usare l'elaborazione interrogativa in stile jQuery può essere dispendioso in termini di tempo e pignolo. Angular consente di produrre facilmente forme *interattive* professionali.

La direttiva **ng-model** fornisce l'associazione bidirezionale con i campi di input e solitamente l'attributo **novalidate** viene anche inserito sull'elemento form per impedire al browser di eseguire la convalida nativa.

Quindi, una forma semplice sarebbe simile a:

```
<form name="form" novalidate>
  <label name="email"> Your email </label>
  <input type="email" name="email" ng-model="email" />
</form>
```

Affinché Angular convalidi gli input, utilizzare esattamente la stessa sintassi di un normale elemento di *input* , fatta eccezione per l'aggiunta dell'attributo **ng-model** per specificare la variabile da associare all'ambito. L'email è mostrata nell'esempio precedente. Per convalidare un numero, la sintassi sarebbe:

```
<input type="number" name="postalcode" ng-model="zipcode" />
```

I passaggi finali per la convalida del modulo di base sono la connessione a una funzione di invio del modulo sul controller utilizzando **ng-submit** , anziché consentire l'invio del modulo predefinito. Questo non è obbligatorio, ma viene solitamente utilizzato, poiché le variabili di input sono già disponibili nell'ambito e quindi disponibili per la funzione di invio. Di solito è anche una buona pratica dare un nome al modulo. Queste modifiche comporterebbero la seguente sintassi:

```
<form name="signup_form" ng-submit="submitFunc()" novalidate>
  <label name="email"> Your email </label>
  <input type="email" name="email" ng-model="email" />
  <button type="submit">Signup</button>
</form>
```

Questo codice sopra è funzionale, ma esistono altre funzionalità fornite da Angular.

Il passo successivo è capire che Angular allega gli attributi di classe usando **ng-pristine** , **ng-dirty** , **ng-valid** e **ng-invalid** per l'elaborazione dei moduli. L'utilizzo di queste classi nel tuo css ti consentirà di **definire i** campi di input **validi / non validi, pristine / dirty** e quindi di alterare la

presentazione mentre l'utente sta inserendo i dati nel modulo.

Forma e Stati di input

Forme e input angolari hanno vari stati che sono utili quando si convalida il contenuto

Stati di input

Stato	Descrizione
`$touched`	Field è stato toccato
`$untouched`	Il campo non è stato toccato
`$pristine`	Il campo non è stato modificato
`$dirty`	Il campo è stato modificato
`$valid`	Il contenuto del campo è valido
`$invalid`	Il contenuto del campo non è valido

Tutti gli stati sopra riportati sono proprietà booleane e possono essere sia vere che false.

Con questi, è molto facile visualizzare messaggi a un utente.

```
<form name="myForm" novalidate>
    <input name="myName" ng-model="myName" required>
    <span ng-show="myForm.myName.$touched && myForm.myName.$invalid">This name is
invalid</span>
</form>
```

Qui, stiamo usando la direttiva `ng-show` per mostrare un messaggio ad un utente se hanno modificato un modulo ma non è valido.

Classi CSS

Angular fornisce anche alcune classi CSS per moduli e input a seconda del loro stato

Classe	Descrizione
`ng-touched`	Field è stato toccato
`ng-untouched`	Il campo non è stato toccato
`ng-pristine`	Il campo non è stato modificato
`ng-dirty`	Il campo è stato modificato
`ng-valid`	Il campo è valido

Classe	Descrizione
`ng-invalid`	Il campo non è valido

Puoi usare queste classi per aggiungere stili ai tuoi moduli

```
input.ng-invalid {
  background-color: crimson;
}
input.ng-valid {
  background-color: green;
}
```

ngMessages

`ngMessages` viene utilizzato per migliorare lo stile per la visualizzazione dei messaggi di convalida nella vista.

Approccio tradizionale

Prima di `ngMessages` , normalmente visualizziamo i messaggi di convalida usando le direttive predefinite angolari `ng-class` . Questo approccio consisteva in rifiuti e un'attività `repetitive` .

Ora, usando `ngMessages` , possiamo creare i nostri messaggi personalizzati.

Esempio

Html:

```
<form name="ngMessagesDemo">
  <input name="firstname" type="text" ng-model="firstname" required>
  <div ng-messages="ngMessagesDemo.firstname.$error">
    <div ng-message="required">Firstname is required.</div>
  </div>
</form>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.3.16/angular.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.3.16/angular-
messages.min.js"></script>
```

JS:

```
var app = angular.module('app', ['ngMessages']);

app.controller('mainCtrl', function ($scope) {
    $scope.firstname = "Rohit";
});
```

Convalida del modulo personalizzato

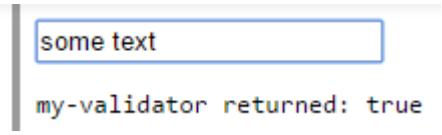
In alcuni casi la convalida di base non è sufficiente. Convalida personalizzata del supporto angolare che aggiunge funzioni di `$validators` all'oggetto `$validators` su `ngModelController` :

```
angular.module('app', [])
  .directive('myValidator', function() {
    return {
      // element must have ng-model attribute
      // or $validators does not work
      require: 'ngModel',
      link: function(scope, elm, attrs, ctrl) {
        ctrl.$validators.myValidator = function(modelValue, viewValue) {
            // validate viewValue with your custom logic
            var valid = (viewValue && viewValue.length > 0) || false;
            return valid;
        };
      }
    };
```

Il validatore è definito come una direttiva che richiede `ngModel` , quindi per applicare il validatore basta aggiungere la direttiva personalizzata al controllo del modulo di input.

```
<form name="form">
  <input type="text"
         ng-model="model"
         name="model"
         my-validator>
  <pre ng-bind="'my-validator returned: ' + form.model.$valid"></pre>
</form>
```

E il `my-validator` non deve essere applicato al controllo dei moduli nativi. Può essere qualsiasi elemento, purché come `ng-model` nei suoi attributi. Ciò è utile quando hai un componente ui di build personalizzato.



Moduli nidificati

A volte è preferibile annidare le forme allo scopo di raggruppare i controlli e gli ingressi logicamente sulla pagina. Tuttavia, i moduli HTML5 non devono essere nidificati. Forniture angolari invece di `ng-form` .

```
<form name="myForm" noValidate>
  <!-- nested form can be referenced via 'myForm.myNestedForm' -->
  <ng-form name="myNestedForm" noValidate>
    <input name="myInput1" ng-minlength="1" ng-model="input1" required />
    <input name="myInput2" ng-minlength="1" ng-model="input2" required />
  </ng-form>

  <!-- show errors for the nested subform here -->
  <div ng-messages="myForm.myNestedForm.$error">
    <!-- note that this will show if either input does not meet the minimum -->
```

```
    <div ng-message="minlength">Length is not at least 1</div>
  </div>
</form>

<!-- status of the form -->
<p>Has any field on my form been edited? {{myForm.$dirty}}</p>
<p>Is my nested form valid? {{myForm.myNestedForm.$valid}}</p>
<p>Is myInput1 valid? {{myForm.myNestedForm.myInput1.$valid}}</p>
```

Ogni parte del modulo contribuisce allo stato generale del modulo. Pertanto, se uno degli input `myInput1` è stato modificato ed è `$dirty` , il suo modulo contenente sarà anche `$dirty` . Questa cascata a ogni modulo che contiene, in modo che entrambi `myNestedForm` e `myForm` sarà `$dirty` .

Validatori asincroni

I validatori asincroni consentono di convalidare le informazioni del modulo sul back-end (utilizzando $ http).

Questi tipi di validatori sono necessari quando è necessario accedere alle informazioni memorizzate sul server che non si possono avere sul client per vari motivi, come la tabella degli utenti e altre informazioni sul database.

Per utilizzare i validatori asincroni, si accede al `ng-model` del proprio `input` e si definiscono le funzioni di callback per la proprietà `$asyncValidators` .

Esempio:

L'esempio seguente controlla se esiste già un nome fornito, il backend restituirà uno stato che rifiuterà la promessa se il nome esiste già o se non è stato fornito. Se il nome non esiste, restituirà una promessa risolta.

```
ngModel.$asyncValidators.usernameValidate = function (name) {
     if (name) {
          return AuthenticationService.checkIfNameExists(name); // returns a promise
     } else {
          return $q.reject("This username is already taken!"); // rejected promise
     }
};
```

Ora ogni volta che viene modificato il `ng-model` dell'ingresso, questa funzione verrà eseguita e restituirà una promessa con il risultato.

Leggi Convalida del modulo online: https://riptutorial.com/it/angularjs/topic/3979/convalida-del-modulo

Capitolo 14: costanti

Osservazioni

MAIUSCUNTO la tua costante : la scrittura costante nel capitale è una pratica comune comune a molte lingue. È anche utile identificare chiaramente la natura degli elementi iniettati:

Quando vedi `.controller('MyController', function($scope, Profile, EVENT))` , sai immediatamente che:

- `$scope` è un elemento angolare
- `Profile` è un servizio o fabbrica personalizzato
- `EVENT` è una costante angolare

Examples

Crea la tua prima costante

```
angular
  .module('MyApp', [])
  .constant('VERSION', 1.0);
```

La tua costante è ora dichiarata e può essere iniettata in un controller, un servizio, una fabbrica, un fornitore e anche in un metodo di configurazione:

```
angular
  .module('MyApp')
  .controller('FooterController', function(VERSION) {
    this.version = VERSION;
  });
```

```
<footer ng-controller="FooterController as Footer">{{ Footer.version }}</footer>
```

Casi d'uso

Non c'è nessuna rivoluzione qui, ma la costante angolare può essere utile specialmente quando la tua applicazione e / o il tuo team iniziano a crescere ... o se semplicemente ami scrivere codice bello!

- **Codice del refactore** Esempio con i nomi degli eventi. Se usi molti eventi nella tua applicazione, i nomi degli eventi sono un po 'ovunque. A quando un nuovo sviluppatore si unisce alla tua squadra, nomina i suoi eventi con una sintassi diversa, ... Puoi facilmente evitarlo raggruppando i nomi degli eventi in una costante:

  ```
  angular
  ```

```
  .module('MyApp')
  .constant('EVENTS', {
    LOGIN_VALIDATE_FORM: 'login::click-validate',
    LOGIN_FORGOT_PASSWORD: 'login::click-forgot',
    LOGIN_ERROR: 'login::notify-error',
    ...
  });
```

```
angular
  .module('MyApp')
  .controller('LoginController', function($scope, EVENT) {
    $scope.$on(EVENT.LOGIN_VALIDATE_FORM, function() {
      ...
    });
  })
```

... e ora i nomi del tuo evento possono trarre benefici dal completamento automatico!

- **Definire la configurazione.** Trova tutta la tua configurazione nello stesso posto:

```
angular
  .module('MyApp')
  .constant('CONFIG', {
    BASE_URL: {
      APP: 'http://localhost:3000',
      API: 'http://localhost:3001'
    },
    STORAGE: 'S3',
    ...
  });
```

- **Isolare le parti.** A volte, ci sono alcune cose di cui non sei molto orgoglioso ... ad esempio il valore di hardcoded. Invece di lasciarli nel tuo codice principale, puoi creare una costante angolare

```
angular
  .module('MyApp')
  .constant('HARDCODED', {
    KEY: 'KEY',
    RELATION: 'has_many',
    VAT: 19.6
  });
```

... e refactoring qualcosa di simile

```
$scope.settings = {
  username: Profile.username,
  relation: 'has_many',
  vat: 19.6
}
```

a

```
$scope.settings = {
  username: Profile.username,
  relation: HARDCODED.RELATION,
  vat: HARDCODED.VAT
}
```

Leggi costanti online: https://riptutorial.com/it/angularjs/topic/3967/costanti

Capitolo 15: Debug

Examples

Debug di base nel markup

Test dell'ottica e produzione del modello

```
<div ng-app="demoApp" ng-controller="mainController as ctrl">
    {{$id}}
    <ul>
        <li ng-repeat="item in ctrl.items">
            {{$id}}<br/>
            {{item.text}}
        </li>
    </ul>
    {{$id}}
    <pre>
       {{ctrl.items | json : 2}}
    </pre>
</div>


angular.module('demoApp', [])
.controller('mainController', MainController);

function MainController() {
    var vm = this;
    vm.items = [{
        id: 0,
        text: 'first'
    },
    {
        id: 1,
        text: 'second'
    },
    {
        id: 2,
        text: 'third'
    }];
}
```

A volte può essere d'aiuto vedere se esiste un nuovo ambito per risolvere i problemi dell'ambito. `$scope.$id` può essere usato in un'espressione ovunque nel markup per vedere se c'è un nuovo $ scope.

Nell'esempio si può vedere che al di fuori del ul-tag è lo stesso scope ($ id = 2) e all'interno di `ng-repeat` ci sono nuovi ambiti figlio per ogni iterazione.

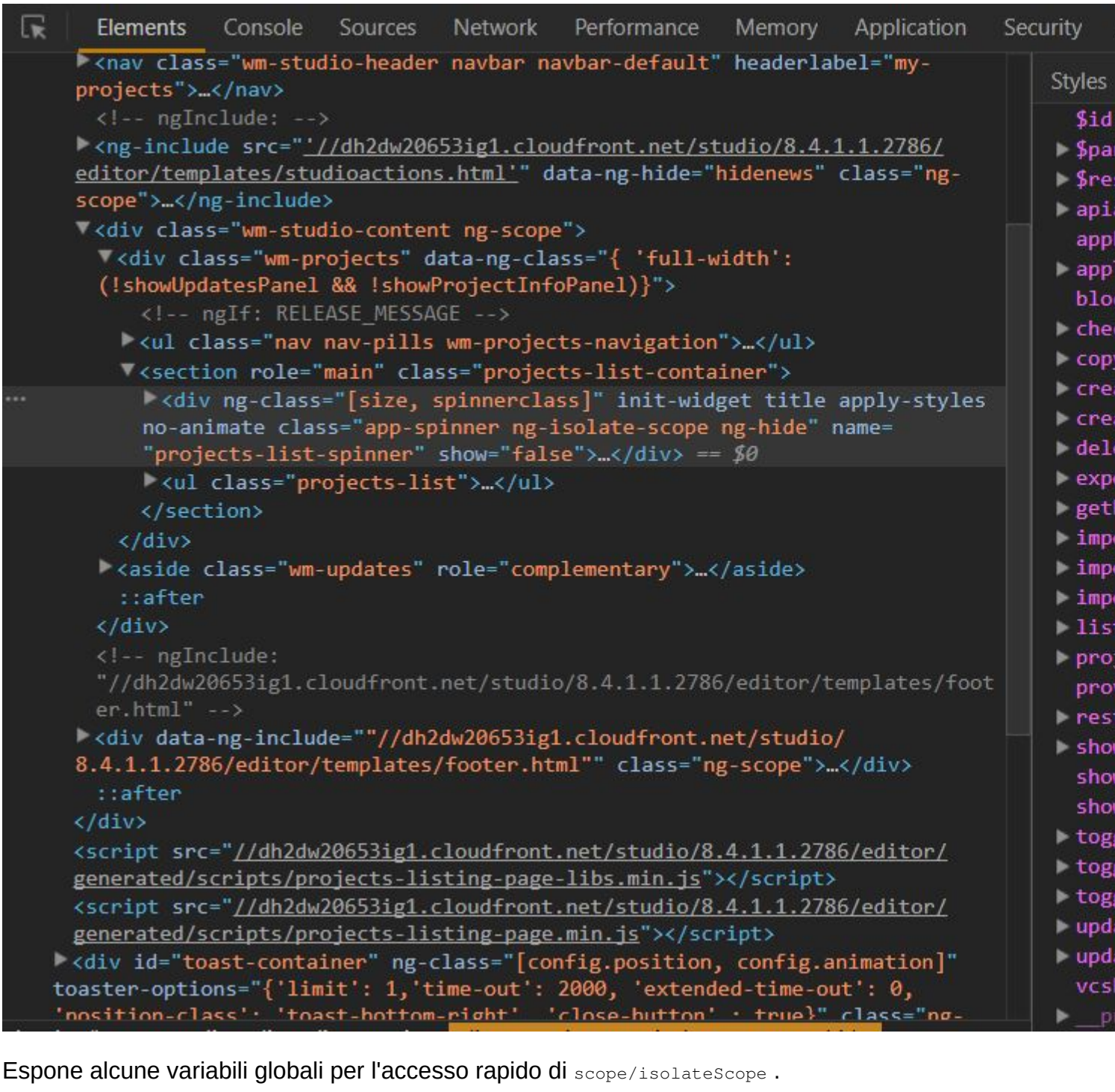
Un output del modello in un pre-tag è utile per vedere i dati correnti del tuo modello. Il filtro `json` crea un output formattato piacevole. Il pre-tag è usato perché all'interno di quel tag qualsiasi carattere di nuova riga `\n` verrà visualizzato correttamente.

dimostrazione

Utilizzando l'estensione chrome di ng-inspect

ng-inspect è una leggera estensione Chrome per il debug delle applicazioni AngularJS.

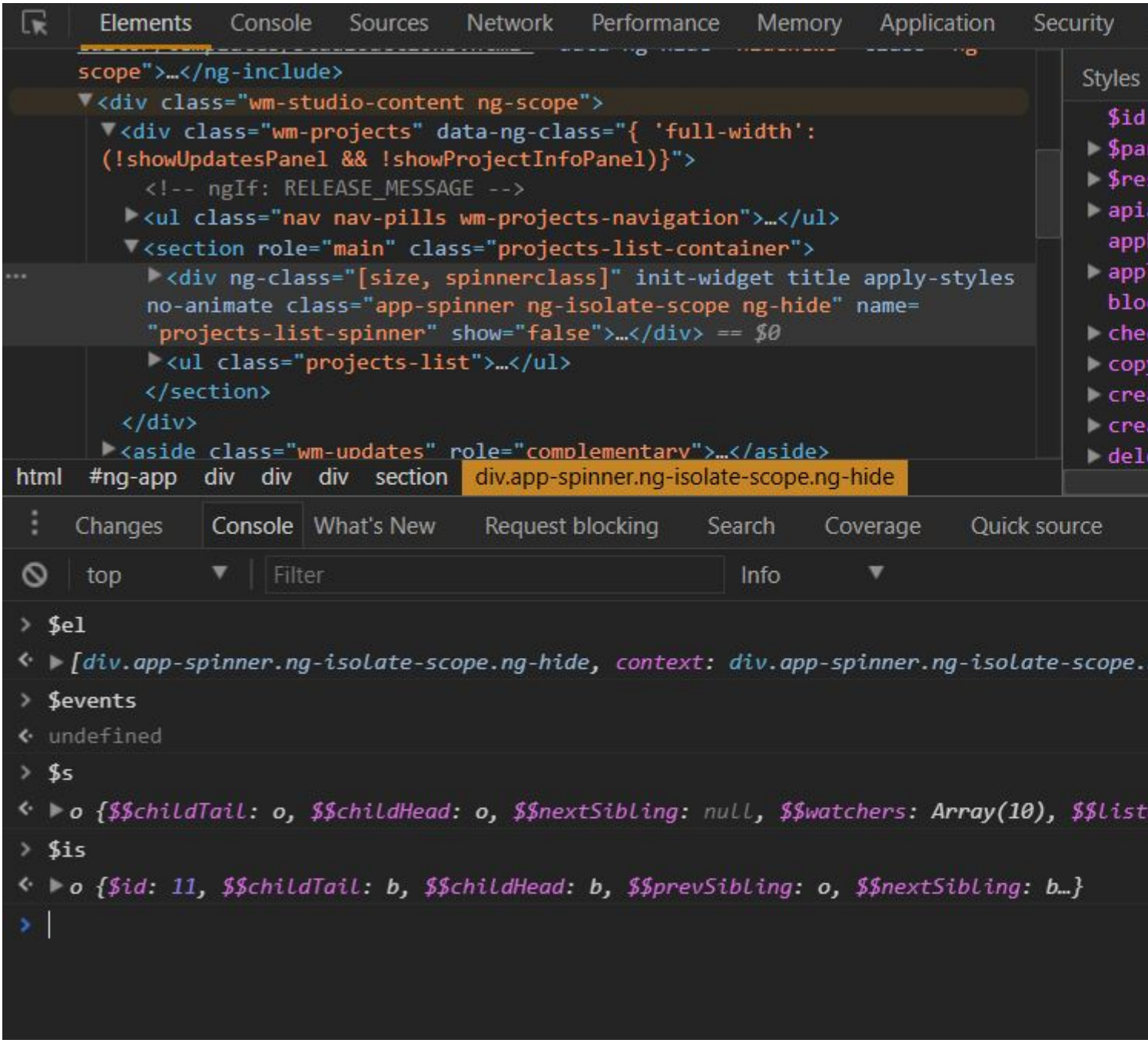
Quando viene selezionato un nodo dal pannello degli elementi, le informazioni relative all'ambito vengono visualizzate nel pannello di ispezione.



Espone alcune variabili globali per l'accesso rapido di `scope/isolateScope` .

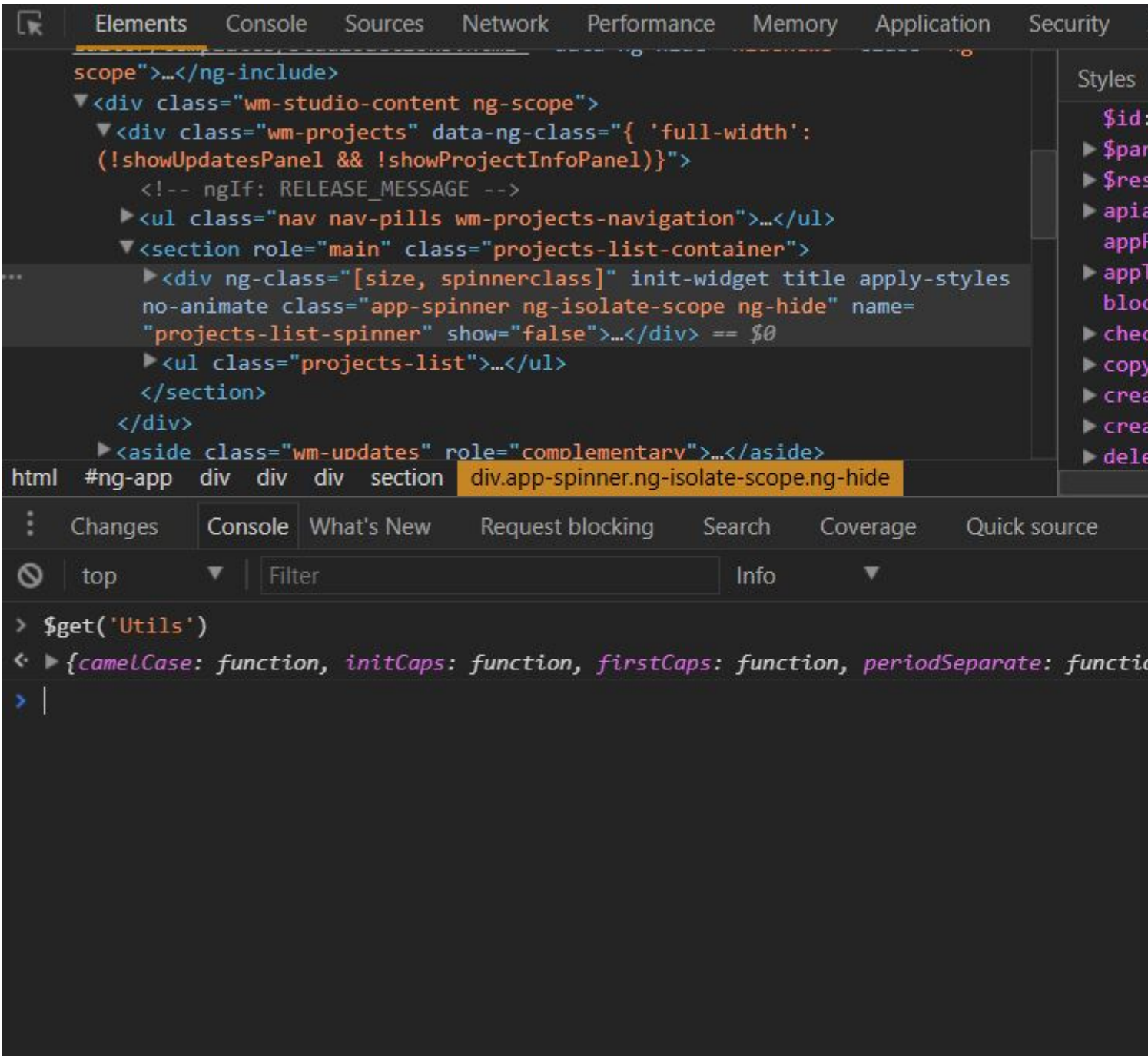
```
$s      -- scope of the selected node
$is     -- isolateScope of the selected node
$el     -- jQuery element reference of the selected node (requiers jQuery)
```

```
$events -- events present on the selected node (requires jQuery)
```



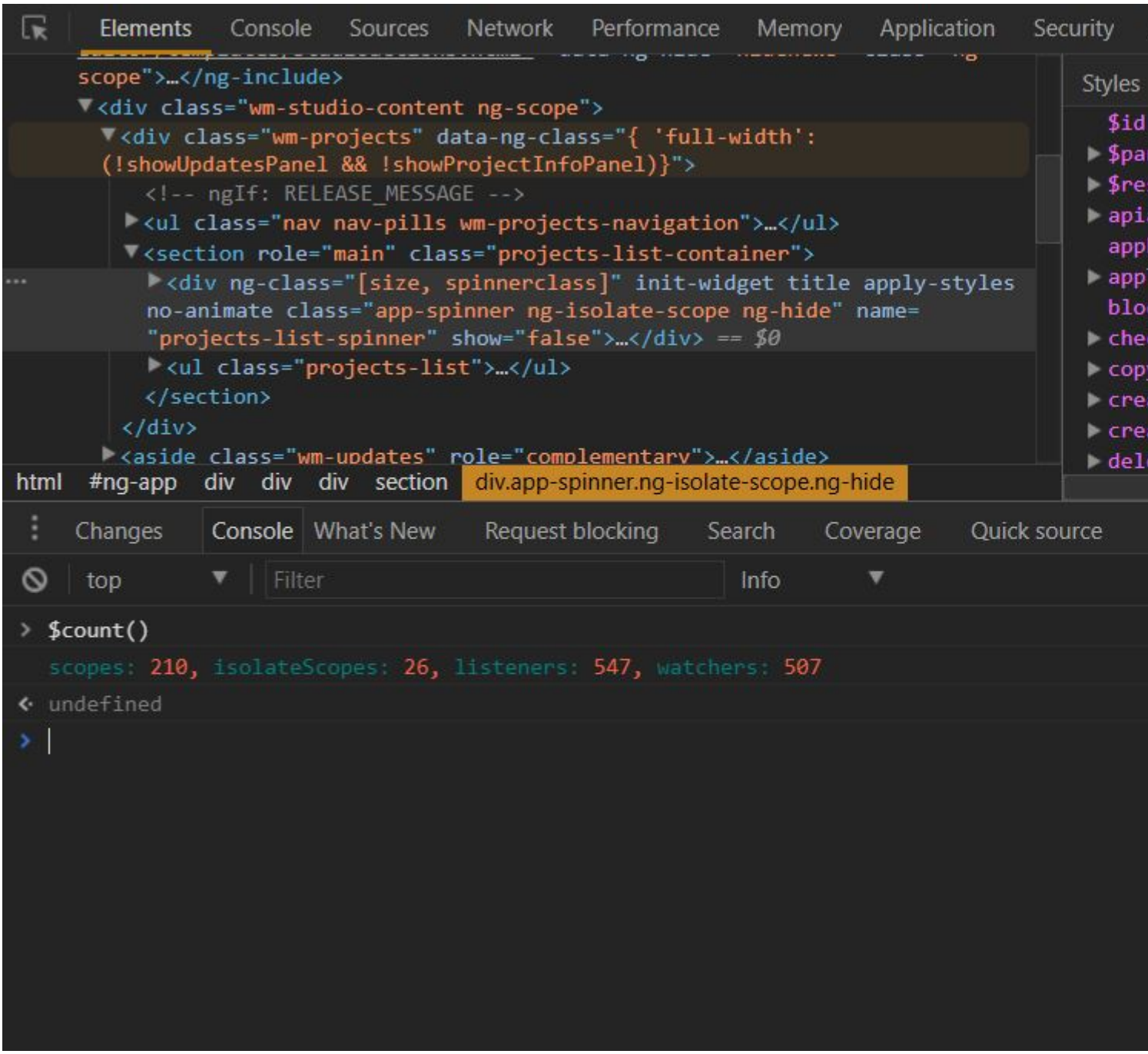
Fornisce un facile accesso a servizi / fabbriche.

Utilizzare `$get()` per recuperare l'istanza di un servizio / fabbrica per nome.



Le prestazioni dell'applicazione possono essere monitorate contando il no.of scopes, isolationScopes, watcher e ascoltatori sull'applicazione.

Usa `$count()` per ottenere il conteggio degli ambiti, isolareScopes, osservatori e ascoltatori.



Nota: questa estensione funzionerà solo quando il debugInfo è abilitato.

Scarica ng-inspect qui

Ottenere l'ambito dell'elemento

In un'app angolare tutto gira attorno all'ambito, se potessimo ottenere uno scope di elementi allora è facile eseguire il debug dell'applicazione angolare. Come accedere all'ambito dell'elemento:

```
angular.element(myDomElement).scope();
e.g.
angular.element(document.getElementById('yourElementId')).scope()  //accessing by ID
```

Ottenere lo scopo del controller: -

```
angular.element('[ng-controller=ctrl]').scope()
```

Un altro modo semplice per accedere a un elemento DOM dalla console (come menzionato da jm) è di fare clic su di esso nella scheda 'elementi' e viene automaticamente salvato come $ 0.

```
angular.element($0).scope();
```

Leggi Debug online: https://riptutorial.com/it/angularjs/topic/4761/debug

Capitolo 16: decoratori

Sintassi

- decoratore (nome, decoratore);

Osservazioni

> **Decoratore** è una funzione che consente di modificare un servizio , fabbrica , direttiva o filtro prima del suo utilizzo. Decorator viene utilizzato per sovrascrivere o modificare il comportamento del servizio. Il valore di ritorno della funzione decoratore può essere il servizio originale o un nuovo servizio che sostituisce o avvolge e delega il servizio originale.

Qualsiasi decorazione ***deve essere*** eseguita nella fase di `config` dell'applicazione angolare iniettando `$provide` e utilizzando la funzione `$provide.decorator` .

> La funzione decoratore ha un oggetto `$delegate` iniettato per fornire l'accesso al servizio che corrisponde al selettore nel decoratore. Questo `$delegate` sarà il servizio che stai decorando. Il valore di ritorno della funzione fornita al decoratore avverrà in base al servizio, alla direttiva o al filtro che viene decorato.

Si dovrebbe considerare l'utilizzo del decoratore solo se qualsiasi altro approccio non è appropriato o si rivela troppo noioso. Se un'applicazione di grandi dimensioni utilizza lo stesso servizio e una parte modifica il comportamento del servizio, è facile creare confusione e / o bug nel processo.

Un tipico caso d'uso sarebbe quando si ha una dipendenza di terze parti che non è possibile aggiornare, ma è necessario che funzioni in modo leggermente diverso o la estenda.

Examples

Decora il servizio, fabbrica

Di seguito è riportato un esempio di decoratore del servizio, che sostituisce la data `null` restituita dal servizio.

```
angular.module('app', [])
  .config(function($provide) {
    $provide.decorator('myService', function($delegate) {
      $delegate.getDate = function() { // override with actual date object
        return new Date();
      };
      return $delegate;
    });
```

```
  })
  .service('myService', function() {
    this.getDate = function() {
      return null; // w/o decoration we'll be returning null
    };
  })
  .controller('myController', function(myService) {
    var vm = this;
    vm.date = myService.getDate();
  });
```

```
<body ng-controller="myController as vm">
  <div ng-bind="vm.date | date:'fullDate'"></div>
</body>
```

> Saturday, August 6, 2016

Decora la direttiva

Le direttive possono essere decorate esattamente come i servizi e possiamo modificare o sostituire qualsiasi funzionalità. Si noti che la direttiva stessa è accessibile alla posizione 0 in $ delegate array e il parametro name nel decoratore deve includere il suffisso `Directive` (maiuscole e minuscole).

Quindi, se la direttiva è denominata `myDate` , è possibile accedervi utilizzando `myDateDirective` utilizzando `$delegate[0]` .

Di seguito è riportato un semplice esempio in cui la direttiva mostra l'ora corrente. Lo decoreremo per aggiornare l'ora corrente a intervalli di un secondo. Senza decorazione mostrerà sempre lo stesso tempo.

```
<body>
  <my-date></my-date>
</body>
```

```
angular.module('app', [])
  .config(function($provide) {
    $provide.decorator('myDateDirective', function($delegate, $interval) {
      var directive = $delegate[0]; // access directive

      directive.compile = function() { // modify compile fn
        return function(scope) {
          directive.link.apply(this, arguments);
          $interval(function() {
            scope.date = new Date(); // update date every second
          }, 1000);
        };
      };

      return $delegate;
    });
```

```
  })
  .directive('myDate', function() {
    return {
      restrict: 'E',
      template: '<span>Current time is {{ date | date:\'MM:ss\' }}</span>',
      link: function(scope) {
        scope.date = new Date(); // get current date
      }
    };
  });
```

> Current time is 08:33

Decorare il filtro

Quando si decorano i filtri, il parametro name deve includere il suffisso `Filter` (case sensitive). Se il filtro è chiamato `repeat` , il parametro decorator è `repeatFilter` . Qui sotto decoreremo un filtro personalizzato che ripete una determinata stringa *n* volte in modo che il risultato sia invertito. Puoi anche decorare i filtri incorporati di angolare nello stesso modo, sebbene non sia raccomandato in quanto può influire sulla funzionalità del framework.

```
<body>
  <div ng-bind="'i can haz cheeseburger ' | repeat:2"></div>
</body>

angular.module('app', [])
  .config(function($provide) {
    $provide.decorator('repeatFilter', function($delegate) {
      return function reverse(input, count) {
        // reverse repeated string
        return ($delegate(input, count)).split('').reverse().join('');
      };
    });
  })
  .filter('repeat', function() {
    return function(input, count) {
      // repeat string n times
      return (input || '').repeat(count || 1);
    };
  });
```

> i can haz cheeseburger i can haz cheeseburger

> regrubeseehc zah nac i regrubeseehc zah nac i

Leggi decoratori online: https://riptutorial.com/it/angularjs/topic/5255/decoratori

Capitolo 17: direttiva ng-class

Examples

Tre tipi di espressioni di classe ng

Angolare supporta tre tipi di espressioni nella direttiva `ng-class` .

1. Stringa

```
<span ng-class="MyClass">Sample Text</span>
```

La specifica di un'espressione che valuta una stringa indica a Angular di trattarla come una variabile $ scope. Angular controllerà l'ambito $ e cercherà una variabile chiamata "MyClass". Qualsiasi testo sia contenuto in "MyClass" diventerà il nome effettivo della classe che viene applicato a questo `<span>` . Puoi specificare più classi separando ogni classe con uno spazio.

Nel tuo controller, potresti avere una definizione simile a questa:

```
$scope.MyClass = "bold-red deleted error";
```

Angular valuterà l'espressione `MyClass` e troverà la definizione $ scope. Applicherà le tre classi "bold-red", "deleted" e "error" all'elemento `<span>` .

Specificando le classi in questo modo è possibile modificare facilmente le definizioni di classe nel controller. Ad esempio, potrebbe essere necessario modificare la classe in base ad altre interazioni utente o nuovi dati caricati dal server. Inoltre, se hai molte espressioni da valutare, puoi farlo in una funzione che definisce l'elenco finale di classi in una variabile `$scope` . Questo può essere più facile che provare a spremere molte valutazioni nell'attributo `ng-class` nel modello HTML.

---

2. Oggetto

Questo è il modo più comunemente usato per definire le classi usando `ng-class` perché consente facilmente di specificare valutazioni che determinano quale classe usare.

Specificare un oggetto contenente coppie chiave-valore. La chiave è il nome della classe che verrà applicato se il valore (un condizionale) viene valutato come true.

```
<style>
    .red { color: red; font-weight: bold; }
    .blue { color: blue; }
    .green { color: green; }
```

```
    .highlighted { background-color: yellow; color: black; }
</style>

<span ng-class="{ red: ShowRed, blue: ShowBlue, green: ShowGreen, highlighted: IsHighlighted
}">Sample Text</span>

<div>Red: <input type="checkbox" ng-model="ShowRed"></div>
<div>Green: <input type="checkbox" ng-model="ShowGreen"></div>
<div>Blue: <input type="checkbox" ng-model="ShowBlue"></div>
<div>Highlight: <input type="checkbox" ng-model="IsHighlighted"></div>
```

---

3. Matrice

Un'espressione che valuta un array consente di utilizzare una combinazione di **stringhe** (vedere n. 1 sopra) e **oggetti condizionali** (n. 2 sopra).

```
<style>
    .bold { font-weight: bold; }
    .strike { text-decoration: line-through; }
    .orange { color: orange; }
</style>

<p ng-class="[ UserStyle, {orange: warning} ]">Array of Both Expression Types</p>
<input ng-model="UserStyle" placeholder="Type 'bold' and/or 'strike'"><br>
<label><input type="checkbox" ng-model="warning"> warning (apply "orange" class)</label>
```

Ciò crea un campo di input di testo associato alla variabile scope `UserStyle` che consente all'utente di digitare qualsiasi nome di classe. Questi saranno applicati dinamicamente all'elemento `<p>` mentre l'utente digita. Inoltre, l'utente può fare clic sulla casella di controllo associata ai dati alla variabile dell'ambito di `warning` . Questo sarà anche applicato dinamicamente all'elemento `<p>` .

Leggi direttiva ng-class online: https://riptutorial.com/it/angularjs/topic/2395/direttiva-ng-class

Capitolo 18: Direttive che utilizzano ngModelController

Examples

Un semplice controllo: valutazione

Costruiamo un semplice controllo, un widget di valutazione, destinato a essere utilizzato come:

```
<rating min="0" max="5" nullifier="true" ng-model="data.rating"></rating>
```

Per ora non ci sono CSS fantasiosi; questo renderebbe come:

```
0 1 2 3 4 5 x
```

Cliccando su un numero seleziona quella valutazione; e facendo clic su "x" imposta la valutazione su null.

```
app.directive('rating', function() {

    function RatingController() {
        this._ngModel = null;
        this.rating = null;
        this.options = null;
        this.min = typeof this.min === 'number' ? this.min : 1;
        this.max = typeof this.max === 'number' ? this.max : 5;
    }

    RatingController.prototype.setNgModel = function(ngModel) {
        this._ngModel = ngModel;

        if( ngModel ) {
            // KEY POINT 1
            ngModel.$render = this._render.bind(this);
        }
    };

    RatingController.prototype._render = function() {
        this.rating = this._ngModel.$viewValue != null ? this._ngModel.$viewValue : -
Number.MAX_VALUE;
    };

    RatingController.prototype._calculateOptions = function() {
        if( this.min == null || this.max == null ) {
            this.options = [];
        }
        else {
            this.options = new Array(this.max - this.min + 1);
            for( var i=0; i < this.options.length; i++ ) {
                this.options[i] = this.min + i;
            }
        }
```

```
    };

    RatingController.prototype.setValue = function(val) {
        this.rating = val;
        // KEY POINT 2
        this._ngModel.$setViewValue(val);
    };

    // KEY POINT 3
    Object.defineProperty(RatingController.prototype, 'min', {
        get: function() {
            return this._min;
        },
        set: function(val) {
            this._min = val;
            this._calculateOptions();
        }
    });

    Object.defineProperty(RatingController.prototype, 'max', {
        get: function() {
            return this._max;
        },
        set: function(val) {
            this._max = val;
            this._calculateOptions();
        }
    });

    return {
        restrict: 'E',
        scope: {
            // KEY POINT 3
            min: '<?',
            max: '<?',
            nullifier: '<?'
        },
        bindToController: true,
        controllerAs: 'ctrl',
        controller: RatingController,
        require: ['rating', 'ngModel'],
        link: function(scope, elem, attrs, ctrls) {
            ctrls[0].setNgModel(ctrls[1]);
        },
        template:
            '<span ng-repeat="o in ctrl.options" href="#" class="rating-option" ng-
class="{\'rating-option-active\': o <= ctrl.rating}" ng-click="ctrl.setValue(o)">{{ o
}}</span>' +
            '<span ng-if="ctrl.nullifier" ng-click="ctrl.setValue(null)" class="rating-
nullifier">&#10006;</span>'
    };
});
```

Punti chiave:

1. Implementa `ngModel.$render` per trasferire il *valore* della *vista* del modello alla tua vista.
2. Chiama `ngModel.$setViewValue()` ogni volta che ritieni che il valore della vista debba essere aggiornato.
3. Il controllo può naturalmente essere parametrizzato; utilizzare `'<'` scope scope per i

parametri, se in Angular> = 1.5 per indicare chiaramente input - unidirezionale. Se devi agire ogni volta che un parametro cambia, puoi usare una proprietà JavaScript (vedi `Object.defineProperty()`) per salvare alcuni orologi.

Nota 1: per non complicare eccessivamente l'implementazione, i valori di valutazione sono inseriti in una matrice: `ctrl.options` . Questo non è necessario; Un'implementazione più efficiente, ma anche più complessa, potrebbe utilizzare la manipolazione del DOM per inserire / rimuovere le valutazioni quando si modifica `min` / `max` .

Nota 2: con l'eccezione delle associazioni di ambito `'<'` , questo esempio può essere utilizzato in Angular <1.5. Se si utilizza Angular> = 1.5, sarebbe una buona idea trasformarlo in un componente e utilizzare l'hook del ciclo di vita `$onInit()` per inizializzare `min` e `max` , invece di farlo nel costruttore del controllore.

E un violino necessario: https://jsfiddle.net/h81mgxma/

Un paio di controlli complessi: modifica un oggetto completo

Un controllo personalizzato non deve limitarsi a cose banali come i primitivi; può modificare cose più interessanti. Qui presentiamo due tipi di controlli personalizzati, uno per la modifica delle persone e uno per la modifica degli indirizzi. Il controllo dell'indirizzo viene utilizzato per modificare l'indirizzo della persona. Un esempio di utilizzo sarebbe:

```
<input-person ng-model="data.thePerson"></input-person>
<input-address ng-model="data.thePerson.address"></input-address>
```

Il modello per questo esempio è volutamente semplicistico:

```
function Person(data) {
  data = data || {};
  this.name = data.name;
  this.address = data.address ? new Address(data.address) : null;
}

function Address(data) {
  data = data || {};
  this.street = data.street;
  this.number = data.number;
}
```

L'editor di indirizzi:

```
app.directive('inputAddress', function() {

    InputAddressController.$inject = ['$scope'];
    function InputAddressController($scope) {
        this.$scope = $scope;
        this._ngModel = null;
        this.value = null;
        this._unwatch = angular.noop;
    }
```

```
    InputAddressController.prototype.setNgModel = function(ngModel) {
        this._ngModel = ngModel;

        if( ngModel ) {
            // KEY POINT 3
            ngModel.$render = this._render.bind(this);
        }
    };

    InputAddressController.prototype._makeWatch = function() {
        // KEY POINT 1
        this._unwatch = this.$scope.$watchCollection(
            (function() {
                return this.value;
            }).bind(this),
            (function(newval, oldval) {
                if( newval !== oldval ) { // skip the initial trigger
                    this._ngModel.$setViewValue(newval !== null ? new Address(newval) : null);
                }
            }).bind(this)
        );
    };

    InputAddressController.prototype._render = function() {
        // KEY POINT 2
        this._unwatch();
        this.value = this._ngModel.$viewValue ? new Address(this._ngModel.$viewValue) : null;
        this._makeWatch();
    };

    return {
        restrict: 'E',
        scope: {},
        bindToController: true,
        controllerAs: 'ctrl',
        controller: InputAddressController,
        require: ['inputAddress', 'ngModel'],
        link: function(scope, elem, attrs, ctrls) {
            ctrls[0].setNgModel(ctrls[1]);
        },
        template:
            '<div>' +
                '<label><span>Street:</span><input type="text" ng-model="ctrl.value.street"
/></label>' +
                '<label><span>Number:</span><input type="text" ng-model="ctrl.value.number"
/></label>' +
            '</div>'
    };
});
```

Punti chiave:

1. Stiamo modificando un oggetto; non vogliamo cambiare direttamente l'oggetto che ci è stato dato dal nostro genitore (vogliamo che il nostro modello sia compatibile con il principio dell'immutabilità). Quindi creiamo una vigilanza superficiale sull'oggetto da modificare e aggiorniamo il modello con `$setViewValue()` ogni volta che una proprietà cambia. Passiamo una *copia* al nostro genitore.
2. Ogni volta che il modello cambia dall'esterno, lo copiamo e salviamo la copia nel nostro

ambito. Ancora principi di immutabilità, sebbene la copia interna non sia immutabile, l'esterno potrebbe benissimo essere. Inoltre ricostruiamo l'orologio (`this_unwatch();this._makeWatch();`), per evitare di far scattare l'osservatore per le modifiche che il modello ci ha spinto. (Vogliamo solo che l'orologio si attivi per le modifiche apportate all'interfaccia utente.)

3. A parte i punti precedenti, implementiamo `ngModel.$render()` e chiamiamo `ngModel.$setViewValue()` come faremmo per un controllo semplice (vedere l'esempio di valutazione).

Il codice per il controllo personalizzato della persona è quasi identico. Il modello sta usando `<input-address>` . In un'implementazione più avanzata potremmo estrarre i controller in un modulo riutilizzabile.

```
app.directive('inputPerson', function() {

    InputPersonController.$inject = ['$scope'];
    function InputPersonController($scope) {
        this.$scope = $scope;
        this._ngModel = null;
        this.value = null;
        this._unwatch = angular.noop;
    }

    InputPersonController.prototype.setNgModel = function(ngModel) {
        this._ngModel = ngModel;

        if( ngModel ) {
            ngModel.$render = this._render.bind(this);
        }
    };

    InputPersonController.prototype._makeWatch = function() {
        this._unwatch = this.$scope.$watchCollection(
            (function() {
                return this.value;
            }).bind(this),
            (function(newval, oldval) {
                if( newval !== oldval ) { // skip the initial trigger
                    this._ngModel.$setViewValue(newval !== null ? new Person(newval) : null);
                }
            }).bind(this)
        );
    };

    InputPersonController.prototype._render = function() {
        this._unwatch();
        this.value = this._ngModel.$viewValue ? new Person(this._ngModel.$viewValue) : null;
        this._makeWatch();
    };

    return {
        restrict: 'E',
        scope: {},
        bindToController: true,
        controllerAs: 'ctrl',
        controller: InputPersonController,
        require: ['inputPerson', 'ngModel'],
```

```
        link: function(scope, elem, attrs, ctrls) {
            ctrls[0].setNgModel(ctrls[1]);
        },
        template:
            '<div>' +
                '<label><span>Name:</span><input type="text" ng-model="ctrl.value.name"
/></label>' +
                '<input-address ng-model="ctrl.value.address"></input-address>' +
            '</div>'
    };
});
```

Nota: qui gli oggetti sono digitati, cioè hanno costruttori adeguati. Questo non è obbligatorio; il modello può essere semplice oggetti JSON. In questo caso basta usare `angular.copy()` posto dei costruttori. Un ulteriore vantaggio è che il controller diventa identico per i due controlli e può essere facilmente estratto in alcuni moduli comuni.

Il violino: https://jsfiddle.net/3tzyqfko/2/

Due versioni del violino hanno estratto il codice comune dei controller: https://jsfiddle.net/agj4cp0e/ e https://jsfiddle.net/ugb6Lw8b/

Leggi Direttive che utilizzano ngModelController online: https://riptutorial.com/it/angularjs/topic/2438/direttive-che-utilizzano-ngmodelcontroller

Capitolo 19: Direttive incorporate

Examples

Espressioni angolari - Testo contro numero

Questo esempio dimostra come vengono valutate le espressioni angolari quando si utilizza `type="text"` e `type="number"` per l'elemento di input. Si consideri il seguente controller e vista:

controllore

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

app.controller('ctrl', function($scope) {
    $scope.textInput = {
        value: '5'
    };
    $scope.numberInput = {
        value: 5
    };
});
```

vista

```
<div ng-app="app" ng-controller="ctrl">
    <input type="text" ng-model="textInput.value">
    {{ textInput.value + 5 }}
    <input type="number" ng-model="numberInput.value">
    {{ numberInput.value + 5 }}
</div>
```

- Quando si utilizza `+` in un'espressione associata all'input di *testo* , l'operatore **concatenerà** le stringhe (primo esempio), visualizzando 55 sullo schermo `*` .
- Quando si utilizza `+` in un'espressione associata all'input del *numero* , l'operatore restituisce la **somma** dei numeri (secondo esempio), visualizzando 10 sullo schermo `*` .

`*` - Fino a quando l'utente non modifica il valore nel campo di immissione, in seguito il display cambierà di conseguenza.

Esempio di lavoro

ngRepeat

`ng-repeat` è una direttiva incorporata in Angular che ti permette di iterare una matrice o un oggetto e ti dà la possibilità di ripetere un elemento una volta per ogni oggetto della collezione.

ng-ripeti un array

```
<ul>
```

```
    <li ng-repeat="item in itemCollection">
       {{item.Name}}
    </li>
</ul>
```

Dove:
item = singolo oggetto nella collezione
itemCollection = La matrice che stai iterando

ng-ripeti un oggetto

```
<ul>
    <li ng-repeat="(key, value) in myObject">
       {{key}} : {{value}}
    </li>
</ul>
```

Dove:
chiave = il nome della proprietà
valore = il valore della proprietà
myObject = l'oggetto che stai iterando

filtra la tua ng-ripetizione per input dell'utente

```
<input type="text" ng-model="searchText">
<ul>
    <li ng-repeat="string in stringArray | filter:searchText">
       {{string}}
    </li>
</ul>
```

Dove:
searchText = il testo che l'utente vuole filtrare per la lista
stringArray = una serie di stringhe, ad esempio `['string', 'array']`

Puoi anche visualizzare o riferire gli elementi filtrati altrove assegnando al filtro un alias di output con `as aliasName` , in questo modo:

```
<input type="text" ng-model="searchText">
<ul>
    <li ng-repeat="string in stringArray | filter:searchText as filteredStrings">
       {{string}}
    </li>
</ul>
<p>There are {{filteredStrings.length}} matching results</p>
```

ng-repeat-start e ng-repeat-end

Per ripetere più elementi DOM definendo un punto iniziale e uno finale è possibile utilizzare le direttive `ng-repeat-start` e `ng-repeat-end` .

```
<ul>
    <li ng-repeat-start="item in [{a: 1, b: 2}, {a: 3, b:4}]">
        {{item.a}}
    </li>
    <li ng-repeat-end>
        {{item.b}}
    </li>
</ul>
```

Produzione:

- 1
- 2
- 3
- 4

È importante chiudere sempre `ng-repeat-start` con `ng-repeat-end` .

variabili

`ng-repeat` espone anche queste variabili all'interno dell'espressione

Variabile	genere	Dettagli
`$index`	Numero	Uguale all'indice dell'attuale iterazione ($ index === 0 valuterà in vero al primo elemento iterato, vedi `$first`)
`$first`	booleano	Valuta true al primo elemento iterato
`$last`	booleano	Valuta true all'ultimo elemento iterato
`$middle`	booleano	Valuta true se l'elemento è compreso tra `$first` e `$last`
`$even`	booleano	Valuta true in un'iterazione pari (equivalente a `$index%2===0`)
`$odd`	booleano	Valuta true in un'iterazione dispari numerata (equivalente a `$index%2===1`)

Considerazioni sulle prestazioni

Il rendering di `ngRepeat` può rallentare, specialmente quando si utilizzano raccolte di grandi dimensioni.

Se gli oggetti nella raccolta hanno una proprietà identificatore, è sempre necessario `track by` dell'identificatore anziché dell'intero oggetto, che è la funzionalità predefinita. Se non è presente alcun identificativo, è sempre possibile utilizzare l' `$index` incorporato.

```
<div ng-repeat="item in itemCollection track by item.id">
<div ng-repeat="item in itemCollection track by $index">
```

Scopo di ngRepeat

`ngRepeat` creerà sempre un ambito figlio isolato, quindi è necessario prestare attenzione se è necessario accedere all'ambito principale all'interno della ripetizione.

Ecco un semplice esempio che mostra come è possibile impostare un valore nello scope genitore da un evento click all'interno di `ngRepeat` .

```
scope val:  {{val}}<br/>
ctrlAs val: {{ctrl.val}}
<ul>
    <li ng-repeat="item in itemCollection">
        <a href="#" ng-click="$parent.val=item.value; ctrl.val=item.value;">
            {{item.label}} {{item.value}}
        </a>
    </li>
</ul>

$scope.val = 0;
this.val = 0;

$scope.itemCollection = [{
    id: 0,
    value: 4.99,
    label: 'Football'
},
{
    id: 1,
    value: 6.99,
    label: 'Baseball'
},
{
    id: 2,
    value: 9.99,
    label: 'Basketball'
}];
```

Se era presente solo `val = item.value` in `ng-click` , non aggiornerà la `val` nell'ambito genitore a causa dell'ambito isolato. Ecco perché l'ambito genitore è accessibile con riferimento `$parent` o con la sintassi `controllerAs` (ad es. `ng-controller="mainController as ctrl"`).

Ng-ripetizione annidata

Puoi anche usare ng-repeat annidato.

```
<div ng-repeat="values in test">
    <div ng-repeat="i in values">
      [{{$parent.$index}},{{$index}}] {{i}}
    </div>
</div>

var app = angular.module("myApp", []);
app.controller("ctrl", function($scope) {
```

```
  $scope.test = [
    ['a', 'b', 'c'],
    ['d', 'e', 'f']
  ];
});
```

Qui per accedere all'indice di parent ng-repeat all'interno di child ng-repeat, puoi usare `$parent.$index` .

ngShow and ngHide

La direttiva `ng-show` mostra o nasconde l'elemento HTML in base a se l'espressione passata è vera o falsa. Se il valore dell'espressione è falsy allora si nasconderà. Se è vero, allora mostrerà.

La direttiva `ng-hide` è simile. Tuttavia, se il valore è falsy, mostrerà l'elemento HTML. Quando l'espressione è veritiera, la nasconderà.

Esempio di JSBin funzionante

Controller :

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

angular.module('app')
  .controller('ExampleController', ExampleController);

function ExampleController() {

  var vm = this;

  //Binding the username to HTML element
  vm.username = '';

  //A taken username
  vm.taken_username = 'StackOverflow';

}
```

vista

```
<section ng-controller="ExampleController as main">

    <p>Enter Password</p>
    <input ng-model="main.username" type="text">

    <hr>

    <!-- Will always show as long as StackOverflow is not typed in -->
    <!-- The expression is always true when it is not StackOverflow -->
    <div style="color:green;" ng-show="main.username != main.taken_username">
      Your username is free to use!
    </div>

    <!-- Will only show when StackOverflow is typed in -->
    <!-- The expression value becomes falsy -->
```

```
    <div style="color:red;" ng-hide="main.username != main.taken_username">
      Your username is taken!
    </div>

    <p>Enter 'StackOverflow' in username field to show ngHide directive.</p>

 </section>
```

ngOptions

`ngOptions` è una direttiva che semplifica la creazione di una casella a discesa html per la selezione di un elemento da una matrice che verrà memorizzata in un modello. L'attributo ngOptions viene utilizzato per generare dinamicamente un elenco di elementi `<option>` per l'elemento `<select>` utilizzando l'array o l'oggetto ottenuto valutando l'espressione di comprensione ngOptions.

Con `ng-options` il markup può essere ridotto a un solo tag select e la direttiva creerà la stessa selezione:

```
<select ng-model="selectedFruitNgOptions"
        ng-options="curFruit as curFruit.label for curFruit in fruit">
</select>
```

C'è un altro modo per creare opzioni di `select` usando `ng-repeat` , ma non è consigliabile usare `ng-repeat` dato che è usato principalmente per scopi generali come, per `forEach` solo loop. Mentre `ng-options` è specifico per la creazione di opzioni di tag `select` .

Sopra l'esempio usando `ng-repeat` sarebbe

```
<select ng-model="selectedFruit">
    <option ng-repeat="curFruit in fruit" value="{{curFruit}}">
        {{curFruit.label}}
    </option>
</select>
```

ESEMPIO COMPLETO

Vediamo l'esempio sopra in dettaglio anche con alcune variazioni in esso.

Modello dati per l'esempio:

```
$scope.fruit = [
    { label: "Apples", value: 4, id: 2 },
    { label: "Oranges", value: 2, id: 1 },
    { label: "Limes", value: 4, id: 4 },
    { label: "Lemons", value: 5, id: 3 }
];
```

```
<!-- label for value in array -->
<select ng-options="f.label for f in fruit" ng-model="selectedFruit"></select>
```

Tag di opzione generato alla selezione:

```
<option value="{ label: "Apples", value: 4, id: 2 }"> Apples </option>
```

effetti:

`f.label` sarà l'etichetta di `<option>` e il valore conterrà l'intero oggetto.

ESEMPIO COMPLETO

```
<!-- select as label for value in array -->
<select ng-options="f.value as f.label for f in fruit" ng-model="selectedFruit"></select>
```

Tag di opzione generato alla selezione:

```
<option value="4"> Apples </option>
```

effetti:

`f.value` (4) sarà il valore in questo caso mentre l'etichetta è sempre la stessa.

ESEMPIO COMPLETO

```
<!-- label group by group for value in array -->
<select ng-options="f.label group by f.value for f in fruit" ng-
model="selectedFruit"></select>
```

Tag di opzione generato alla selezione:

```
<option value="{ label: "Apples", value: 4, id: 2 }"> Apples </option>
```

effetti:

Le opzioni saranno raggruppate in base al loro `value` . Le opzioni con lo stesso `value` rientrano in una categoria

ESEMPIO COMPLETO

```
<!-- label disable when disable for value in array -->
<select ng-options="f.label disable when f.value == 4 for f in fruit" ng-
model="selectedFruit"></select>
```

Tag di opzione generato alla selezione:

```
<option disabled="" value="{ label: "Apples", value: 4, id: 2 }"> Apples </option>
```

effetti:

"Mele" e "Limes" saranno disabilitati (impossibile selezionare) a causa della condizione `disable`

`when f.value==4` . Tutte le opzioni con `value=4` devono essere disabilitate

ESEMPIO COMPLETO

```
<!-- label group by group for value in array track by trackexpr -->
<select ng-options="f.value as f.label group by f.value for f in fruit track by f.id" ng-
model="selectedFruit"></select>
```

Tag di opzione generato alla selezione:

```
<option value="4"> Apples </option>
```

effetti:

Non v'è cambiamento visibile quando si usa `trackBy` , ma angolare rileverà cambiamenti nel `id` anziché di riferimento che è quasi sempre una soluzione migliore.

ESEMPIO COMPLETO

```
<!-- label for value in array | orderBy:orderexpr track by trackexpr -->
<select ng-options="f.label for f in fruit | orderBy:'id' track by f.id" ng-
model="selectedFruit"></select>
```

Tag di opzione generato alla selezione:

```
<option disabled="" value="{ label: "Apples", value: 4, id: 2 }"> Apples </option>
```

effetti:

`orderBy` è un filtro standard AngularJS che organizza le opzioni in ordine crescente (per impostazione predefinita), quindi "Arance" in questo apparirà 1 dal suo `id` = 1.

ESEMPIO COMPLETO

Tutti i `<select>` con `ng-options` devono avere `ng-model` allegato.

ngModel

Con ng-model puoi associare una variabile a qualsiasi tipo di campo di input. È possibile visualizzare la variabile usando doppie parentesi graffe, ad esempio `{{myAge}}` .

```
<input type="text" ng-model="myName">
<p>{{myName}}</p>
```

Mentre digiti nel campo di input o lo cambi in qualunque modo vedrai istantaneamente il valore nel paragrafo update.

La variabile ng-model, in questo caso, sarà disponibile nel controller come `$scope.myName` . Se si utilizza la sintassi del `controllerAs` :

```
<div ng-controller="myCtrl as mc">
    <input type="text" ng-model="mc.myName">
    <p>{{mc.myName}}</p>
</div>
```

Sarà necessario fare riferimento allo scope del controller pre-in sospeso l'alias del controller definito nell'attributo ng-controller nella variabile ng-model. In questo modo non avrai bisogno di iniettare `$scope` nel tuo controller per fare riferimento alla tua variabile ng-model, la variabile sarà disponibile come `this.myName` all'interno della funzione del tuo controller.

ngClass

Supponiamo di dover mostrare lo stato di un utente e di avere diverse classi CSS che potrebbero essere utilizzate. Angular rende molto facile scegliere da un elenco di diverse classi possibili che consentono di specificare un elenco di oggetti che include condizionali. Angular è in grado di utilizzare la classe corretta in base alla verità dei condizionali.

Il tuo oggetto dovrebbe contenere coppie chiave / valore. La chiave è un nome di classe che verrà applicato quando il valore (condizionale) viene valutato su true.

```
<style>
    .active { background-color: green; color: white; }
    .inactive { background-color: gray; color: white; }
    .adminUser { font-weight: bold; color: yellow; }
    .regularUser { color: white; }
</style>

<span ng-class="{
    active: user.active,
    inactive: !user.active,
    adminUser: user.level === 1,
    regularUser: user.level === 2
}">John Smith</span>
```

Angular controllerà l'oggetto `$scope.user` per vedere lo stato `active` e il numero del `level` . A seconda dei valori in queste variabili, Angular applicherà lo stile di corrispondenza allo `<span>` .

ngIf

`ng-if` è una direttiva simile a **`ng-show`** ma inserisce o rimuove l'elemento dal DOM invece di semplicemente nasconderlo. Angular 1.1.5 ha introdotto la direttiva ng-If. È possibile utilizzare la direttiva ng-if sopra le versioni 1.1.5. Ciò è utile perché Angular non elaborerà i digest per gli elementi all'interno di una `ng-if` rimossa `ng-if` riducendo il carico di lavoro di Angular in particolare per i binding di dati complessi.

A differenza di `ng-show` , la direttiva `ng-if` crea un ambito figlio che utilizza l'ereditarietà prototipale. Ciò significa che l'impostazione di un valore primitivo sull'ambito figlio non si applica al genitore. Per impostare una primitiva sull'ambito `$parent` proprietà `$parent` sull'ambito figlio.

JavaScript

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

angular.module('MyApp').controller('myController', ['$scope', '$window', function
myController($scope, $window) {
    $scope.currentUser= $window.localStorage.getItem('userName');
}]);
```

vista

```
<div ng-controller="myController">
    <div ng-if="currentUser">
        Hello, {{currentUser}}
    </div>
    <div ng-if="!currentUser">
        <a href="/login">Log In</a>
        <a href="/register">Register</a>
    </div>
</div>
```

DOM Se `currentUser` non è indefinito

```
<div ng-controller="myController">
    <div ng-if="currentUser">
        Hello, {{currentUser}}
    </div>
    <!-- ng-if: !currentUser -->
</div>
```

DOM Se `currentUser` non è definito

```
<div ng-controller="myController">
    <!-- ng-if: currentUser -->
    <div ng-if="!currentUser">
        <a href="/login">Log In</a>
        <a href="/register">Register</a>
    </div>
</div>
```

Esempio di lavoro

Promessa della funzione

La direttiva ngIf accetta anche le funzioni, che richiedono logicamente di restituire true o false.

```
<div ng-if="myFunction()">
    <span>Span text</span>
</div>
```

Il testo di span verrà visualizzato solo se la funzione restituisce true.

```
$scope.myFunction = function() {
    var result = false;
    // Code to determine the boolean value of result
    return result;
};
```

Come qualsiasi espressione angolare, la funzione accetta qualsiasi tipo di variabile.

ngMouseenter e ngMouseleave

Le direttive `ng-mouseenter` e `ng-mouseleave` sono utili per eseguire gli eventi e applicare lo stile CSS quando si posiziona il cursore all'interno o all'esterno degli elementi DOM.

La direttiva `ng-mouseenter` esegue un'espressione quando un mouse entra in un evento (quando l'utente inserisce il puntatore del mouse sull'elemento DOM in cui risiede questa direttiva)

HTML

```
<div ng-mouseenter="applyStyle = true" ng-class="{'active': applyStyle}">
```

Nell'esempio sopra, quando l'utente punta il mouse sul `div` , `applyStyle` diventa `true` , che a sua volta applica la classe CSS `.active` alla classe `ng-class` .

La direttiva `ng-mouseleave` esegue un'espressione uno un evento di uscita del mouse (quando l'utente `ng-mouseleave` il cursore del mouse dall'elemento DOM in cui risiede questa direttiva)

HTML

```
<div ng-mouseenter="applyStyle = true" ng-mouseleaver="applyStyle = false" ng-
class="{'active': applyStyle}">
```

Riutilizzando il primo esempio, ora quando l'utente si allontana dal puntatore del mouse dal div, la classe `.active` viene rimossa.

ngDisabled

Questa direttiva è utile per limitare gli eventi di input in base a determinate condizioni esistenti.

La direttiva `ng-disabled` accetta ed espressione che dovrebbe valutare valori veri o falsi.

`ng-disabled` è usato per applicare in modo condizionale l'attributo `disabled` su un elemento di `input` .

HTML

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

<button ng-disabled="vm.name.length===0" ng-click="vm.submitMe">Submit</button>
```

`vm.name.length===0` viene valutato come true se la lunghezza `input` è 0, che è disabilita il pulsante, impedendo all'utente di attivare l'evento click di `ng-click`

ngDblclick

La direttiva `ng-dblclick` è utile quando si desidera associare un evento di doppio clic agli elementi DOM.

Questa direttiva accetta un'espressione

HTML

```
<input type="number" ng-model="num = num + 1" ng-init="num=0">

<button ng-dblclick="num++">Double click me</button>
```

Nell'esempio sopra, il valore mantenuto `input` verrà incrementato quando si fa doppio clic sul pulsante.

Cheat Sheet delle direttive integrate

`ng-app` Imposta la sezione AngularJS.

`ng-init` Imposta un valore di variabile predefinito.

`ng-bind` Alternativa al modello {{}}.

`ng-bind-template` Lega più espressioni alla vista.

`ng-non-bindable` che i dati non sono associabili.

`ng-bind-html` Lega la proprietà HTML interna di un elemento HTML.

`ng-change` Valuta l'espressione specificata quando l'utente cambia l'input.

`ng-checked` Imposta la casella di controllo.

`ng-class` Imposta dinamicamente la classe css.

`ng-cloak` Impedisce la visualizzazione del contenuto finché AngularJS non ha preso il controllo.

`ng-click` Esegue un metodo o un'espressione quando si fa clic sull'elemento.

`ng-controller` Collega una classe controller alla vista.

`ng-disabled` Controlla la proprietà disabilitata dell'elemento del modulo

`ng-form` Imposta un modulo

`ng-href` Associa dinamicamente le variabili AngularJS all'attributo href.

`ng-include` Utilizzato per recuperare, compilare e includere un frammento HTML esterno nella tua pagina.

`ng-if` Rimuove o ricrea un elemento nel DOM a seconda di un'espressione

`ng-switch` Controlla in modo condizionale il controllo in base all'espressione corrispondente.

`ng-model` Lega un input, seleziona, textarea ecc. elementi con proprietà model.

`ng-readonly` Utilizzato per impostare un attributo readonly su un elemento.

`ng-repeat` Utilizzato per scorrere ogni elemento di una raccolta per creare un nuovo modello.

`ng-selected` Utilizzato per impostare l'opzione selezionata nell'elemento.

`ng-show/ng-hide` Mostra / nascondi gli elementi in base a un'espressione.

`ng-src` Associa dinamicamente le variabili AngularJS all'attributo src.

`ng-submit` Associa le espressioni angolari agli eventi onsubmit.

`ng-value` Associa le espressioni angolari al valore di.

`ng-required` Associa le espressioni angolari agli eventi onsubmit.

`ng-style` Imposta lo `ng-style` CSS su un elemento HTML.

`ng-pattern` Aggiunge il modello validatore a ngModel.

`ng-maxlength` Aggiunge il validatore maxlength a ngModel.

`ng-minlength` Aggiunge il validatore minlength a ngModel.

`ng-classeven` Funziona in combinazione con ngRepeat e ha effetto solo su righe dispari (pari).

`ng-classodd` Funziona in combinazione con ngRepeat e ha effetto solo su righe dispari (pari).

`ng-cut` Utilizzato per specificare il comportamento personalizzato sull'evento di taglio.

`ng-copy` Usato per specificare il comportamento personalizzato sull'evento di copia.

`ng-paste` Utilizzato per specificare il comportamento personalizzato sull'evento incolla.

`ng-options` Utilizzato per generare dinamicamente un elenco di elementi per l'elemento.

`ng-list` Utilizzato per convertire una stringa in lista in base al delimitatore specificato.

`ng-open` Usato per impostare l'attributo open sull'elemento, se l'espressione dentro ngOpen è true.

Fonte (modificato un po ')

ngClick

La direttiva `ng-click` associa un evento click a un elemento DOM.

La direttiva `ng-click` consente di specificare il comportamento personalizzato quando si fa clic su un elemento di DOM.

È utile quando si desidera allegare eventi di clic sui pulsanti e gestirli sul controller.

Questa direttiva accetta un'espressione con l'oggetto events disponibile come `$event`

HTML

```
<input ng-click="onClick($event)">Click me</input>
```

controllore

```
.controller("ctrl", function($scope) {
    $scope.onClick = function(evt) {
        console.debug("Hello click event: %o ",evt);
    }
})
```

HTML

```
<button ng-click="count = count + 1" ng-init="count=0">
  Increment
</button>
<span>
  count: {{count}}
</span>
```

HTML

```
<button ng-click="count()" ng-init="count=0">
  Increment
</button>
<span>
  count: {{count}}
</span>
```

controllore

```
...

$scope.count = function(){
    $scope.count = $scope.count + 1;
}
...
```

Quando si fa clic sul pulsante, un'invocazione della funzione `onClick` stamperà "Hello click event" seguito dall'oggetto evento.

ngRequired

`ng-required` aggiunge o rimuove l'attributo di convalida `required` su un elemento, che a sua volta abiliterà e disabiliterà la chiave di convalida `require` per l' `input` .

È usato per definire opzionalmente se è richiesto un elemento di `input` per avere un valore non vuoto. La direttiva è utile quando si progetta la convalida su moduli HTML complessi.

HTML

```
<input type="checkbox" ng-model="someBooleanValue">
<input type="text" ng-model="username" ng-required="someBooleanValue">
```

NG-modello-options

`ng-model-options` consente di modificare il comportamento predefinito di `ng-model` , questa direttiva consente di registrare eventi che si attivano quando il modello ng viene aggiornato e di applicare un effetto antirimbalzo.

Questa direttiva accetta un'espressione che valuterà un oggetto definizione o un riferimento a un valore di ambito.

Esempio:

```
<input type="text" ng-model="myValue" ng-model-options="{'debounce': 500}">
```

L'esempio precedente allegherà un effetto antirimbalzo di 500 millisecondi su `myValue` , che farà aggiornare il modello a 500 ms dopo che l'utente ha finito di digitare `input` (ovvero, quando `myValue` terminato l'aggiornamento).

Proprietà dell'oggetto disponibili

1. `updateOn` : specifica quale evento deve essere associato all'input

   ```
   ng-model-options="{ updateOn: 'blur'}" // will update on blur
   ```

2. `debounce` : specifica un ritardo di alcuni millisecondi verso l'aggiornamento del modello

   ```
   ng-model-options="{'debounce': 500}" // will update the model after 1/2 second
   ```

3. `allowInvalid` : un flag booleano che consente un valore non valido per il modello, aggirando la convalida del modulo predefinito, per impostazione predefinita questi valori verranno considerati non `undefined` .

4. `getterSetter` : un flag booleano che indica se trattare il `ng-model` come una funzione getter /

setter invece di un valore di modello normale. La funzione verrà quindi eseguita e restituirà il valore del modello.

Esempio:

```
<input type="text" ng-model="myFunc" ng-model-options="{'getterSetter': true}">

$scope.myFunc = function() {return "value";}
```

5. `timezone` : definisce il fuso orario per il modello se l'input è della `date` o `time` . tipi

ngCloak

La direttiva `ngCloak` viene utilizzata per impedire che il modello html Angolare venga visualizzato brevemente dal browser nel suo formato raw (non compilato) durante il caricamento dell'applicazione. - Visualizza sorgente

HTML

```
<div ng-cloak>
  <h1>Hello {{ name }}</h1>
</div>
```

`ngCloak` può essere applicato all'elemento body, ma l'uso preferito è applicare più direttive ngCloak a piccole porzioni della pagina per consentire il rendering progressivo della vista del browser.

La direttiva `ngCloak` non ha parametri.

Vedi anche: Prevenire lo sfarfallio

ngInclude

ng-include consente di delegare il controllo di una parte della pagina a un controller specifico. Potresti voler farlo perché la complessità di quel componente sta diventando tale da incapsulare tutta la logica in un controller dedicato.

Un esempio è:

```
<div ng-include
     src="'/gridview'"
     ng-controller='gridController as gc'>
</div>
```

Si noti che `/gridview` dovrà essere servito dal server Web come URL distinto e legittimo.

Inoltre, si noti che `src` -attribute accetta un'espressione Angolare. Potrebbe trattarsi di una variabile o di una chiamata di funzione, ad esempio o, come in questo esempio, di una costante di stringa. In questo caso è necessario assicurarsi di **avvolgere l'URL di origine tra virgolette singole** , quindi verrà valutato come costante di stringa. Questa è una fonte comune di confusione.

All'interno di `/gridview` html, puoi fare riferimento a `gridController` come se fosse avvolto attorno alla pagina, ad esempio:

```
<div class="row">
  <button type="button" class="btn btn-default" ng-click="gc.doSomething()"></button>
</div>
```

ngSrc

L'utilizzo del markup angolare come `{{hash}}` in un attributo src non funziona correttamente. Il browser recupera dall'URL con il testo letterale `{{hash}}` finché Angular sostituisce l'espressione all'interno di `{{hash}}` . `ng-src` direttiva `ng-src` sovrascrive l'attributo `src` originale per l'elemento tag immagine e risolve il problema

```
<div ng-init="pic = 'pic_angular.jpg'">
    <h1>Angular</h1>
    <img ng-src="{{pic}}">
</div>
```

ngPattern

La direttiva `ng-pattern` accetta un'espressione che valuta un modello di espressione regolare e utilizza tale modello per convalidare un input testuale.

Esempio:

Diciamo che vogliamo che un elemento `<input>` diventi valido quando è value (ng-model) è un indirizzo IP valido.

Modello:

```
<input type="text" ng-model="ipAddr" ng-pattern="ipRegex" name="ip" required>
```

controller:

```
$scope.ipRegex = /\b(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(?:25[0-5]|2[0-4][0-
9]|[01]?[0-9][0-9]?)\b/;
```

ngValue

Utilizzato principalmente in `ng-repeat` ngValue è utile quando si generano dinamicamente elenchi di pulsanti di opzione usando ngRepeat

```
<script>
   angular.module('valueExample', [])
     .controller('ExampleController', ['$scope', function($scope) {
       $scope.names = ['pizza', 'unicorns', 'robots'];
       $scope.my = { favorite: 'unicorns' };
     }]);
```

```
</script>
 <form ng-controller="ExampleController">
   <h2>Which is your favorite?</h2>
     <label ng-repeat="name in names" for="{{name}}">
       {{name}}
       <input type="radio"
              ng-model="my.favorite"
              ng-value="name"
              id="{{name}}"
              name="favorite">
     </label>
   <div>You chose {{my.favorite}}</div>
 </form>
```

Lavoro plnkr

ngCopy

La direttiva `ngCopy` specifica il comportamento da eseguire su un evento di copia.

Impedire a un utente di copiare i dati

```
<p ng-copy="blockCopy($event)">This paragraph cannot be copied</p>
```

Nel controller

```
$scope.blockCopy = function(event) {
    event.preventDefault();
    console.log("Copying won't work");
}
```

ngPaste

La direttiva `ngPaste` specifica il comportamento personalizzato da eseguire quando un utente incolla il contenuto

```
<input ng-paste="paste=true" ng-init="paste=false" placeholder='paste here'>
pasted: {{paste}}
```

ngHref

ngHref è usato al posto dell'attributo href, se abbiamo espressioni angolari all'interno del valore href. La direttiva ngHref sovrascrive l'attributo href originale di un tag html utilizzando l'attributo href come tag, tag, ecc.

La direttiva ngHref assicura che il collegamento non sia interrotto anche se l'utente fa clic sul collegamento prima che AngularJS abbia valutato il codice.

Esempio 1

```
<div ng-init="linkValue = 'http://stackoverflow.com'">
    <p>Go to <a ng-href="{{linkValue}}">{{linkValue}}</a>!</p>
</div>
```

Esempio 2 Questo esempio ottiene dinamicamente il valore href dalla casella di input e lo carica come valore href.

```
<input ng-model="value" />
<a id="link" ng-href="{{value}}">link</a>
```

Esempio 3

```
<script>
angular.module('angularDoc', [])
.controller('myController', function($scope) {
  // Set some scope value.
  // Here we set bootstrap version.
  $scope.bootstrap_version = '3.3.7';

  // Set the default layout value
  $scope.layout = 'normal';
});
</script>
<!-- Insert it into Angular Code -->
<link rel="stylesheet" ng-href="//maxcdn.bootstrapcdn.com/bootstrap/{{ bootstrap_version
}}/css/bootstrap.min.css">
<link rel="stylesheet" ng-href="layout-{{ layout }}.css">
```

ngList

La direttiva `ng-list` viene utilizzata per convertire una stringa delimitata da un input di testo a una matrice di stringhe o viceversa.

La direttiva `ng-list` usa un delimitatore predefinito di `", "` (spazio virgola).

È possibile impostare manualmente il delimitatore assegnando a `ng-list` un delimitatore come questo `ng-list="; "` .

In questo caso il delimitatore è impostato su un punto e virgola seguito da uno spazio.

Di default `ng-list` ha un attributo `ng-trim` che è impostato su true. `ng-trim` quando false, rispetterà lo spazio bianco nel delimitatore. Per impostazione predefinita, `ng-list` non prende in considerazione lo spazio bianco a meno che non si imposti `ng-trim="false"` .

Esempio:

```
angular.module('test', [])
  .controller('ngListExample', ['$scope', function($scope) {
    $scope.list = ['angular', 'is', 'cool!'];
}]);
```

Un delimitatore del cliente è impostato per essere `;` . E il modello della casella di input è impostato

sull'array che è stato creato sull'oscilloscopio.

```
<body ng-app="test" ng-controller="ngListExample">
  <input ng-model="list" ng-list="; " ng-trim="false">
</body>
```

La casella di input verrà visualizzata con il contenuto: `angular; is; cool!`

Leggi Direttive incorporate online: https://riptutorial.com/it/angularjs/topic/706/direttive-incorporate

Capitolo 20: Direttive personalizzate

introduzione

Qui apprenderai la funzione Directives di AngularJS. Di seguito troverai informazioni su quali sono le Direttive, oltre a esempi di base e avanzati su come usarle.

Parametri

Parametro	Dettagli
scopo	Proprietà per impostare l'ambito della direttiva. Può essere impostato come falso, vero o come ambito isolato: {@, =, <, &}.
campo di applicazione: falsy	La direttiva usa l'ambito genitore. Nessun ambito creato per direttiva.
ambito: vero	La direttiva eredita l'ambito genitore prototipicamente come un nuovo ambito figlio. Se ci sono più direttive sullo stesso elemento che richiede un nuovo ambito, condivideranno un nuovo ambito.
scopo: { @ }	Associazione unidirezionale di una proprietà dell'ambito direttiva a un valore di attributo DOM. Poiché il valore dell'attributo è associato al genitore, cambierà nell'ambito della direttiva.
ambito: {=}	Associazione dell'attributo bidirezionale che modifica l'attributo nel genitore se l'attributo direttiva cambia e viceversa.
ambito: {<}	Associazione unidirezionale di una proprietà dell'ambito direttiva e un'espressione dell'attributo DOM. L'espressione viene valutata nel genitore. Questo controlla l'identità del valore padre in modo che le modifiche a una proprietà dell'oggetto nel padre non vengano riflesse nella direttiva. Le modifiche a una proprietà dell'oggetto in una direttiva si rifletteranno in parent, poiché entrambi fanno riferimento allo stesso oggetto
scopo: { & }	Consente alla direttiva di passare i dati a un'espressione da valutare nel genitore.
compilare: funzione	Questa funzione viene utilizzata per eseguire la trasformazione DOM sul modello di direttiva prima dell'esecuzione della funzione di collegamento. Accetta `tElement` (il modello di direttiva) e `tAttr` (elenco di attributi dichiarati sulla direttiva). Non ha accesso all'ambito. Può restituire una funzione che verrà registrata come funzione `post-link` oppure restituire un oggetto con proprietà `pre` e `post` con le quali saranno registrate come

Parametro	Dettagli
	funzioni `pre-link` e `post-link` .
link: funzione / oggetto	La proprietà del collegamento può essere configurata come funzione o oggetto. Può ricevere i seguenti argomenti: ambito (ambito direttiva), iElement (elemento DOM in cui viene applicata la direttiva), iAttrs (raccolta di attributi dell'elemento DOM), controller (matrice di controller richiesta dalla direttiva), transcludeFn. È utilizzato principalmente per impostare i listener DOM, osservare le proprietà del modello per le modifiche e aggiornare il DOM. Esegue dopo la clonazione del modello. È configurato indipendentemente se non esiste una funzione di compilazione.
funzione pre-link	Funzione di collegamento che viene eseguita prima delle funzioni di collegamento figlio. Per impostazione predefinita, le funzioni di collegamento della direttiva figlio vengono eseguite prima delle funzioni di collegamento della direttiva padre e la funzione di pre-collegamento consente al genitore di collegarsi per primo. Un caso d'uso è se il bambino richiede dati dal genitore.
funzione post-link	Funzione di collegamento che i dirigenti dopo gli elementi figlio sono collegati al genitore. Viene comunemente utilizzato per il collegamento di gestori di eventi e l'accesso alle direttive secondarie, ma i dati richiesti dalla direttiva figlio non dovrebbero essere impostati qui perché la direttiva figlio sarà già stata collegata.
limitare: stringa	Definisce come chiamare la direttiva dall'interno del DOM. Possibili valori (supponendo che il nome della nostra direttiva sia `demoDirective`): `E` - Nome dell'elemento (`<demo-directive></demo-directive>`), `A` - Attributo (`<div demo-directive></div>`), `C` - Corrispondenza (`<div class="demo-directive"></div>`), `M` - Per commento (`<!-- directive: demo-directive -->`). La proprietà `restrict` può anche supportare più opzioni, ad esempio - `restrict: "AC"` limiterà la direttiva a *Attribute* OR *Class* . Se omesso, il valore **predefinito** è `"EA"` (Elemento o Attributo).
richiede: 'demoDirective'	Individuare il controller di demoDirective sull'elemento corrente e iniettarne il controller come quarto argomento della funzione di collegamento. Fai un errore se non lo trovi.
richiede: "? demoDirective"	Tentare di individuare il controller di demoDirective o passare null al collegamento fn se non trovato.
richiede: '^ demoDirective'	Individua il controller di demoDirective cercando l'elemento e i suoi genitori. Fai un errore se non lo trovi.
richiede: '^^ demoDirective'	Individua il controller di demoDirective cercando i genitori dell'elemento. Fai un errore se non lo trovi.
richiede: '? ^	Cerca di localizzare il controller di demoDirective cercando l'elemento ei

Parametro	Dettagli
demoDirective'	suoi genitori o passa null al link fn se non trovato.
richiede: '? ^^ demoDirective'	Cerca di localizzare il controller di demoDirective cercando i genitori dell'elemento o passa null al link fn se non lo trovi.

Examples

Creare e consumare direttive personalizzate

Le direttive sono una delle funzionalità più potenti di angularjs. Le direttive angularjs personalizzate sono utilizzate per estendere la funzionalità di html creando nuovi elementi html o attributi personalizzati per fornire un determinato comportamento a un tag html.

directive.js

```
// Create the App module if you haven't created it yet
var demoApp= angular.module("demoApp", []);

// If you already have the app module created, comment the above line and create a reference
of the app module
var demoApp = angular.module("demoApp");


// Create a directive using the below syntax
// Directives are used to extend the capabilities of html element
// You can either create it as an Element/Attribute/class
// We are creating a directive named demoDirective. Notice it is in CamelCase when we are
defining the directive just like ngModel
// This directive will be activated as soon as any this element is encountered in html

demoApp.directive('demoDirective', function () {

  // This returns a directive definition object
  // A directive definition object is a simple JavaScript object used for configuring the
directive’s behaviour,template..etc
  return {
    // restrict: 'AE', signifies that directive is Element/Attribute directive,
    // "E" is for element, "A" is for attribute, "C" is for class, and "M" is for comment.
    // Attributes are going to be the main ones as far as adding behaviors that get used the
most.
    // If you don't specify the restrict property it will default to "A"
    restrict :'AE',

    // The values of scope property decides how the actual scope is created and used inside a
directive. These values can be either “false”, “true” or “{}”. This creates an isolate scope
for the directive.
    // '@' binding is for passing strings. These strings support {{}} expressions for
interpolated values.
    // '=' binding is for two-way model binding. The model in parent scope is linked to the
model in the directive's isolated scope.
    // '&' binding is for passing a method into your directive's scope so that it can be
called within your directive.
    // The method is pre-bound to the directive's parent scope, and supports arguments.
    scope: {
```

```
      name: "@",  // Always use small casing here even if it's a mix of 2-3 words
    },

    // template replaces the complete element with its text.
    template: "<div>Hello {{name}}!</div>",

    // compile is called during application initialization. AngularJS calls it once when html
page is loaded.
    compile: function(element, attributes) {
      element.css("border", "1px solid #cccccc");

      // linkFunction is linked with each element with scope to get the element specific data.
      var linkFunction = function($scope, element, attributes) {
        element.html("Name: <b>"+$scope.name +"</b>");
        element.css("background-color", "#ff00ff");
      };
      return linkFunction;
    }
  };
});
```

Questa direttiva può quindi essere utilizzata in App come:

```
<html>

   <head>
      <title>Angular JS Directives</title>
   </head>
   <body>
   <script src =
"http://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>
   <script src="directive.js"></script>
    <div ng-app = "demoApp">
       <!-- Notice we are using Spinal Casing here -->
       <demo-directive name="World"></demo-directive>

      </div>
   </body>
</html>
```

Modello oggetto definizione direttiva

```
demoApp.directive('demoDirective', function () {
  var directiveDefinitionObject = {
    multiElement:
    priority:
    terminal:
    scope: {},
    bindToController: {},
    controller:
    controllerAs:
    require:
    restrict:
    templateNamespace:
    template:
    templateUrl:
    transclude:
    compile:
```

```
    link: function(){}
  };
  return directiveDefinitionObject;
});
```

1. **`multiElement`** - impostato su true e tutti i nodi DOM tra l'inizio e la fine del nome della direttiva verranno raccolti e raggruppati come elementi direttivi
2. **`priority`** : consente di specificare l'ordine per applicare le direttive quando vengono definite più direttive su un singolo elemento DOM. Le direttive con numeri più alti vengono compilate per prime.
3. **`terminal`** - impostato su true e la priorità corrente sarà l'ultima serie di direttive da eseguire
4. **`scope`** : definisce il campo di applicazione della direttiva
5. **`bind to controller`** - associa le proprietà dell'ambito direttamente al controller di istruzioni
6. **`controller`** - funzione di costruzione del controller
7. **`require`** - richiede un'altra direttiva e inietta il suo controller come quarto argomento della funzione di collegamento
8. **`controllerAs`** - nome riferimento al controller nell'ambito della direttiva per consentire al controller di fare riferimento al modello di direttiva.
9. **`restrict`** - limita la direttiva a Element, Attribute, Class o Comment
10. **`templateNameSpace`** - imposta il tipo di documento utilizzato dal modello di direttiva: html, svg o math. html è l'impostazione predefinita
11. **`template`** - markup html che per impostazione predefinita sostituisce il contenuto dell'elemento della direttiva o avvolge il contenuto dell'elemento direttiva se transclude è true
12. **`templateUrl`** - url fornito in modo asincrono per il modello
13. **`transclude`** - Estrae il contenuto dell'elemento in cui appare la direttiva e lo mette a disposizione della direttiva. I contenuti sono compilati e forniti alla direttiva come funzione di inclusione.
14. **`compile`** - funzione per trasformare il DOM del template
15. **`link`** - usato solo se la proprietà compilata non è definita. La funzione di collegamento è responsabile della registrazione degli ascoltatori DOM e dell'aggiornamento del DOM. Viene eseguito dopo che il modello è stato clonato.

Esempio di direttiva di base

superman-directive.js

```
angular.module('myApp', [])
  .directive('superman', function() {
    return {
      // restricts how the directive can be used
      restrict: 'E',
      templateUrl: 'superman-template.html',
      controller: function() {
        this.message = "I'm superman!"
      },
      controllerAs: 'supermanCtrl',
      // Executed after Angular's initialization. Use commonly
      // for adding event handlers and DOM manipulation
      link: function(scope, element, attributes) {
        element.on('click', function() {
```

```
          alert('I am superman!')
        });
      }
    }
  });
```

superman-template.html

```
<h2>{{supermanCtrl.message}}</h2>
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.5.0/angular.js"></script>
  <script src="superman-directive.js"><script/>
</head>
<body>
<div ng-app="myApp">
  <superman></superman>
</div>
</body>
</html>
```

Puoi controllare di più sulle funzioni di `restrict` e `link` della direttiva sulla documentazione ufficiale di AngularJS sulle Direttive

Come creare un componente resuable usando la direttiva

Le direttive AngularJS sono ciò che controlla il rendering dell'HTML all'interno di un'applicazione AngularJS. Possono essere un elemento HTML, un attributo, una classe o un commento. Le direttive sono utilizzate per manipolare il DOM, associare un nuovo comportamento agli elementi HTML, l'associazione dei dati e molti altri. Alcuni esempi di direttive che fornisce angolare sono ng-model, ng-hide, ng-if.

Allo stesso modo si può creare la propria direttiva personalizzata e renderli resuable. Per creare direttive personalizzate Riferimento . L'idea che sta alla base della creazione di direttive riutilizzabili è quella di creare un insieme di direttive / componenti scritte da te proprio come angularjs ci fornisce usando angular.js. Queste direttive riutilizzabili possono essere particolarmente utili quando si dispone di una suite di applicazioni / applicazioni che richiede un comportamento, un aspetto e un comportamento coerenti. Un esempio di tale componente riutilizzabile può essere una semplice barra degli strumenti che potresti voler utilizzare nella tua applicazione o in diverse applicazioni, ma vuoi che si comportino allo stesso modo o abbiano lo stesso aspetto.

In primo luogo, crea una cartella denominata resuableComponents nella cartella dell'app e crea riusableModuleApp.js

reusableModuleApp.js:

```
(function(){

 var reusableModuleApp = angular.module('resuableModuleApp', ['ngSanitize']);

 //Remember whatever dependencies you have in here should be injected in the app module where
it is intended to be used or it's scripts should be included in your main app
                              //We will be injecting ng-sanitize

 resubaleModuleApp.directive('toolbar', toolbar)

 toolbar.$inject=['$sce'];

 function toolbar($sce){

     return{
     restrict :'AE',
        //Defining below isolate scope actually provides window for the directive to take data
from app that will be using this.
        scope : {
                value1: '=',
                value2: '=',
                },

         }
     template : '<ul>  <li><a ng-click="Add()" href="">{{value1}}</a></li>  <li><a ng-
click="Edit()" href="#">{{value2}}</a></li> </ul> ',
         link : function(scope, element, attrs){

               //Handle's Add function
               scope.Add = function(){


               };

               //Handle's Edit function
               scope.Edit = function(){

               };
     }
  }

});
```

mainApp.js:

```
(function(){
   var mainApp = angular.module('mainApp', ['reusableModuleApp']); //Inject resuableModuleApp
in your application where you want to use toolbar component

   mainApp.controller('mainAppController', function($scope){
      $scope.value1 = "Add";
      $scope.value2 = "Edit";

   });

 });
```

index.html:

```
<!doctype html>
<html ng-app="mainApp">
<head>
 <title> Demo Making a reusable component
<head>
 <body ng-controller="mainAppController">

    <!-- We are providing data to toolbar directive using mainApp'controller -->
    <toolbar value1="value1" value2="value2"></toolbar>

   <!-- We need to add the dependent js files on both apps here -->
    <script src="js/angular.js"></script>
    <script src="js/angular-sanitize.js"></script>

    <!-- your mainApp.js should be added afterwards --->
     <script src="mainApp.js"></script>

    <!-- Add your reusable component js files here -->
     <script src="resuableComponents/reusableModuleApp.js"></script>

 </body>
</html>
```

Le direttive sono componenti riutilizzabili per impostazione predefinita. Quando si apportano direttive in un modulo angolare separato, in realtà è esportabile e riutilizzabile tra le diverse applicazioni angolari. È possibile aggiungere semplicemente nuove direttive all'interno di reusableModuleApp.js e riusableModuleApp può avere il proprio controller, i servizi, l'oggetto DDO all'interno della direttiva per definire il comportamento.

Direttiva di base con modello e ambito isolato

La creazione di una direttiva personalizzata con *un ambito isolato* separerà l'ambito **all'interno** della direttiva dall'ambito **esterno** , al fine di evitare che la nostra direttiva modifichi accidentalmente i dati nell'ambito genitore e impedisca la lettura dei dati privati dall'ambito principale.

Per creare un ambito isolato e consentire ancora alla nostra direttiva personalizzata di comunicare con l'ambito esterno, possiamo usare l'opzione `scope` che descrive come **mappare** i collegamenti dell'ambito interno della direttiva con l'ambito esterno.

I binding attuali sono realizzati con **attributi** aggiuntivi allegati alla direttiva. Le impostazioni di binding sono definite con l'opzione `scope` e un oggetto con coppie chiave-valore:

- Una **chiave** , che corrisponde alla proprietà dell'ambito isolato della nostra direttiva
- Un **valore** che indica ad Angular come associare l'ambito interno della direttiva a un **attributo** corrispondente

Semplice esempio di una direttiva con un ambito isolato:

```
var ProgressBar = function() {
```

```
  return {
    scope: { // This is how we define an isolated scope
      current: '=', // Create a REQUIRED bidirectional binding by using the 'current'
attribute
      full: '=?maxValue' // Create an OPTIONAL (Note the '?'): bidirectional binding using
'max-value' attribute to the `full` property in our directive isolated scope
    }
    template: '<div class="progress-back">' +
              '  <div class="progress-bar"' +
              '       ng-style="{width: getProgress()}">' +
              '  </div>' +
              '</div>',
    link: function(scope, el, attrs) {
      if (scope.full === undefined) {
        scope.full = 100;
      }
      scope.getProgress = function() {
        return (scope.current / scope.size * 100) + '%';
      }
    }
  }
}

ProgressBar.$inject = [];
angular.module('app').directive('progressBar', ProgressBar);
```

Esempio di come utilizzare questa direttiva e collegare i dati dall'ambito del controller all'ambito interno della direttiva:

controller:

```
angular.module('app').controller('myCtrl', function($scope) {
    $scope.currentProgressValue = 39;
    $scope.maxProgressBarValue = 50;
});
```

Vista:

```
<div ng-controller="myCtrl">
    <progress-bar current="currentProgressValue"></progress-bar>
    <progress-bar current="currentProgressValue" max-value="maxProgressBarValue"></progress-
bar>
</div>
```

Costruire un componente riutilizzabile

Le direttive possono essere utilizzate per costruire componenti riutilizzabili. Ecco un esempio di un componente "casella utente":

userBox.js

```
angular.module('simpleDirective', []).directive('userBox', function() {
  return {
    scope: {
        username: '=username',
```

```
        reputation: '=reputation'
    },
    templateUrl: '/path/to/app/directives/user-box.html'
  };
});
```

Controller.js

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

myApp.controller('Controller', function($scope) {

    $scope.user = "John Doe";
    $scope.rep = 1250;

    $scope.user2 = "Andrew";
    $scope.rep2 = 2850;

});
```

myPage.js

```
<html lang="en" ng-app="myApp">
  <head>
    <script src="/path/to/app/angular.min.js"></script>
    <script src="/path/to/app/js/controllers/Controller.js"></script>
    <script src="/path/to/app/js/directives/userBox.js"></script>
  </head>

  <body>

    <div ng-controller="Controller">
        <user-box username="user" reputation="rep"></user-box>
        <user-box username="user2" reputation="rep2"></user-box>
    </div>

  </body>
</html>
```

user-box.html

```
<div>{{username}}</div>
<div>{{reputation}} reputation</div>
```

Il risultato sarà:

```
John Doe
1250 reputation
Andrew
2850 reputation
```

Decoratore della direttiva

A volte potresti aver bisogno di funzionalità aggiuntive da una direttiva. Invece di riscrivere

(copiare) la direttiva, è possibile modificare il comportamento della direttiva.

Il decoratore verrà eseguito durante la fase $ iniettata.

Per farlo, prova un .config al tuo modulo. La direttiva si chiama myDirective, quindi devi configurare myDirectiveDirective. (questo in una convenzione angolare [leggi sui provider]).

Questo esempio cambierà templateUrl della direttiva:

```
angular.module('myApp').config(function($provide){
        $provide.decorator('myDirectiveDirective', function($delegate){
             var directive = $delegate[0]; // this is the actual delegated, your directive
             directive.templateUrl = 'newTemplate.html'; // you change the directive template
        return $delegate;
    })
});
```

Questo esempio aggiunge un evento onClick all'elemento direttivo quando si fa clic, questo accade durante la fase di compilazione.

```
angular.module('myApp').config(function ($provide) {
        $provide.decorator('myDirectiveTwoDirective', function ($delegate) {
            var directive = $delegate[0];
            var link = directive.link; // this is directive link phase
            directive.compile = function () { // change the compile of that directive
                return function (scope, element, attrs) {
                    link.apply(this, arguments); // apply this at the link phase
                    element.on('click', function(){ // when add an onclick that log hello when
the directive is clicked.
                            console.log('hello!');
                    });
                };
            };
            return $delegate;
        });

    });
```

Un approccio simile può essere utilizzato per provider e servizi.

Ereditarietà e interoperabilità della direttiva

Le direttive angolari js possono essere annidate o rese interoperabili.

In questo esempio, la direttiva Adir espone alla direttiva Bdir il suo controller $ scope, poiché Bdir richiede Adir.

```
angular.module('myApp',[]).directive('Adir', function () {
        return {
            restrict: 'AE',
            controller: ['$scope', function ($scope) {
                    $scope.logFn = function (val) {
                            console.log(val);
                        }
```

```
                }]
            }
    })
```

Assicurati di impostare richiede: '^ Adir' (guarda la documentazione angolare, alcune versioni non richiede ^ carattere).

```
.directive('Bdir', function () {
        return {
            restrict: 'AE',
            require: '^Adir', // Bdir require Adir
            link: function (scope, elem, attr, Parent) {
            // Parent is Adir but can be an array of required directives.
                elem.on('click', function ($event) {
                    Parent.logFn("Hello!"); // will log "Hello! at parent dir scope
                    scope.$apply(); // apply to parent scope.
                });
            }
        }
    }]);
```

Puoi annidare la tua direttiva in questo modo:

```
<div a-dir><span b-dir></span></div>
<a-dir><b-dir></b-dir> </a-dir>
```

Non è necessario che le direttive siano nidificate nel codice HTML.

Leggi Direttive personalizzate online: https://riptutorial.com/it/angularjs/topic/965/direttive-personalizzate

Capitolo 21: eventi

Parametri

parametri	Tipi di valori
evento	Object {name: "eventName", targetScope: scope, defaultPrevented: false, currentScope: childScope}
args	dati che sono stati passati insieme all'esecuzione di un evento

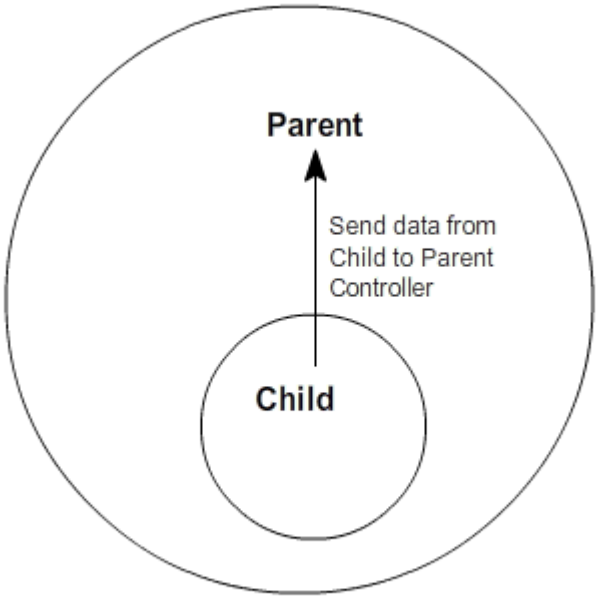
Examples

Usando il sistema di eventi angolari

$ Portata. $ Emettere

Usando `$scope.$emit` genererà un nome di evento verso l'alto attraverso la gerarchia dell'ambito e notificherà a `$scope` . Il ciclo di vita dell'evento parte dall'ambito in cui è stato chiamato `$emit` .

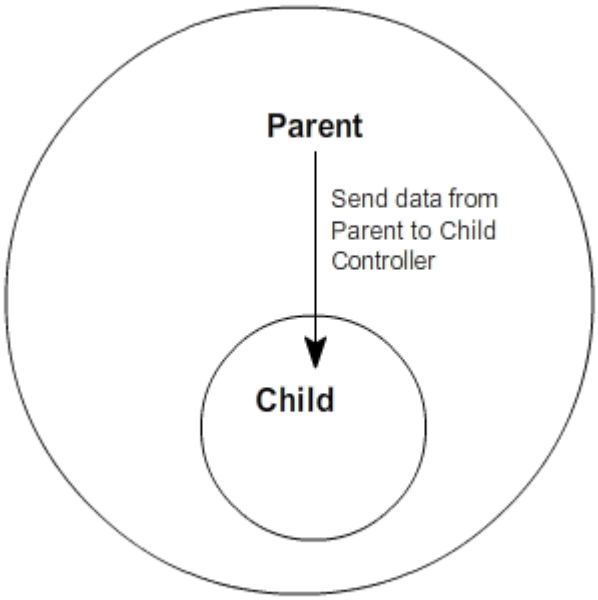
Wireframe funzionante:



$ Portata. $ Trasmissione

Usando `$scope.$broadcast` genererà un evento in `$scope` . Possiamo ascoltare questi eventi usando `$scope.$on`

Wireframe funzionante:



Sintassi:

```
// firing an event upwards
$scope.$emit('myCustomEvent', 'Data to send');

// firing an event downwards
$scope.$broadcast('myCustomEvent', {
  someProp: 'some value'
});

// listen for the event in the relevant $scope
$scope.$on('myCustomEvent', function (event, data) {
  console.log(data); // 'Data from the event'
});
```

Invece di `$scope` puoi usare `$rootScope` , in tal caso il tuo evento sarà disponibile in tutti i controller indipendentemente dall'ambito di questi controller

Pulisci evento registrato in AngularJS

Il motivo per pulire gli eventi registrati perché anche il controller è stato distrutto, la gestione dell'evento registrato è ancora attiva. Quindi il codice funzionerà come di sicuro inaspettato.

```
// firing an event upwards
$rootScope.$emit('myEvent', 'Data to send');

// listening an event
var listenerEventHandler = $rootScope.$on('myEvent', function(){
    //handle code
});

$scope.$on('$destroy', function() {
    listenerEventHandler();
});
```

Usi e significato

Questi eventi possono essere utilizzati per comunicare tra 2 o più controller.

`$emit` invia un evento verso l'alto attraverso la gerarchia dell'ambito, mentre `$broadcast` invia un evento verso il basso a tutti gli ambiti figlio. Questo è stato magnificamente spiegato qui .

Ci possono essere fondamentalmente due tipi di scenario durante la comunicazione tra i controller:

---

1. Quando i controllori hanno una relazione Parent-Child. (possiamo usare principalmente `$scope` in tali scenari)

---

2. Quando i controllori non sono indipendenti l'uno dall'altro e sono necessari per essere informati sulle reciproche attività. (possiamo usare `$rootScope` in tali scenari)

es .: per qualsiasi sito di e-commerce, supponiamo di avere `ProductListController` (che controlla la pagina di elenco dei prodotti quando viene fatto clic su qualsiasi marca di prodotto) e `CartController` (per gestire gli articoli del carrello). Ora, quando clicchiamo sul pulsante **Aggiungi al carrello** , deve essere informato anche su `CartController` , in modo che possa riflettere il conteggio / i dettagli del nuovo articolo carrello nella barra di navigazione del sito. Questo può essere ottenuto usando `$rootScope` .

Con `$scope.$emit`

```
<html>
  <head>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.4/angular.js"></script>
    <script>
        var app = angular.module('app', []);

        app.controller("FirstController", function ($scope) {
            $scope.$on('eventName', function (event, args) {
                $scope.message = args.message;
            });
```

```
        });

        app.controller("SecondController", function ($scope) {
            $scope.handleClick = function (msg) {
                $scope.$emit('eventName', {message: msg});
            };
        });

    </script>
  </head>
  <body ng-app="app">
    <div ng-controller="FirstController" style="border:2px ;padding:5px;">
        <h1>Parent Controller</h1>
        <p>Emit Message : {{message}}</p>
        <br />
        <div ng-controller="SecondController" style="border:2px;padding:5px;">
            <h1>Child Controller</h1>
            <input ng-model="msg">
            <button ng-click="handleClick(msg);">Emit</button>
        </div>
    </div>
  </body>
</html>
```

Con `$scope.$broadcast` :

```
<html>
  <head>
    <title>Broadcasting</title>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.4/angular.js"></script>
    <script>
        var app = angular.module('app', []);

        app.controller("FirstController", function ($scope) {
            $scope.handleClick = function (msg) {
                $scope.$broadcast('eventName', {message: msg});
            };

        });

        app.controller("SecondController", function ($scope) {
            $scope.$on('eventName', function (event, args) {
                $scope.message = args.message;
            });
        });

    </script>
  </head>
  <body ng-app="app">
    <div ng-controller="FirstController" style="border:2px solid ; padding:5px;">
        <h1>Parent Controller</h1>
        <input ng-model="msg">
        <button ng-click="handleClick(msg);">Broadcast</button>
        <br /><br />
        <div ng-controller="SecondController" style="border:2px solid ;padding:5px;">
            <h1>Child Controller</h1>
            <p>Broadcast Message : {{message}}</p>
        </div>
    </div>
  </body>
```

```
</html>
```

Annullare sempre $ rootScope. $ Sugli ascoltatori nell'evento $ destory dell'ambito

$ rootScope. $ sugli ascoltatori rimarrà in memoria se navighi su un altro controller. Ciò creerà una perdita di memoria se il controller non rientra nell'ambito.

non

```
angular.module('app').controller('badExampleController', badExample);

badExample.$inject = ['$scope', '$rootScope'];
function badExample($scope, $rootScope) {

    $rootScope.$on('post:created', function postCreated(event, data) {});

}
```

Fare

```
angular.module('app').controller('goodExampleController', goodExample);

goodExample.$inject = ['$scope', '$rootScope'];
function goodExample($scope, $rootScope) {

    var deregister = $rootScope.$on('post:created', function postCreated(event, data) {});

    $scope.$on('$destroy', function destroyScope() {
        deregister();
    });

}
```

Leggi eventi online: https://riptutorial.com/it/angularjs/topic/1922/eventi

Capitolo 22: filtri

Examples

Il tuo primo filtro

I filtri sono un tipo speciale di funzione che può modificare il modo in cui qualcosa viene stampato sulla pagina, o può essere usato per filtrare un array, o un'azione di `ng-repeat` . Puoi creare un filtro chiamando il metodo `app.filter()` , passandogli un nome e una funzione. Vedere gli esempi di seguito per dettagli sulla sintassi.

Ad esempio, creiamo un filtro che cambierà una stringa in modo che sia tutto maiuscolo (essenzialmente un wrapper della funzione javascript `.toUpperCase()`):

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

// just like making a controller, you must give the
// filter a unique name, in this case "toUppercase"
app.filter('toUppercase', function(){
    // all the filter does is return a function,
    // which acts as the "filtering" function
    return function(rawString){
        // The filter function takes in the value,
        // which we modify in some way, then return
        // back.
        return rawString.toUpperCase();
    };
});
```

Diamo uno sguardo più da vicino a ciò che sta succedendo sopra.

Innanzitutto, stiamo creando un filtro chiamato "toUppercase", che è proprio come un controller; `app.filter(...)` . Quindi, la funzione di quel filtro restituisce la funzione di filtro effettiva. Quella funzione prende un singolo oggetto, che è l'oggetto da filtrare, e dovrebbe restituire la versione filtrata dell'oggetto.

Nota: *in questa situazione, supponiamo che l'oggetto che viene passato al filtro sia una stringa, e quindi sappia utilizzare sempre il filtro solo sulle stringhe. Detto questo, è possibile apportare un ulteriore miglioramento al filtro che esegue il loop dell'oggetto (se si tratta di un array) e quindi rende ogni elemento una stringa maiuscola.*

Ora usiamo il nostro nuovo filtro in azione. Il nostro filtro può essere utilizzato in due modi, sia in un modello angolare o come una funzione javascript (come riferimento angolare iniettato).

Javascript

Basta iniettare l'oggetto `$filter` angolare sul controller, quindi utilizzarlo per recuperare la funzione

filtro usando il suo nome.

```
app.controller("MyController", function($scope, $filter){
    this.rawString = "Foo";
    this.capsString = $filter("toUppercase")(this.rawString);
});
```

HTML

Per una direttiva angolare, utilizzare il simbolo pipe (`|`) seguito dal nome del filtro nella direttiva dopo la stringa effettiva. Ad esempio, diciamo che abbiamo un controller chiamato `MyController` che ha una stringa chiamata `rawString` come elemento di essa.

```
<div ng-controller="MyController as ctrl">
    <span>Capital rawString: {{ ctrl.rawString | toUppercase }}</span>
</div>
```

Nota del redattore: *Angular ha un numero di filtri incorporati, incluso "maiuscolo", e il filtro "toUppercase" è inteso solo come una demo per mostrare facilmente come funzionano i filtri, ma* non è *necessario creare la propria funzione maiuscola.*

Filtro personalizzato per rimuovere i valori

Un tipico caso d'uso per un filtro è quello di rimuovere i valori da una matrice. In questo esempio passiamo in una matrice e rimuoviamo tutti i null trovati in essa, restituendo la matrice.

```
function removeNulls() {
    return function(list) {
        for (var i = list.length - 1; i >= 0; i--) {
            if (typeof list[i] === 'undefined' ||
                    list[i] === null) {
                list.splice(i, 1);
            }
        }
        return list;
    };
}
```

Quello sarebbe usato nel HTML come

```
{{listOfItems | removeNulls}}
```

o in un controller come

```
listOfItems = removeNullsFilter(listOfItems);
```

Filtro personalizzato per formattare i valori

Un altro caso d'uso per i filtri è quello di formattare un singolo valore. In questo esempio,

passiamo un valore e viene restituito un valore booleano vero appropriato.

```
function convertToBooleanValue() {
    return function(input) {
        if (typeof input !== 'undefined' &&
                input !== null &&
                (input === true || input === 1 || input === '1' || input
                        .toString().toLowerCase() === 'true')) {
            return true;
        }
        return false;
    };
}
```

Quale in HTML dovrebbe essere usato in questo modo:

```
{{isAvailable | convertToBooleanValue}}
```

O in un controller come:

```
var available = convertToBooleanValueFilter(isAvailable);
```

Esecuzione del filtro in un array figlio

Questo esempio è stato fatto per dimostrare come è possibile eseguire un filtro approfondito in un array *figlio* senza la necessità di un filtro personalizzato.

controller:

```
(function() {
  "use strict";
  angular
    .module('app', [])
    .controller('mainCtrl', mainCtrl);

  function mainCtrl() {
    var vm = this;

    vm.classifications = ["Saloons", "Sedans", "Commercial vehicle", "Sport car"];
    vm.cars = [
       {
          "name":"car1",
          "classifications":[
             {
                "name":"Saloons"
             },
             {
                "name":"Sedans"
             }
          ]
       },
       {
          "name":"car2",
          "classifications":[
             {
```

```
                "name":"Saloons"
             },
             {
                "name":"Commercial vehicle"
             }
          ]
       },
       {
          "name":"car3",
          "classifications":[
             {
                "name":"Sport car"
             },
             {
                "name":"Sedans"
             }
          ]
       }
    ];
  }
})();
```

Vista:

```
<body ng-app="app" ng-controller="mainCtrl as main">
  Filter car by classification:
  <select ng-model="classificationName"
          ng-options="classification for classification in main.classifications"></select>
  <br>
  <ul>
    <li ng-repeat="car in main.cars |
                   filter: { classifications: { name: classificationName } } track by $index"
                  ng-bind-template="{{car.name}} - {{car.classifications | json}}">
    </li>
  </ul>
</body>
```

Controlla la DEMO completa.

Utilizzo dei filtri in un controller o servizio

Iniettando `$filter` , qualsiasi filtro definito nel modulo Angular può essere utilizzato in controller, servizi, direttive o anche altri filtri.

```
angular.module("app")
  .service("users", usersService)
  .controller("UsersController", UsersController);

function usersService () {
  this.getAll = function () {
    return [{
      id: 1,
      username: "john"
    }, {
      id: 2,
      username: "will"
    }, {
```

```
      id: 3,
      username: "jack"
    }];
  };
}

function UsersController ($filter, users) {
  var orderByFilter = $filter("orderBy");

  this.users = orderByFilter(users.getAll(), "username");
  // Now the users are ordered by their usernames: jack, john, will

  this.users = orderByFilter(users.getAll(), "username", true);
  // Now the users are ordered by their usernames, in reverse order: will, john, jack
}
```

Accedere a un elenco filtrato al di fuori di una ng-repeat

Occasionalmente vorrai accedere al risultato dei tuoi filtri al di fuori della `ng-repeat` , forse per indicare il numero di elementi che sono stati filtrati. Puoi farlo usando `as [variablename]` sintassi `as [variablename]` sulla `ng-repeat` .

```
<ul>
  <li ng-repeat="item in vm.listItems | filter:vm.myFilter as filtered">
    {{item.name}}
  </li>
</ul>
<span>Showing {{filtered.length}} of {{vm.listItems.length}}</span>
```

Leggi filtri online: https://riptutorial.com/it/angularjs/topic/1401/filtri

Capitolo 23: Filtri personalizzati

Examples

Esempio di filtro semplice

I filtri formattano il valore di un'espressione per la visualizzazione all'utente. Possono essere utilizzati in modelli di visualizzazione, controller o servizi. Questo esempio crea un filtro (`addZ`) quindi lo utilizza in una vista. Tutto questo filtro fa aggiungere una "Z" maiuscola alla fine della stringa.

example.js

```
angular.module('main', [])
    .filter('addZ', function() {
            return function(value) {
                return value + "Z";
            }
     })
    .controller('MyController', ['$scope', function($scope) {
        $scope.sample = "hello";
    }])
```

example.html

All'interno della vista, il filtro viene applicato con la seguente sintassi: `{ variable | filter}` . In questo caso, la variabile che abbiamo definito nel controller, `sample` , viene filtrata dal filtro che abbiamo creato, `addZ` .

```
<div ng-controller="MyController">
   <span>{{sample | addZ}}</span>
</div>
```

Uscita prevista

```
helloZ
```

Utilizzare un filtro in un controller, un servizio o un filtro

Dovrai iniettare `$filter` :

```
angular
  .module('filters', [])
  .filter('percentage', function($filter) {
    return function (input) {
      return $filter('number')(input * 100) + ' %';
```

```
    };
  });
```

Crea un filtro con parametri

Per impostazione predefinita, un filtro ha un singolo parametro: la variabile su cui è applicato. Ma puoi passare più parametri alla funzione:

```
angular
  .module('app', [])
  .controller('MyController', function($scope) {
    $scope.example = 0.098152;
  })
  .filter('percentage', function($filter) {
    return function (input, decimals) {
      return $filter('number')(input * 100, decimals) + ' %';
    };
  });
```

Ora puoi dare una precisione al filtro `percentage` :

```
<span ng-controller="MyController">{{ example | percentage: 2 }}</span>
=> "9.81 %"
```

... ma altri parametri sono opzionali, puoi comunque utilizzare il filtro predefinito:

```
<span ng-controller="MyController">{{ example | percentage }}</span>
=> "9.8152 %"
```

Leggi Filtri personalizzati online: https://riptutorial.com/it/angularjs/topic/2552/filtri-personalizzati

Capitolo 24: Filtri personalizzati con ES6

Examples

FileSize Filter utilizzando ES6

Abbiamo qui un filtro Dimensione file per descrivere come aggiungere un filtro di costo a un modulo esistente:

```
let fileSize=function (size,unit,fixedDigit) {
return size.toFixed(fixedDigit) + ' '+unit;
};

  let fileSizeFilter=function () {
        return function (size) {
            if (isNaN(size))
                size = 0;

            if (size < 1024)
                return size + ' octets';

            size /= 1024;

            if (size < 1024)
               return fileSize(size,'Ko',2);

            size /= 1024;

            if (size < 1024)
               return  fileSize(size,'Mo',2);

            size /= 1024;

            if (size < 1024)
               return  fileSize(size,'Go',2);

            size /= 1024;
            return  fileSize(size,'To',2);
        };
    };
export default fileSizeFilter;
```

Il filtro chiama nel modulo:

```
import fileSizeFilter from 'path...';
let myMainModule =
    angular.module('mainApp', [])
   .filter('fileSize', fileSizeFilter);
```

Il codice html dove chiamiamo il filtro:

```
<div ng-app="mainApp">
```

```
    <div>
        <input type="text" ng-model="size" />
    </div>
    <div>
        <h3>Output:</h3>
        <p>{{size| Filesize}}</p>
    </div>
</div>
```

Leggi Filtri personalizzati con ES6 online: https://riptutorial.com/it/angularjs/topic/9421/filtri-personalizzati-con-es6

Capitolo 25: Funzioni ausiliarie integrate

Examples

angular.equals

La funzione `angular.equals` confronta e determina se 2 oggetti o valori sono uguali, `angular.equals` esegue un confronto profondo e restituisce true se e solo se almeno una delle seguenti condizioni è soddisfatta.

> angular.equals (valore1, valore2)

1. Se gli oggetti o i valori superano il confronto `===`
2. Se entrambi gli oggetti o valori sono dello stesso tipo e tutte le loro proprietà sono uguali utilizzando `angular.equals`
3. Entrambi i valori sono uguali a `NaN`
4. Entrambi i valori rappresentano il risultato della stessa espressione regolare.

Questa funzione è utile quando è necessario confrontare in profondità gli oggetti o gli array con i loro valori o risultati piuttosto che con semplici riferimenti.

Esempi

```
angular.equals(1, 1) // true
angular.equals(1, 2) // false
angular.equals({}, {}) // true, note that {}==={} is false
angular.equals({a: 1}, {a: 1}) // true
angular.equals({a: 1}, {a: 2}) // false
angular.equals(NaN, NaN) // true
```

angular.isString

La funzione `angular.isString` restituisce true se l'oggetto o il valore ad esso assegnato è del tipo `string`

> angular.isString (valore1)

Esempi

```
angular.isString("hello") // true
angular.isString([1, 2]) // false
angular.isString(42) // false
```

Questo è l'equivalente di esibirsi

```
typeof someValue === "string"
```

angular.isArray

La funzione `angular.isArray` restituisce true se e solo se l'oggetto o il valore passato ad esso è del tipo `Array` .

angular.isArray (valore)

Esempi

```
angular.isArray([]) // true
angular.isArray([2, 3]) // true
angular.isArray({}) // false
angular.isArray(17) // false
```

È l'equivalente di

```
Array.isArray(someValue)
```

angular.merge

La funzione angular.merge prende tutte le proprietà enumerabili dall'oggetto sorgente per estendere profondamente l'oggetto di destinazione.

La funzione restituisce un riferimento all'oggetto di destinazione ora esteso

angular.merge (destination, source)

Esempi

```
angular.merge({}, {}) // {}
angular.merge({name: "king roland"}, {password: "12345"})
// {name: "king roland", password: "12345"}
angular.merge({a: 1}, [4, 5, 6]) // {0: 4, 1: 5, 2: 6, a: 1}
angular.merge({a: 1}, {b: {c: {d: 2}}}) // {"a":1,"b":{"c":{"d":2}}}
```

angular.isDefined and angular.isUndefined

La funzione `angular.isDefined` verifica un valore se è definito

angular.isDefined (someValue)

Questo è l'equivalente di esibirsi

```
value !== undefined; // will evaluate to true is value is defined
```

Esempi

```
angular.isDefined(42) // true
angular.isDefined([1, 2]) // true
angular.isDefined(undefined) // false
```

```
angular.isDefined(null) // true
```

La funzione `angular.isUndefined` verifica se un valore non è definito (è effettivamente l'opposto di `angular.isDefined`)

    angular.isUndefined (someValue)

Questo è l'equivalente di esibirsi

```
value === undefined; // will evaluate to true is value is undefined
```

O semplicemente

```
!angular.isDefined(value)
```

Esempi

```
angular.isUndefined(42) // false
angular.isUndefined(undefined) // true
```

angular.isDate

La funzione `angular.isDate` restituisce true se e solo se l'oggetto passato ad esso è del tipo Date.

    angular.isDate (valore)

Esempi

```
angular.isDate("lone star") // false
angular.isDate(new Date()) // true
```

angular.isNumber

La funzione `angular.isNumber` restituisce true se e solo se l'oggetto o il valore passato ad esso è del tipo Number, questo include + Infinity, -Infinity e NaN

    angular.isNumber (valore)

Questa funzione non causerà un tipo di coercizione come

```
"23" == 23 // true
```

Esempi

```
angular.isNumber("23") // false
angular.isNumber(23) // true
angular.isNumber(NaN) // true
angular.isNumber(Infinity) // true
```

Questa funzione non causerà un tipo di coercizione come

```
"23" == 23 // true
```

angular.isFunction

La funzione `angular.isFunction` determina e restituisce true se e solo se il valore passato è un riferimento a una funzione.

La funzione restituisce un riferimento all'oggetto di destinazione ora esteso

    angular.isFunction (fn)

Esempi

```
var onClick = function(e) {return e};
angular.isFunction(onClick); // true

var someArray = ["pizza", "the", "hut"];
angular.isFunction(someArray ); // false
```

angular.toJson

La funzione `angular.toJson` prenderà un oggetto e lo serializzerà in una stringa formattata JSON.

A differenza della funzione nativa `JSON.stringify` , questa funzione rimuoverà tutte le proprietà che iniziano con `$$` (come angolare di solito prefigura le proprietà interne con `$$`)

```
angular.toJson(object)
```

Poiché i dati devono essere serializzati prima di passare attraverso una rete, questa funzione è utile per trasformare qualsiasi dato che desideri trasmettere in JSON.

Questa funzione è utile anche per il debug in quanto funziona in modo simile a un metodo `.toString` .

Esempi:

```
angular.toJson({name: "barf", occupation: "mog", $$somebizzareproperty: 42})
// "{"name":"barf","occupation":"mog"}"
angular.toJson(42)
// "42"
angular.toJson([1, "2", 3, "4"])
// "[1,"2",3,"4"]"
var fn = function(value) {return value}
angular.toJson(fn)
// undefined, functions have no representation in JSON
```

angular.fromJson

La funzione `angular.fromJson` deserializza una stringa JSON valida e restituisce un oggetto o una matrice.

angular.fromJson (string | oggetto)

Si noti che questa funzione non è limitata alle sole stringhe, ma produrrà una rappresentazione di qualsiasi oggetto passato ad esso.

Esempi:

```
angular.fromJson("{\"yogurt\": \"strawberries\"}")
// Object {yogurt: "strawberries"}
angular.fromJson('{jam: "raspberries"}')
// will throw an exception as the string is not a valid JSON
angular.fromJson(this)
// Window {external: Object, chrome: Object, _gaq: Y, angular: Object, ng339: 3…}
angular.fromJson([1, 2])
// [1, 2]
typeof angular.fromJson(new Date())
// "object"
```

angular.noop

L' `angular.noop` è una funzione che non esegue operazioni, si passa `angular.noop` quando è necessario fornire un argomento di funzione che non farà nulla.

angular.noop ()

Un uso comune di `angular.noop` può essere quello di fornire un callback vuoto a una funzione che altrimenti genera un errore quando viene passato ad esso qualcosa di diverso da una funzione.

Esempio:

```
$scope.onSomeChange = function(model, callback) {
    updateTheModel(model);
    if (angular.isFunction(callback)) {
        callback();
    } else {
        throw new Error("error: callback is not a function!");
    }
};

$scope.onSomeChange(42, function() {console.log("hello callback")});
// will update the model and print 'hello callback'
$scope.onSomeChange(42, angular.noop);
// will update the model
```

Ulteriori esempi:

```
angular.noop() // undefined
angular.isFunction(angular.noop) // true
```

angular.isObject

`angular.isObject` restituisce true se e solo se l'argomento passato ad esso è un oggetto, questa funzione restituirà anche true per una matrice e restituirà false per `null` anche se `typeof null` è `object` .

angular.isObject (valore)

Questa funzione è utile per il controllo del tipo quando è necessario elaborare un oggetto definito.

Esempi:

```
angular.isObject({name: "skroob", job: "president"})
// true
angular.isObject(null)
// false
angular.isObject([null])
// true
angular.isObject(new Date())
// true
angular.isObject(undefined)
// false
```

angular.isElement

`angular.isElement` restituisce true se l'argomento passato ad esso è un elemento DOM o un elemento jQuery spostato.

angular.isElement (elem)

Questa funzione è utile per digitare check se un argomento passato è un elemento prima di essere elaborato come tale.

Esempi:

```
angular.isElement(document.querySelector("body"))
// true
angular.isElement(document.querySelector("#some_id"))
// false if "some_id" is not using as an id inside the selected DOM
angular.isElement("<div></div>")
// false
```

angular.copy

La funzione `angular.copy` accetta un oggetto, una matrice o un valore e ne crea una copia profonda.

angular.copy ()

Esempio:

Oggetti:

```
let obj = {name: "vespa", occupation: "princess"};
```

```
let cpy = angular.copy(obj);
cpy.name = "yogurt"
// obj = {name: "vespa", occupation: "princess"}
// cpy = {name: "yogurt", occupation: "princess"}
```

Array:

```
var w = [a, [b, [c, [d]]]];
var q = angular.copy(w);
// q = [a, [b, [c, [d]]]]
```

Nell'esempio precedente `angular.equals(w, q)` valuterà true perché `.equals` verifica l'uguaglianza per valore. tuttavia `w === q` valuterà come falso perché il confronto stretto tra oggetti e matrici viene fatto per riferimento.

angular.identity

La funzione `angular.identity` restituisce il primo argomento passato ad esso.

    angular.identity (argomento)

Questa funzione è utile per la programmazione funzionale, è possibile fornire questa funzione come predefinita nel caso in cui non sia stata superata una funzione prevista.

Esempi:

```
angular.identity(42) // 42
```

```
var mutate = function(fn, num) {
    return angular.isFunction(fn) ? fn(num) : angular.identity(num)
}

mutate(function(value) {return value-7}, 42) // 35
mutate(null, 42) // 42
mutate("mount. rushmore", 42) // 42
```

angular.forEach

`angular.forEach` accetta un oggetto e una funzione iteratore. Quindi esegue la funzione iteratore su ciascuna proprietà / valore enumerabile dell'oggetto. Questa funzione funziona anche sugli array.

Come la versione JS di `Array.prototype.forEach` La funzione non esegue iterazioni sulle proprietà ereditate (proprietà prototype), tuttavia la funzione non tenterà di elaborare un valore `null` o `undefined` e lo restituirà semplicemente.

    angular.forEach (oggetto, funzione (valore, chiave) {// funzione});

Esempi:

```
angular.forEach({"a": 12, "b": 34}, (value, key) => console.log("key: " + key + ", value: " +
```

```
value))
// key: a, value: 12
// key: b, value: 34
angular.forEach([2, 4, 6, 8, 10], (value, key) => console.log(key))
// will print the array indices: 1, 2, 3, 4, 5
angular.forEach([2, 4, 6, 8, 10], (value, key) => console.log(value))
// will print the array values: 2, 4, 6, 7, 10
angular.forEach(undefined, (value, key) => console.log("key: " + key + ", value: " + value))
// undefined
```

Leggi Funzioni ausiliarie integrate online: https://riptutorial.com/it/angularjs/topic/3032/funzioni-ausiliarie-integrate

Capitolo 26: Il Sé o questa variabile in un controller

introduzione

Questa è una spiegazione di un modello comune e generalmente considerata la migliore pratica che è possibile vedere nel codice AngularJS.

Examples

Comprensione dello scopo del sé variabile

Quando si utilizza "controller come sintassi" si darebbe al controller un alias nell'html quando si utilizza la direttiva ng-controller.

```
<div ng-controller="MainCtrl as main">
</div>
```

È quindi possibile accedere a proprietà e metodi dalla variabile ***principale*** che rappresenta l'istanza del controller. Ad esempio, accedi alla proprietà di ***saluto*** del nostro controller e visualizzalo sullo schermo:

```
<div ng-controller="MainCtrl as main">
    {{ main.greeting }}
</div>
```

Ora, nel nostro controller, dobbiamo impostare un valore per la proprietà di saluto della nostra istanza controller (diversamente da $ scope o qualcos'altro):

```
angular
.module('ngNjOrg')
.controller('ForgotPasswordController',function ($log) {
  var self = this;

  self.greeting = "Hello World";
})
```

Per poter visualizzare correttamente l'HTML, era necessario impostare la proprietà di saluto su ***questo*** all'interno del corpo del controller. Sto creando una variabile intermedia denominata ***self*** che contiene un riferimento a questo. Perché? Considera questo codice:

```
angular
.module('ngNjOrg')
.controller('ForgotPasswordController',function ($log) {
  var self = this;
```

```
  self.greeting = "Hello World";

  function itsLate () {
    this.greeting = "Goodnight";
  }

})
```

In questo codice sopra puoi aspettarti che il testo sullo schermo si aggiorni quando viene chiamato il metodo *itsLate* , ma di fatto non lo fa. JavaScript utilizza le regole di scoping a livello di funzione in modo che il "questo" all'interno di itsLate si riferisca a qualcosa di diverso che "questo" al di fuori del corpo del metodo. Tuttavia, possiamo ottenere il risultato desiderato se usiamo la variabile ***automatica*** :

```
 angular
.module('ngNjOrg')
.controller('ForgotPasswordController',function ($log) {
  var self = this;

  self.greeting = "Hello World";

  function itsLate () {
    self.greeting = "Goodnight";
  }

})
```

Questa è la bellezza dell'uso di una variabile "autonoma" nei controller: è possibile accedervi da qualsiasi parte nel controller e sempre essere sicuri che faccia riferimento all'istanza del controller.

Leggi Il Sé o questa variabile in un controller online: https://riptutorial.com/it/angularjs/topic/8867/il-se-o-questa-variabile-in-un-controller

Capitolo 27: Iniezione di dipendenza

Sintassi

- myApp.controller ('MyController', function ($ scope) {...}); // codice non minificato
- myApp.controller ('MyController', ['$ scope', function ($ scope) {...}]); // supporto minification
- function MyController () {}

  MyController. $ Inject = ['$ scope'];

  myApp.controller ('MyController', MyController); // $ inserisce l'annotazione
- $ Injector.get ('iniettabile'); // Iniezione dinamica / runtime

Osservazioni

I provider non possono essere iniettati in blocchi di `run` .

Servizi o valori non possono essere iniettati in blocchi di `config` .

Assicurati di annotare le iniezioni in modo che il codice non si interrompa in minification.

Examples

iniezioni

L'esempio più semplice di un'iniezione in un'app Angular: l'iniezione di `$scope` a un `Controller` angolare:

```
angular.module('myModule', [])
.controller('myController', ['$scope', function($scope) {
    $scope.members = ['Alice', 'Bob'];
    ...
}])
```

Quanto sopra illustra un'iniezione di `$scope` in un `controller` , ma è lo stesso se si inserisce un modulo in un altro. Il processo è lo stesso.

Il sistema di Angular è incaricato di risolvere le dipendenze per te. Se crei un servizio, ad esempio, puoi elencarlo come nell'esempio sopra e sarà disponibile per te.

È possibile utilizzare DI - Dipendenza iniezione - ovunque si stia definendo un componente.

Si noti che nell'esempio precedente viene utilizzata la cosiddetta "Annotazione di array in linea". Significa, scriviamo esplicitamente come stringhe i nomi delle nostre dipendenze. Lo facciamo per

prevenire l'interruzione dell'applicazione quando il codice viene ridotto per la produzione. La minificazione del codice cambia i nomi delle variabili (ma non le stringhe), che interrompe l'iniezione. Usando le stringhe, Angular sa quali dipendenze vogliamo.

Molto importante: l'ordine dei nomi delle stringhe deve essere uguale ai parametri nella funzione.

Esistono strumenti che automatizzano questo processo e si prendono cura di questo per te.

Iniezioni dinamiche

C'è anche un'opzione per richiedere dinamicamente i componenti. Puoi farlo usando il servizio `$injector` :

```
myModule.controller('myController', ['$injector', function($injector) {
    var myService = $injector.get('myService');
}]);
```

Nota: mentre questo metodo potrebbe essere usato per prevenire il problema della dipendenza circolare che potrebbe interrompere la tua app, non è considerata una buona pratica aggirare il problema usandolo. La dipendenza circolare di solito indica che c'è un'anomalia nell'architettura della tua applicazione, e dovresti invece indirizzarla.

$ inietta annotazione di proprietà

Equivalentemente, possiamo usare l'annotazione di proprietà `$inject` per ottenere lo stesso risultato sopra:

```
var MyController = function($scope) {
  // ...
}
MyController.$inject = ['$scope'];
myModule.controller('MyController', MyController);
```

Carica dinamicamente il servizio AngularJS in JavaScript vaniglia

È possibile caricare i servizi AngularJS in JavaScript vanilla utilizzando il metodo `injector()` AngularJS. Ogni elemento jqLite richiamato chiamando `angular.element()` ha un metodo `injector()` che può essere utilizzato per recuperare l'iniettore.

```
var service;
var serviceName = 'myService';

var ngAppElement = angular.element(document.querySelector('[ng-app],[data-ng-app]') ||
document);
var injector = ngAppElement.injector();

if(injector && injector.has(serviceNameToInject)) {
    service = injector.get(serviceNameToInject);
}
```

Nell'esempio precedente proviamo a recuperare l'elemento jqLite che contiene la radice dell'applicazione AngularJS (`ngAppElement`). Per fare ciò, usiamo il metodo `angular.element()` , cercando un elemento DOM contenente l'attributo `ng-app` o `data-ng-app` o, se non esiste, torniamo all'elemento `document` . Usiamo `ngAppElement` per recuperare l'istanza di injector (con `ngAppElement.injector()`). L'istanza di injector viene utilizzata per verificare se il servizio da iniettare esiste (con `injector.has()`) e quindi per caricare il servizio (con `injector.get()`) all'interno della variabile di `service` .

Leggi Iniezione di dipendenza online: https://riptutorial.com/it/angularjs/topic/1582/iniezione-di-dipendenza

Capitolo 28: Interceptor HTTP

introduzione

Il servizio $ http di AngularJS ci consente di comunicare con un back-end e fare richieste HTTP. Ci sono casi in cui vogliamo catturare ogni richiesta e manipolarla prima di inviarla al server. Altre volte vorremmo acquisire la risposta e elaborarla prima di completare la chiamata. La gestione globale degli errori HTTP può essere anche un buon esempio di tale necessità. Gli intercettori sono creati esattamente per questi casi.

Examples

Iniziare

Il servizio `$http` integrato di Angular ci consente di inviare richieste HTTP. Oftentime, la necessità di fare cose prima o dopo una richiesta, ad esempio aggiungendo a ciascuna richiesta un token di autenticazione o creando una logica di gestione degli errori generica.

HttpInterceptor generico passo dopo passo

Crea un file HTML con il seguente contenuto:

```
<!DOCTYPE html>
<html>
<head>
  <title>Angular Interceptor Sample</title>
  <script src="https://code.angularjs.org/1.5.8/angular.min.js"></script>
  <script src="app.js"></script>
  <script src="appController.js"></script>
  <script src="genericInterceptor.js"></script>
</head>
<body ng-app="interceptorApp">
    <div ng-controller="appController as vm">
        <button ng-click="vm.sendRequest()">Send a request</button>
    </div>
</body>
</html>
```

Aggiungi un file JavaScript chiamato 'app.js':

```
var interceptorApp = angular.module('interceptorApp', []);

interceptorApp.config(function($httpProvider) {
        $httpProvider.interceptors.push('genericInterceptor');
});
```

Aggiungi un altro chiamato "appController.js":

```
(function() {
```

```
    'use strict';

    function appController($http) {
        var vm = this;

        vm.sendRequest = function(){
            $http.get('http://google.com').then(function(response){
                console.log(response);
            });
        };
    }

    angular.module('interceptorApp').controller('appController',['$http', appController]);
})();
```

E infine il file contenente l'intercettore stesso 'genericInterceptor.js':

```
(function() {
    "use strict";

    function genericInterceptor($q) {
        this.responseError = function (response) {
            return $q.reject(response);
        };

        this.requestError = function(request){
            if (canRecover(rejection)) {
                return responseOrNewPromise
            }
            return $q.reject(rejection);
        };

        this.response = function(response){
            return response;
        };

        this.request = function(config){
            return config;
        }
    }

    angular.module('interceptorApp').service('genericInterceptor', genericInterceptor);
})();
```

Il 'genericInterceptor' copre le possibili funzioni che possiamo sovrascrivere aggiungendo un comportamento extra alla nostra applicazione.

Messaggio flash sulla risposta utilizzando l'intercettore http

Nel file di visualizzazione

Nel file html di base (index.html) in cui di solito includiamo gli script angolari o l'html condiviso nell'app, lasciamo un elemento div vuoto, i messaggi flash verranno visualizzati all'interno di questo elemento div

```
<div class="flashmessage" ng-if="isVisible">
    {{flashMessage}}
</div>
```

File di script

Nel metodo di configurazione del modulo angolare, iniettare httpProvider, httpProvider ha una proprietà di matrice dell'intercettore, premere l'intercettore personalizzato. Nell'esempio corrente l'intercettore personalizzato intercetta solo la risposta e chiama un metodo collegato a rootScope.

```
var interceptorTest = angular.module('interceptorTest', []);

    interceptorTest.config(['$httpProvider',function ($httpProvider) {

        $httpProvider.interceptors.push(["$rootScope",function ($rootScope) {
            return {     //intercept only the response
                        'response': function (response)
                                    {

$rootScope.showFeedBack(response.status,response.data.message);

                                      return response;
                                    }
                    };
        }]);

    }])
```

Poiché solo i provider possono essere iniettati nel metodo di configurazione di un modulo angolare (ovvero httpProvider e non il rootscope), dichiarare il metodo collegato a rootscope all'interno del metodo run del modulo angolare.

Visualizza anche il messaggio all'interno di $ timeout in modo che il messaggio abbia la proprietà flash, che scompare dopo un tempo limite. Nel nostro esempio i suoi 3000 ms.

```
interceptorTest.run(["$rootScope","$timeout",function($rootScope,$timeout){
     $rootScope.showFeedBack = function(status,message){

        $rootScope.isVisible = true;
        $rootScope.flashMessage = message;
        $timeout(function(){$rootScope.isVisible = false },3000)
    }
}]);
```

Insidie comuni

Cercando di iniettare **$ rootScope o altri servizi** all'interno del metodo di **configurazione** del modulo angolare, il ciclo di vita di un'app angolare non consente di **generare quell'errore di** provider sconosciuto. Solo i **fornitori** possono essere iniettati nel metodo di **configurazione** del modulo angolare

Leggi Interceptor HTTP online: https://riptutorial.com/it/angularjs/topic/6484/interceptor-http

Capitolo 29: Migrazione verso Angular 2+

introduzione

AngularJS è stato completamente riscritto utilizzando il linguaggio TypeScript e rinominato in Angular.

C'è molto che può essere fatto a un'app AngularJS per facilitare il processo di migrazione. Come dice la guida ufficiale all'aggiornamento , è possibile eseguire diversi "passaggi di preparazione" per ridefinire la tua app, rendendola sempre più vicina al nuovo stile angolare.

Examples

Conversione della tua app AngularJS in una struttura orientata ai componenti

Nel nuovo framework Angular, i **componenti** sono i principali elementi costitutivi che compongono l'interfaccia utente. Quindi, uno dei primi passaggi che consente a un'app AngularJS di essere migrato al nuovo Angular consiste nel rifattorizzarlo in una struttura più orientata ai componenti.

I componenti sono stati introdotti anche nel vecchio AngularJS a partire dalla versione **1.5+** . L'uso di componenti in un'app AngularJS non solo renderà la sua struttura più vicina al nuovo Angular 2+, ma lo renderà anche più modulare e più facile da mantenere.

Prima di andare oltre, consiglio di consultare la pagina ufficiale della documentazione di AngularJS sui componenti , in cui i loro vantaggi e il loro utilizzo sono ben spiegati.

Vorrei piuttosto menzionare alcuni suggerimenti su come convertire il vecchio codice orientato da `ng-controller` nel nuovo stile orientato ai `component` .

Inizia a rompere la tua app in componenti

Tutte le app orientate ai componenti hanno in genere uno o alcuni componenti che includono altri sotto-componenti. Quindi, perché non creare il primo componente che conterrà semplicemente la tua app (o una grande parte di essa).

Supponiamo di avere un pezzo di codice assegnato a un controller, denominato `UserListController` , e vogliamo creare un componente di esso, che `UserListComponent` .

HTML corrente:

```
<div ng-controller="UserListController as listctrl">
    <ul>
        <li ng-repeat="user in myUserList">
            {{ user }}
        </li>
```

```
    </ul>
</div>
```

JavaScript attuale:

```
app.controller("UserListController", function($scope, SomeService) {

    $scope.myUserList = ['Shin', 'Helias', 'Kalhac'];

    this.someFunction = function() {
        // ...
    }

    // ...
}
```

nuovo HTML:

```
<user-list></user-list>
```

nuovo JavaScript:

```
app.component("UserList", {
    templateUrl: 'user-list.html',
    controller: UserListController
});

function UserListController(SomeService) {

    this.myUserList = ['Shin', 'Helias', 'Kalhac'];

    this.someFunction = function() {
        // ...
    }

    // ...
}
```

Nota come non stiamo più iniettando `$scope` nella funzione controller e ora stiamo dichiarando `this.myUserList` invece di `$scope.myUserList` ;

nuovo file modello `user-list.component.html` :

```
<ul>
    <li ng-repeat="user in $ctrl.myUserList">
        {{ user }}
    </li>
</ul>
```

Nota come ora ci stiamo riferendo alla variabile `myUserList` , che appartiene al controller, usando `$ctrl.myUserList` dal html invece di `$scope.myUserList` .

Questo perché, come probabilmente avete capito dopo aver letto la documentazione, `$ctrl` nel template ora si riferisce alla funzione controller.

E i controller e le rotte?

Nel caso in cui il controller fosse associato al modello usando il sistema di routing invece di `ng-controller` , quindi se si ha qualcosa del genere:

```
$stateProvider
  .state('users', {
    url: '/users',
    templateUrl: 'user-list.html',
    controller: 'UserListController'
  })
  // ..
```

puoi semplicemente cambiare la dichiarazione dello stato in:

```
$stateProvider
    .state('users', {
      url: '/',
      template: '<user-list></user-list>'
    })
    // ..
```

Qual'è il prossimo?

Ora che hai un componente contenente la tua app (se contiene l'intera applicazione o parte di essa, come una vista), dovresti iniziare a suddividere il tuo componente in più componenti annidati, avvolgendone parti in nuovi sotto-componenti , e così via.

Dovresti iniziare a usare le caratteristiche del componente come

- **Input** e **Output** vincoli
- **hook del ciclo di vita** come `$onInit()` , `$onChanges()` , ecc ...

Dopo aver letto la documentazione del componente si dovrebbe già sapere come utilizzare tutte quelle caratteristiche dei componenti, ma se avete bisogno di un esempio concreto di un vero e proprio semplice applicazione, è possibile controllare questo .

Inoltre, se all'interno del controller del componente sono presenti alcune funzioni che contengono molto codice logico, è consigliabile prendere in considerazione l'idea di spostare tale logica in servizi .

Conclusione

Adottando un approccio basato sui componenti, il tuo AngularJS si avvicina di un passo per migrarlo nel nuovo framework Angular, ma lo rende anche migliore e molto più modulare.

Naturalmente ci sono molti altri passi che puoi fare per andare oltre nella nuova direzione Angular 2+, che elencherò nei seguenti esempi.

Presentazione dei moduli Webpack e ES6

Utilizzando un ***caricatore di moduli*** come Webpack possiamo beneficiare del sistema di moduli integrato disponibile in **ES6** (così come in **TypeScript**). Possiamo quindi utilizzare le funzionalità di importazione ed esportazione che ci consentono di specificare quali parti di codice possiamo condividere tra diverse parti dell'applicazione.

Quando poi portiamo le nostre applicazioni in produzione, i caricatori di moduli semplificano anche il confezionamento di tutti i pacchetti di produzione con batterie incluse.

Leggi Migrazione verso Angular 2+ online:
https://riptutorial.com/it/angularjs/topic/9942/migrazione-verso-angular-2plus

Capitolo 30: moduli

Examples

moduli

Il modulo funge da contenitore di diverse parti della tua app come controller, servizi, filtri, direttive, ecc. I moduli possono essere referenziati da altri moduli tramite il meccanismo di iniezione delle dipendenze di Angular.

Creare un modulo:

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

Array `[]` passato nell'esempio sopra è l' *elenco dei moduli* `app` dipende da, se non ci sono dipendenze allora passiamo a Empty Array cioè `[]` .

Iniezione di un modulo come dipendenza di un altro modulo:

```
angular.module('app', [
    'app.auth',
    'app.dashboard'
]);
```

Fare riferimento a un modulo:

```
angular
    .module('app');
```

moduli

Il modulo è un contenitore per varie parti delle applicazioni: controller, servizi, filtri, direttive, ecc.

Perché usare i moduli
La maggior parte delle applicazioni ha un metodo principale che istanzia e collega tra loro le diverse parti dell'applicazione.
Le app angolari non hanno il metodo principale.
Ma in AngularJs il processo dichiarativo è facile da capire e si può impacchettare il codice come moduli riutilizzabili.
I moduli possono essere caricati in qualsiasi ordine poiché i moduli ritardano l'esecuzione.

dichiarare un modulo

```
var app = angular.module('myApp', []);
// Empty array is list of modules myApp is depends on.
// if there are any required dependancies,
```

```
// then you can add in module, Like ['ngAnimate']

app.controller('myController', function() {

  // write your business logic here
});
```

Caricamento del modulo e dipendenze

1. Blocchi di configurazione: - vengono eseguiti durante la fase di provider e configurazione.

   ```
   angular.module('myModule', []).
   config(function(injectables) {
     // here you can only inject providers in to config blocks.
   });
   ```

2. Esegui blocchi: - vengono eseguiti dopo la creazione dell'iniettore e vengono utilizzati per avviare l'applicazione.

   ```
   angular.module('myModule', []).
   run(function(injectables) {
     // here you can only inject instances in to config blocks.
   });
   ```

Leggi moduli online: https://riptutorial.com/it/angularjs/topic/844/moduli

Capitolo 31: MVC angolare

introduzione

In **AngularJS** il pattern **MVC** è implementato in JavaScript e HTML. La vista è definita in HTML, mentre il modello e il controller sono implementati in JavaScript. Ci sono diversi modi in cui questi componenti possono essere messi insieme in AngularJS ma la forma più semplice inizia con la vista.

Examples

La vista statica con controller

demo mvc

Ciao mondo

Definizione della funzione del controller

```
var indexController = myApp.controller("indexController", function ($scope) {
    // Application logic goes here
});
```

Aggiunta di informazioni al modello

```
var indexController = myApp.controller("indexController", function ($scope) {
    // controller logic goes here
    $scope.message = "Hello Hacking World"
});
```

Leggi MVC angolare online: https://riptutorial.com/it/angularjs/topic/8667/mvc-angolare

Capitolo 32: ng-repeat

introduzione

La direttiva `ngRepeat` crea un'istanza di un modello una volta per elemento da una raccolta. La raccolta deve essere una matrice o un oggetto. Ogni istanza del modello ottiene il proprio ambito, dove la variabile del ciclo data è impostata sull'elemento della raccolta corrente e `$index` è impostato sull'indice o sulla chiave dell'elemento.

Sintassi

- `<element ng-repeat="expression"></element>`
- `<div ng-repeat="(key, value) in myObj">...</div>`
- `<div ng-repeat="variable in expression">...</div>`

Parametri

Variabile	Dettagli
`$index`	offset dell'iteratore del *numero* dell'elemento ripetuto (0..length-1)
`$first`	*booleano* vero se l'elemento ripetuto è il primo nell'iteratore.
`$middle`	*booleano* vero se l'elemento ripetuto si trova tra il primo e l'ultimo nell'iteratore.
`$last`	*booleano* vero se l'elemento ripetuto è l'ultimo nell'iteratore.
`$even`	*booleano* vero se la posizione iteratore `$index` è pari (altrimenti false).
`$odd`	*booleano* true se la posizione iterator `$index` è dispari (altrimenti false).

Osservazioni

AngularJS fornisce questi parametri come variabili speciali disponibili nell'espressione ng-repeat e ovunque all'interno del tag HTML su cui vive la ng-repeat.

Examples

Iterare sulle proprietà dell'oggetto

```
<div ng-repeat="(key, value) in myObj"> ... </div>
```

Per esempio

```
<div ng-repeat="n in [42, 42, 43, 43]">
  {{n}}
</div>
```

Tracciamento e duplicati

`ngRepeat` utilizza $ watchCollection per rilevare le modifiche nella raccolta. Quando si verifica una modifica, `ngRepeat` esegue quindi le modifiche corrispondenti al DOM:

- Quando viene aggiunto un elemento, una nuova istanza del modello viene aggiunta al DOM.
- Quando un elemento viene rimosso, l'istanza del modello viene rimossa dal DOM.
- Quando gli articoli vengono riordinati, i rispettivi modelli vengono riordinati nel DOM.

duplicati

- `track by` per qualsiasi elenco che possa includere valori duplicati.
- `track by` accelera anche le modifiche delle liste in modo significativo.
- Se in questo caso non si usa la funzione `track by` , si ottiene l'errore: `[ngRepeat:dupes]`

```
$scope.numbers = ['1','1','2','3','4'];

<ul>
  <li ng-repeat="n in numbers track by $index">
    {{n}}
  </li>
</ul>
```

ng-repeat-start + ng-repeat-end

AngularJS 1.2 `ng-repeat` gestisce più elementi con `ng-repeat-start` e `ng-repeat-end` :

```
// table items
$scope.tableItems = [
    {
        row1: 'Item 1: Row 1',
        row2: 'Item 1: Row 2'
    },
    {
        row1: 'Item 2: Row 1',
        row2: 'Item 2: Row 2'
    }
];

// template
<table>
    <th>
        <td>Items</td>
    </th>
    <tr ng-repeat-start="item in tableItems">
        <td ng-bind="item.row1"></td>
    </tr>
    <tr ng-repeat-end>
        <td ng-bind="item.row2"></td>
    </tr>
```

```
</table>
```

Produzione:

Elementi
Elemento 1: riga 1
Elemento 1: riga 2
Articolo 2: Riga 1
Elemento 2: riga 2

Leggi ng-repeat online: https://riptutorial.com/it/angularjs/topic/8118/ng-repeat

Capitolo 33: ng-style

introduzione

La direttiva 'ngStyle' consente di impostare lo stile CSS su un elemento HTML in modo condizionale. Proprio come utilizzare l'attributo di *stile* sull'elemento HTML in progetti non AngularJS, possiamo usare `ng-style` in angularjs per applicare stili basati su alcune condizioni booleane.

Sintassi

- `<ANY ng-style="expression"></ANY >`
- `<ANY class="ng-style: expression;"> ... </ANY>`

Examples

Uso di ng-style

Sotto l'esempio cambia l'opacità dell'immagine in base al parametro "stato".

```
<img class="img-responsive"  ng-src="{{imagesrc}}"
    ng-style="{'opacity' : (status == 2) ? 1 : 0.5}">
```

Leggi ng-style online: https://riptutorial.com/it/angularjs/topic/8773/ng-style

Capitolo 34: ng-view

introduzione

ng-view è una direttiva in-build che angular utilizza come contenitore per passare da una vista all'altra. {info} ngRoute non fa più parte del file angular.js di base, quindi dovrai includere il file angular-route.js dopo il file javascript angolare di base. Possiamo configurare un percorso usando la funzione "quando" del $ routeProvider. Dobbiamo prima specificare il percorso, quindi in un secondo parametro fornire un oggetto con una proprietà templateUrl e una proprietà controller.

Examples

ng-view

`ng-view` è una direttiva usata con `$route` per rendere una vista parziale nel layout della pagina principale. Qui in questo esempio, Index.html è il nostro file principale e quando l'utente atterra su "/" indirizza il templateURL home.html sarà reso in Index.html dove viene menzionata `ng-view` .

```
angular.module('ngApp', ['ngRoute'])

.config(function($routeProvider){
  $routeProvider.when("/",
    {
      templateUrl: "home.html",
      controller: "homeCtrl"
    }
  );
});

angular.module('ngApp').controller('homeCtrl',['$scope', function($scope) {
  $scope.welcome= "Welcome to stackoverflow!";
}]);

//Index.html
<body ng-app="ngApp">
    <div ng-view></div>
</body>

//Home Template URL or home.html
<div><h2>{{welcome}}</h2></div>
```

Navigazione di registrazione

1. Inseriamo il modulo nell'applicazione

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

2. ora usiamo $ routeProvider da "ngRoute"

```
    Registration.config(function($routeProvider) {

});
```

3. infine, integrando il percorso, definiamo l'instradamento "/ add" all'applicazione nel caso in cui l'applicazione ottenga "/ add" si devia su regi.htm

```
 Registration.config(function($routeProvider) {
    $routeProvider
    .when("/add", {
        templateUrl : "regi.htm"
    })
});
```

Leggi ng-view online: https://riptutorial.com/it/angularjs/topic/8833/ng-view

Capitolo 35: Opzioni di collegamenti AngularJS (`=`, `@`, `&` ecc.)

Osservazioni

Usa questo plunker per giocare con degli esempi.

Examples

@ rilegatura unidirezionale, associazione degli attributi.

Passa in un valore letterale (non un oggetto), come una stringa o un numero.

L'ambito figlio ottiene il proprio valore, se aggiorna il valore, l'ambito genitore ha il suo valore precedente (l'ambito figlio non può modificare il valore dell'ambito parentale). Quando viene modificato il valore dell'ambito genitore, anche il valore dell'ambito secondario verrà modificato. Tutte le interpolazioni vengono visualizzate ogni volta durante la digitazione, non solo sulla creazione di direttive.

```
<one-way text="Simple text." <!-- 'Simple text.' -->
         simple-value="123" <!-- '123' Note, is actually a string object. -->
         interpolated-value="{{parentScopeValue}}" <!-- Some value from parent scope. You
can't change parent scope value, only child scope value. Note, is actually a string object. --
>
         interpolated-function-value="{{parentScopeFunction()}}" <!-- Executes parent scope
function and takes a value. -->

         <!-- Unexpected usage. -->
         object-item="{{objectItem}}" <!-- Converts object|date to string. Result might be:
'{"a":5,"b":"text"}'. -->
         function-item="{{parentScopeFunction}}"> <!-- Will be an empty string. -->
</one-way>
```

= rilegatura a doppio senso.

Passando un valore per riferimento, si desidera condividere il valore tra entrambi gli ambiti e manipolarli da entrambi gli ambiti. Non si dovrebbe usare {{...}} per l'interpolazione.

```
<two-way text="'Simple text.'" <!-- 'Simple text.' -->
         simple-value="123" <!-- 123 Note, is actually a number now. -->
         interpolated-value="parentScopeValue" <!-- Some value from parent scope. You may
change it in one scope and have updated value in another. -->
         object-item="objectItem" <!-- Some object from parent scope. You may change object
properties in one scope and have updated properties in another. -->

         <!-- Unexpected usage. -->
         interpolated-function-value="parentScopeFunction()" <!-- Will raise an error. -->
         function-item="incrementInterpolated"> <!-- Pass the function by reference and you
may use it in child scope. -->
```

```
</two-way>
```

Passare la funzione per riferimento è una cattiva idea: consentire all'oscilloscopio di cambiare la definizione di una funzione e verranno creati due osservatori non necessari, è necessario ridurre al minimo il numero di osservatori.

& binding di funzioni, binding di espressioni.

Passa un metodo in una direttiva. Fornisce un modo per eseguire un'espressione nel contesto dell'ambito principale. Il metodo verrà eseguito nell'ambito del genitore, è possibile passare alcuni parametri dall'ambito figlio. Non si dovrebbe usare {{...}} per l'interpolazione. Quando si utilizza & in una direttiva, viene generata una funzione che restituisce il valore dell'espressione valutata rispetto all'ambito principale (non lo stesso di = in cui si passa solo un riferimento).

```
<expression-binding interpolated-function-value="incrementInterpolated(param)" <!--
interpolatedFunctionValue({param: 'Hey'}) will call passed function with an argument. -->
                    function-item="incrementInterpolated" <!-- functionItem({param: 'Hey'})()
will call passed function, but with no possibility set up a parameter. -->
                    text="'Simple text.'"  <!-- text() == 'Simple text.'-->
                    simple-value="123" <!-- simpleValue() == 123 -->
                    interpolated-value="parentScopeValue" <!-- interpolatedValue() == Some
value from parent scope. -->
                    object-item="objectItem"> <!-- objectItem() == Object item from parent
scope. -->
</expression-binding>
```

Tutti i parametri saranno inclusi in funzioni.

Associazione disponibile tramite un semplice campione

```
angular.component("SampleComponent", {
  bindings: {
     title: '@',
     movies: '<',
     reservation: "=",
     processReservation: "&"
  }
});
```

Qui abbiamo tutti gli elementi vincolanti.

@ indica che abbiamo bisogno di un **binding** molto **semplice** , dall'ambito padre all'ambito figli, senza alcun osservatore, in alcun modo. Ogni aggiornamento nello scope genitore rimarrebbe nell'ambito genitore e qualsiasi aggiornamento sull'ambito secondario non verrebbe comunicato all'ambito genitore.

< indica un **legame unidirezionale** . Gli aggiornamenti nell'ambito genitore verrebbero propagati all'ambito figli, ma qualsiasi aggiornamento nell'ambito figli non verrebbe applicato all'ambito principale.

= è già noto come rilegatura a doppio senso. Ogni aggiornamento sull'ambito genitore verrebbe

applicato a quelli secondari e ogni aggiornamento secondario verrebbe applicato all'ambito principale.

& viene ora utilizzato per un binding di output. Secondo la documentazione del componente, dovrebbe essere usato per fare riferimento al metodo dello scope genitore. Invece di manipolare l'ambito figli, basta chiamare il metodo genitore con i dati aggiornati!

Bind attributo opzionale

```
bindings: {
   mandatory: '='
   optional: '=?',
   foo: '=?bar'
}
```

Gli attributi facoltativi dovrebbero essere contrassegnati con un punto interrogativo: `=?` o `=?bar` . È protezione per l'eccezione `($compile:nonassign)` .

Leggi Opzioni di collegamenti AngularJS (`=`, `@`, `&` ecc.) online: https://riptutorial.com/it/angularjs/topic/6149/opzioni-di-collegamenti-angularjs----------amp---ecc--

Capitolo 36: Preparati per la produzione - Grunt

Examples

Visualizza il preloading

Quando viene richiesta la prima visualizzazione temporale, normalmente Angular richiede `XHR` per ottenere quella vista. Per i progetti di medie dimensioni, il numero di visualizzazioni può essere significativo e può rallentare la reattività dell'applicazione.

È **buona norma precaricare** tutte le viste contemporaneamente per progetti di piccole e medie dimensioni. Per i progetti più grandi è buona norma aggregarli anche in alcuni blocchi significativi, ma alcuni altri metodi possono essere utili per suddividere il carico. Per automatizzare questa attività è utile utilizzare le attività Grunt o Gulp.

Per precaricare le viste, possiamo usare `$templateCache` object. Questo è un oggetto, dove angolare memorizza ogni vista ricevuta dal server.

È possibile utilizzare il modulo `html2js` , che convertirà tutte le nostre viste in un modulo - file js. Quindi avremo bisogno di iniettare quel modulo nella nostra applicazione e basta.

Per creare un file concatenato di tutte le viste possiamo usare questa attività

```
module.exports = function (grunt) {
 //set up the location of your views here
 var viewLocation = ['app/views/**.html'];

 grunt.initConfig({
        pkg: require('./package.json'),
            //section that sets up the settings for concatenation of the html files into one
file
            html2js: {
                        options: {
                            base: '',
                            module: 'app.templates', //new module name
                            singleModule: true,
                            useStrict: true,
                            htmlmin: {
                                collapseBooleanAttributes: true,
                                collapseWhitespace: true
                            }
                        },
                        main: {
                            src: viewLocation,
                            dest: 'build/app.templates.js'
                        }
                    },
            //this section is watching for changes in view files, and if there was a change,
it will regenerate the production file. This task can be handy during development.
            watch: {
```

```
                views:{
                    files: viewLocation,
                    tasks: ['buildHTML']
                },
            }
        });

        //to automatically generate one view file
        grunt.loadNpmTasks('grunt-html2js');

        //to watch for changes and if the file has been changed, regenerate the file
        grunt.loadNpmTasks('grunt-contrib-watch');

        //just a task with friendly name to reference in watch
        grunt.registerTask('buildHTML', ['html2js']);
};
```

Per utilizzare questo metodo di concatinazione, è necessario apportare 2 modifiche: nel file `index.html` è necessario fare riferimento al file di visualizzazione concatenato

```
<script src="build/app.templates.js"></script>
```

Nel file, dove stai dichiarando la tua app, devi iniettare la dipendenza

```
angular.module('app', ['app.templates'])
```

Se si utilizzano router popolari come `ui-router` , non ci sono cambiamenti nel modo in cui si fa riferimento ai modelli

```
    .state('home', {
        url: '/home',
        views: {
            "@": {
                controller: 'homeController',
                //this will be picked up from $templateCache
                templateUrl: 'app/views/home.html'
            },
        }

    })
```

Ottimizzazione dello script

È **buona norma combinare i file JS** e ridurli al minimo. Per progetti più grandi potrebbero esserci centinaia di file JS e aggiunge latenza inutile per caricare ogni file separatamente dal server.

Per la minimizzazione angolare è necessario avere tutte le funzioni annotate. Quello in necessario per la dipendenza da una dipendenza angolare è una corretta minifrizione. (Durante la minificazione, i nomi delle funzioni e le variabili verranno rinominati e interromperà l'iniezione della dipendenza se non verranno intraprese ulteriori azioni.)

Durante la minificaiton `$scope` e `myService` variabili saranno sostituiti da altri valori. L'iniezione di dipendenza angolare funziona in base al nome, pertanto questi nomi non dovrebbero cambiare

```
.controller('myController', function($scope, myService){
})
```

Angolare comprenderà la notazione dell'array, poiché la minificazione non sostituirà i valori letterali stringa.

```
.controller('myController', ['$scope','myService', function($scope, myService){
}])
```

- In primo luogo concatideremo tutti i file end-to-end.
- In secondo luogo useremo il modulo `ng-annotate` , che preparerà il codice per la minimizzazione
- Infine applicheremo il modulo `uglify` .

module.exports = function (grunt) {// imposta qui il percorso degli script per riutilizzarlo nel codice var scriptLocation = ['app / scripts / *. js'];

```
 grunt.initConfig({
        pkg: require('./package.json'),
            //add necessary annotations for safe minification
         ngAnnotate: {
            angular: {
                src: ['staging/concatenated.js'],
                dest: 'staging/anotated.js'
            }
        },
        //combines all the files into one file
        concat: {
                js: {
                    src: scriptLocation,
                    dest: 'staging/concatenated.js'
                }
            },
        //final uglifying
        uglify: {
            options: {
                report: 'min',
                mangle: false,
                sourceMap:true
            },
            my_target: {
                files: {
                    'build/app.min.js': ['staging/anotated.js']
                }
            }
        },

        //this section is watching for changes in JS files, and if there was a change, it will
regenerate the production file. You can choose not to do it, but I like to keep concatenated
version up to date
        watch: {
            scripts: {
                files: scriptLocation,
                tasks: ['buildJS']
            }
        }
```

```
});

    //module to make files less readable
    grunt.loadNpmTasks('grunt-contrib-uglify');

    //mdule to concatenate files together
    grunt.loadNpmTasks('grunt-contrib-concat');

    //module to make angularJS files ready for minification
    grunt.loadNpmTasks('grunt-ng-annotate');

    //to watch for changes and if the file has been changed, regenerate the file
    grunt.loadNpmTasks('grunt-contrib-watch');

    //task that sequentially executes all steps to prepare JS file for production
    //concatinate all JS files
    //annotate JS file (prepare for minification
    //uglify file
     grunt.registerTask('buildJS', ['concat:js', 'ngAnnotate', 'uglify']);
};
```

Leggi Preparati per la produzione - Grunt online:
https://riptutorial.com/it/angularjs/topic/4434/preparati-per-la-produzione---grunt

Capitolo 37: Profilazione e prestazioni

Examples

7 miglioramenti delle prestazioni semplici

1) Usa ng-repeat con parsimonia

L'uso di `ng-repeat` nelle viste comporta generalmente scarse prestazioni, in particolare quando ci sono `ng-repeat` annidate.

Questo è super lento!

```
<div ng-repeat="user in userCollection">
  <div ng-repeat="details in user">
    {{details}}
  </div>
</div>
```

Cerca di evitare il più possibile le ripetizioni annidate. Un modo per migliorare le prestazioni di `ng-repeat` è utilizzare la `track by $index` (o qualche altro campo id). Di default, `ng-repeat` traccia l'intero oggetto. Con `track by` , Angular controlla l'oggetto solo dall'indice `$index` o dall'id oggetto.

```
<div ng-repeat="user in userCollection track by $index">
  {{user.data}}
</div>
```

Utilizza altri approcci come impaginazione , scroll virtuali , scroll infiniti o limitTo: inizia quando possibile per evitare iterazioni su raccolte di grandi dimensioni.

---

2) legare una volta

Angolare ha l'associazione dati bidirezionale. Viene fornito con un costo di essere lento se usato troppo.

Prestazioni più lente

```
<!-- Default data binding has a performance cost -->
<div>{{ my.data }}</div>
```

Prestazioni più veloci (AngularJS> = 1.3)

```
<!-- Bind once is much faster -->
<div>{{ ::my.data }}</div>

<div ng-bind="::my.data"></div>
```

```
<!-- Use single binding notation in ng-repeat where only list display is needed  -->
<div ng-repeat="user in ::userCollection">
  {{::user.data}}
</div>
```

L'uso della notazione "bind once" indica ad Angular di attendere che il valore si stabilizzi dopo la prima serie di cicli digest. Angular utilizzerà quel valore nel DOM, quindi rimuoverà tutti gli osservatori in modo che diventi un valore statico e non sia più legato al modello.

Il `{{}}` è molto più lento.

Questo `ng-bind` è una direttiva e posizionerà un osservatore sulla variabile passata. Quindi il `ng-bind` si applicherà solo quando il valore passato cambia effettivamente.

Le parentesi d'altra parte saranno sporche controllate e aggiornate in ogni `$digest` , anche se non è necessario.

---

3) Le funzioni e i filtri dell'oscilloscopio richiedono tempo

AngularJS ha un ciclo digest. Tutte le tue funzioni sono in una vista e i filtri vengono eseguiti ogni volta che viene eseguito il ciclo di digest. Il ciclo digest verrà eseguito ogni volta che il modello viene aggiornato e può rallentare la tua app (il filtro può essere colpito più volte prima che la pagina venga caricata).

Evita questo:

```
<div ng-controller="bigCalulations as calc">
  <p>{{calc.calculateMe()}}</p>
  <p>{{calc.data | heavyFilter}}</p>
</div>
```

Approccio migliore

```
<div ng-controller="bigCalulations as calc">
  <p>{{calc.preCalculatedValue}}</p>
  <p>{{calc.data | lightFilter}}</p>
</div>
```

Dove può essere il controller:

```
app.controller('bigCalulations', function(valueService) {
    // bad, because this is called in every digest loop
    this.calculateMe = function() {
        var t = 0;
        for(i = 0; i < 1000; i++) {
            t += i;
        }
        return t;
    }
    // good, because this is executed just once and logic is separated in service to    keep
```

```
the controller light
    this.preCalulatedValue = valueService.valueCalculation(); // returns 499500
});
```

---

4 spettatori

Gli spettatori abbandonano tremendamente le prestazioni. Con più osservatori, il ciclo digest richiederà più tempo e l'interfaccia utente rallenterà. Se l'osservatore rileva il cambiamento, avvia il ciclo digest e ricrea la vista.

Esistono tre modi per eseguire la sorveglianza manuale per i cambiamenti variabili in Angolare.

`$watch()` - guarda per le variazioni di valore

`$watchCollection()` - osserva i cambiamenti nella collezione (guarda più del normale `$watch`)

`$watch(..., true)` - **Evita** il più possibile, eseguirà "deep watch" e `watchCollection` le prestazioni (guarda più di `watchCollection`)

Nota che se stai legando le variabili nella vista stai creando nuovi orologi: usa `{{::variable}}` per impedire la creazione di un orologio, specialmente nei loop.

Di conseguenza è necessario tenere traccia di quanti osservatori stai usando. Puoi contare gli osservatori con questo script (credito a @Words Like Jared Number of watchers)

```
(function() {
    var root = angular.element(document.getElementsByTagName('body')),
        watchers = [],
        f = function(element) {
        angular.forEach(['$scope', '$isolateScope'], function(scopeProperty) {
            if(element.data() && element.data().hasOwnProperty(scopeProperty)) {
                angular.forEach(element.data()[scopeProperty].$$watchers, function(watcher) {
                watchers.push(watcher);
                });
            }
        });

        angular.forEach(element.children(), function(childElement) {
            f(angular.element(childElement));
        });
    };

    f(root);

    // Remove duplicate watchers
    var watchersWithoutDuplicates = [];
    angular.forEach(watchers, function(item) {
        if(watchersWithoutDuplicates.indexOf(item) < 0) {
            watchersWithoutDuplicates.push(item);
        }
    });
    console.log(watchersWithoutDuplicates.length);
})();
```

---

5) ng-if / ng-show

Queste funzioni sono molto simili nel comportamento. `ng-if` rimuove gli elementi dal DOM mentre `ng-show` nasconde solo gli elementi ma mantiene tutti i gestori. Se hai parti del codice che non vuoi mostrare, usa `ng-if` .

Dipende dal tipo di utilizzo, ma spesso uno è più adatto dell'altro.

- Se l'elemento non è necessario, utilizzare `ng-if`
- Per attivare / disattivare rapidamente, utilizzare `ng-show/ng-hide`

```
<div ng-repeat="user in userCollection">
  <p ng-if="user.hasTreeLegs">I am special<!-- some complicated DOM --></p>
  <p ng-show="user.hasSubscribed">I am awesome<!-- switch this setting on and off --></p>
</div>
```

In caso di dubbio, usa `ng-if` e prova!

---

6) Disabilitare il debug

Per impostazione predefinita, le direttive e gli ambiti di bind lasciano classi e markup aggiuntivi nel codice per aiutare con vari strumenti di debug. Disabilitare questa opzione significa che non esegui più il rendering di questi vari elementi durante il ciclo di digest.

```
angular.module('exampleApp', []).config(['$compileProvider', function ($compileProvider) {
    $compileProvider.debugInfoEnabled(false);
}]);
```

---

7) Usa l'iniezione delle dipendenze per esporre le tue risorse

Dependency Injection è un pattern di progettazione software in cui a un oggetto vengono assegnate le sue dipendenze, piuttosto che l'oggetto che le crea da sé. Si tratta di rimuovere le dipendenze hard-coded e rendere possibile cambiarle quando necessario.

Potreste chiedervi il costo delle prestazioni associato a tale analisi delle stringhe di tutte le funzioni iniettabili. Angular si occupa di ciò memorizzando nella cache la proprietà $ inject dopo la prima volta. Quindi questo non accade ogni volta che una funzione deve essere invocata.

SUGGERIMENTO PRO: se stai cercando l'approccio con le migliori prestazioni, vai con l'approccio annotazione delle proprietà $ inject. Questo approccio evita completamente l'analisi della definizione della funzione poiché questa logica è racchiusa all'interno del seguente controllo nella funzione annotate: if (! ($ Inject = fn. $ Inject)). Se $ inject è già disponibile, non è richiesta alcuna analisi!

```
var app = angular.module('DemoApp', []);
```

```
var DemoController = function (s, h) {
    h.get('https://api.github.com/users/angular/repos').success(function (repos) {
        s.repos = repos;
    });
}
// $inject property annotation
DemoController['$inject'] = ['$scope', '$http'];

app.controller('DemoController', DemoController);
```

PRO TIP 2: è possibile aggiungere una direttiva `ng-strict-di` sullo stesso elemento di `ng-app` per attivare la modalità DI rigorosa che genera un errore ogni volta che un servizio tenta di utilizzare annotazioni implicite. Esempio:

```
<html ng-app="DemoApp" ng-strict-di>
```

O se usi il bootstrap manuale:

```
angular.bootstrap(document, ['DemoApp'], {
    strictDi: true
});
```

Bind Once

Angular ha la reputazione di avere un eccezionale binding di dati bidirezionale. Per impostazione predefinita, Angular sincronizza continuamente i valori associati tra il modello e i componenti di visualizzazione in qualsiasi momento in cui i dati cambiano nel componente del modello o della vista.

Questo ha un costo di essere un po 'lento se usato troppo. Questo avrà un successo in termini di prestazioni più grandi:

Cattiva performance: `{{my.data}}`

Aggiungi due punti `::` prima del nome della variabile per utilizzare l'associazione una tantum. In questo caso, il valore viene aggiornato solo una volta definito my.data. Stai indicando esplicitamente di non guardare le modifiche dei dati. Angular non eseguirà alcun controllo del valore, risultante con un numero inferiore di espressioni valutate in ciascun ciclo di digestione.

Buoni esempi di prestazioni che utilizzano l'associazione una tantum

```
{{::my.data}}
<span ng-bind="::my.data"></span>
<span ng-if="::my.data"></span>
<span ng-repeat="item in ::my.data">{{item}}</span>
<span ng-class="::{ 'my-class': my.data }"></div>
```

Nota: questo rimuove l'associazione bidirezionale dei dati per `my.data` , quindi ogni volta che questo campo cambia nell'applicazione, lo stesso non si rifletterà automaticamente nella vista. Quindi **usalo solo per valori che non cambieranno per tutta la durata della tua applicazione** .

Funzioni e filtri dell'ambito

AngularJS ha un ciclo di digest e tutte le tue funzioni in una vista e i filtri vengono eseguiti ogni volta che viene eseguito il ciclo di digest. Il ciclo digest verrà eseguito ogni volta che il modello viene aggiornato e può rallentare la tua app (il filtro può essere colpito più volte, prima che la pagina venga caricata).

Dovresti evitare questo:

```
<div ng-controller="bigCalulations as calc">
  <p>{{calc.calculateMe()}}</p>
  <p>{{calc.data | heavyFilter}}</p>
</div>
```

Approccio migliore

```
<div ng-controller="bigCalulations as calc">
  <p>{{calc.preCalculatedValue}}</p>
  <p>{{calc.data | lightFilter}}</p>
</div>
```

Dove il campione del controller è:

```
.controller("bigCalulations", function(valueService) {
  // bad, because this is called in every digest loop
  this.calculateMe = function() {
    var t = 0;
    for(i = 0; i < 1000; i++) {
      t = t + i;
    }
    return t;
  }
  //good, because it is executed just once and logic is separated in service to    keep the
controller light
  this.preCalulatedValue = valueService.caluclateSumm(); // returns 499500
});
```

osservatori

Gli osservatori necessari per guardare un certo valore e rilevare che questo valore è cambiato.

Dopo la chiamata `$watch()` o `$watchCollection` nuovo watcher aggiungere alla raccolta interna del watcher nell'ambito corrente.

Quindi, che cos'è l'osservatore?

Watcher è una funzione semplice, che viene richiamata in ogni ciclo di digest e restituisce un valore. Angolare controlla il valore restituito, se non è uguale a come era nella chiamata precedente - una richiamata che è stata passata nel secondo parametro alla funzione `$watch()` o `$watchCollection` verrà eseguita.

```
(function() {
  angular.module("app", []).controller("ctrl", function($scope) {
    $scope.value = 10;
    $scope.$watch(
      function() { return $scope.value; },
      function() { console.log("value changed"); }
    );
  }
})();
```

Gli osservatori sono assassini di prestazioni. Più osservatori hai, più tempo impiegano a fare un ciclo digest, l'interfaccia utente più lenta. Se un osservatore rileva delle modifiche, avvia il ciclo digest (ricalcolo su tutto lo schermo)

Esistono tre modi per eseguire la visualizzazione manuale delle modifiche variabili in Angolare.

`$watch()` - guarda solo per cambiamenti di valore

`$watchCollection()` - osserva i cambiamenti nella collezione (guarda più del normale $ watch)

`$watch(..., true)` - **Evita** il più possibile, eseguirà "deep watch" e ucciderà la performance (guarda più di watchCollection)

Nota che se stai legando le variabili nella vista, stai creando nuovi osservatori - usa `{{::variable}}` non creare watcher, specialmente nei loop

Di conseguenza è necessario tenere traccia di quanti osservatori stai usando. Puoi contare gli osservatori con questo script (credito a @Words Like Jared - Come contare il numero totale di orologi su una pagina?

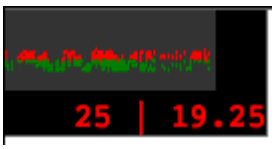
```
(function() {
  var root = angular.element(document.getElementsByTagName("body")),
      watchers = [];

  var f = function(element) {

    angular.forEach(["$scope", "$isolateScope"], function(scopeProperty) {
      if(element.data() && element.data().hasOwnProperty(scopeProperty)) {
        angular.forEach(element.data()[scopeProperty].$$watchers, function(watcher) {
          watchers.push(watcher);
        });
      }
    });

    angular.forEach(element.children(), function(childElement) {
      f(angular.element(childElement));
    });

  };

  f(root);

  // Remove duplicate watchers
  var watchersWithoutDuplicates = [];
  angular.forEach(watchers, function(item) {
```

```
    if(watchersWithoutDuplicates.indexOf(item) < 0) {
      watchersWithoutDuplicates.push(item);
    }
  });

  console.log(watchersWithoutDuplicates.length);

})();
```

Se non vuoi creare il tuo script, c'è una utility open source chiamata ng-stats che usa un grafico in tempo reale incorporato nella pagina per darti un'idea del numero di orologi che Angular sta gestendo, così come frequenza e durata dei cicli digest nel tempo. L'utilità espone una funzione globale denominata `showAngularStats` che è possibile chiamare per configurare il modo in cui si desidera far funzionare il grafico.

```
showAngularStats({
  "position": "topleft",
  "digestTimeThreshold": 16,
  "autoload": true,
  "logDigest": true,
  "logWatches": true
});
```

Il codice di esempio sopra mostra automaticamente il seguente grafico sulla pagina (demo interattiva).



ng-if vs ng-show

Queste funzioni sono molto simili nel comportamento. La differenza è che `ng-if` rimuove gli elementi dal DOM. Se ci sono grandi parti del codice che non verranno mostrate, allora `ng-if` è la strada da percorrere. `ng-show` nasconderà solo gli elementi ma manterrà tutti i gestori.

ng-se

La direttiva ngIf rimuove o ricrea una porzione dell'albero DOM in base a un'espressione. Se l'espressione assegnata a ngIf restituisce un valore falso, l'elemento viene rimosso dal DOM, altrimenti un clone dell'elemento viene reinserito nel DOM.

ng-spettacolo

La direttiva ngShow mostra o nasconde l'elemento HTML dato in base all'espressione fornita all'attributo ngShow. L'elemento è mostrato o nascosto rimuovendo o aggiungendo la classe CSS ng-hide sull'elemento.

Esempio

```
<div ng-repeat="user in userCollection">
  <p ng-if="user.hasTreeLegs">I am special
    <!-- some complicated DOM -->
  </p>
  <p ng-show="user.hasSubscribed">I am aweosme
    <!-- switch this setting on and off -->
  </p>
</div>
```

Conclusione

Dipende dal tipo di utilizzo, ma spesso uno è più adatto dell'altro (ad esempio, se il 95% delle volte l'elemento non è necessario, utilizzare `ng-if` ; se è necessario attivare o disattivare la visibilità dell'elemento DOM, utilizzare `ng-show`).

In caso di dubbio, usa `ng-if` e prova!

Nota : `ng-if` crea un nuovo ambito isolato, mentre `ng-show` e `ng-hide` no. Usa `$parent.property` se la proprietà dell'ambito genitore non è direttamente accessibile in essa.

Rimbalza il tuo modello

```
<div ng-controller="ExampleController">
    <form name="userForm">
        Name:
        <input type="text" name="userName"
           ng-model="user.name"
           ng-model-options="{ debounce: 1000 }" />
            <button ng-click="userForm.userName.$rollbackViewValue();
user.name=''">Clear</button><br />
    </form>
    <pre>user.name = </pre>
</div>
```

Nell'esempio sopra stiamo impostando un valore di antirimbalzo di 1000 millisecondi che è 1 secondo. Questo è un ritardo considerevole, ma impedirà l'input da ripetutamente `ng-model` thrash con molti cicli `$digest` .

Utilizzando il debounce sui campi di input e in qualsiasi altro punto in cui non è richiesto un aggiornamento istantaneo, è possibile aumentare notevolmente le prestazioni delle app Angular. Non solo puoi ritardare il tempo, ma puoi anche ritardare quando l'azione viene attivata. Se non si desidera aggiornare il modello ng su ogni sequenza di tasti, è anche possibile eseguire l'aggiornamento su sfocatura.

Cancellare sempre gli ascoltatori registrati su altri ambiti diversi dall'ambito corrente

È sempre necessario annullare la registrazione degli ambiti diversi dall'ambito corrente, come illustrato di seguito:

```
//always deregister these
$rootScope.$on(...);
$scope.$parent.$on(...);
```

Non è necessario annullare la registrazione dei listener in base allo scope corrente, poiché angular si prenderà cura di esso:

```
//no need to deregister this
$scope.$on(...);
```

`$rootScope.$on` ascoltatori rimarrà in memoria se `$rootScope.$on` un altro controller. Ciò creerà una perdita di memoria se il controller non rientra nell'ambito.

non

```
angular.module('app').controller('badExampleController', badExample);
badExample.$inject = ['$scope', '$rootScope'];

function badExample($scope, $rootScope) {
    $rootScope.$on('post:created', function postCreated(event, data) {});
}
```

Fare

```
angular.module('app').controller('goodExampleController', goodExample);
goodExample.$inject = ['$scope', '$rootScope'];

function goodExample($scope, $rootScope) {
    var deregister = $rootScope.$on('post:created', function postCreated(event, data) {});

    $scope.$on('$destroy', function destroyScope() {
        deregister();
    });
}
```

Leggi Profilazione e prestazioni online: https://riptutorial.com/it/angularjs/topic/1921/profilazione-e-prestazioni

Capitolo 38: Profilo delle prestazioni

Examples

Tutto sul profilo

Cos'è il profiling?

Per definizione, il profilo è una forma di analisi dinamica del programma che misura, ad esempio, lo spazio (memoria) o la complessità temporale di un programma, l'uso di particolari istruzioni, o la frequenza e la durata delle chiamate di funzione.

Perché è necessario?

La profilazione è importante perché non è possibile ottimizzare in modo efficace fino a quando non si sa che cosa sta facendo il proprio programma per la maggior parte del tempo. Senza misurare il tempo di esecuzione del programma (profilazione), non saprai se lo hai effettivamente migliorato.

Strumenti e tecniche:

1. Strumenti di sviluppo integrati di Chrome

   Questo include un set completo di strumenti da utilizzare per la profilazione. Puoi approfondire la ricerca di colli di bottiglia nel tuo file javascript, file css, animazioni, consumo di CPU, perdite di memoria, rete, sicurezza, ecc.

   Crea una registrazione della Timeline e cerca eventi di Evaluate Script lunghi e sospetti. Se ne trovi uno, puoi abilitare JS Profiler e rifare la registrazione per ottenere informazioni più dettagliate su esattamente quali funzioni JS sono state chiamate e per quanto tempo ciascuna ha preso. Leggi di più...

2. FireBug (usare con Firefox)

3. Dynatrace (utilizzare con IE)

4. Batarang (usare con Chrome)

   È un componente aggiuntivo obsoleto per il browser Chrome, anche se è stabile e può essere utilizzato per monitorare modelli, prestazioni, dipendenze per un'applicazione angolare. Funziona bene per applicazioni su piccola scala e può darti un'idea di cosa tiene la variabile scope a vari livelli. Ti parla di osservatori attivi, guarda le espressioni, guarda le raccolte nell'app.

5. Watcher (utilizzare con Chrome)

   Interfaccia utente piacevole e semplicistica per contare il numero di osservatori in un'app Angolare.

6. Usa il seguente codice per scoprire manualmente il numero di osservatori nella tua app angolare (credito a @Words Like Jared Number of watchers)

```
(function() {
    var root = angular.element(document.getElementsByTagName('body')),
        watchers = [],
        f = function(element) {
        angular.forEach(['$scope', '$isolateScope'], function(scopeProperty) {
            if(element.data() && element.data().hasOwnProperty(scopeProperty)) {
                angular.forEach(element.data()[scopeProperty].$$watchers, function(watcher) {
                watchers.push(watcher);
                });
            }
        });

        angular.forEach(element.children(), function(childElement) {
            f(angular.element(childElement));
        });
    };

    f(root);

    // Remove duplicate watchers
    var watchersWithoutDuplicates = [];
    angular.forEach(watchers, function(item) {
        if(watchersWithoutDuplicates.indexOf(item) < 0) {
            watchersWithoutDuplicates.push(item);
        }
    });
    console.log(watchersWithoutDuplicates.length);
})();
```

7. Sono disponibili diversi strumenti / siti Web online che facilitano un'ampia gamma di funzionalità per creare un profilo dell'applicazione.

   Uno di questi siti è: https://www.webpagetest.org/

   Con questo è possibile eseguire un test di velocità del sito Web gratuito da più località in tutto il mondo utilizzando i browser reali (IE e Chrome) e alla velocità di connessione reale del consumatore. È possibile eseguire semplici test o eseguire test avanzati tra cui transazioni multi-step, acquisizione video, blocco dei contenuti e molto altro.

Prossimi passi:

Fatto con il profilo. Ti porta solo a metà strada lungo la strada. Il prossimo compito è quello di trasformare i risultati in azioni per ottimizzare l'applicazione. Consulta questa documentazione su come migliorare le prestazioni della tua app angolare con semplici trucchi.

Happy Coding :)

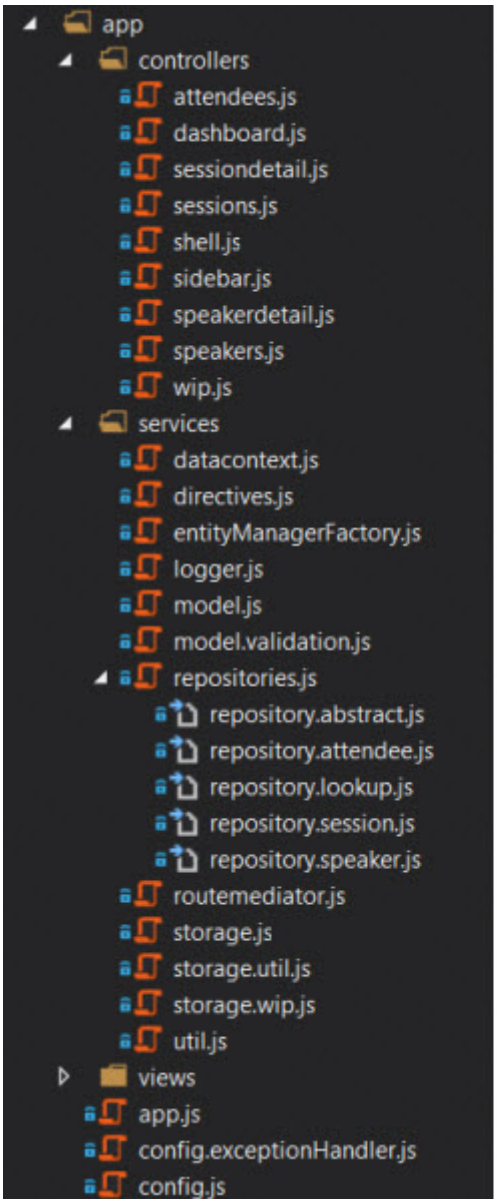
Leggi Profilo delle prestazioni online: https://riptutorial.com/it/angularjs/topic/7033/profilo-delle-prestazioni

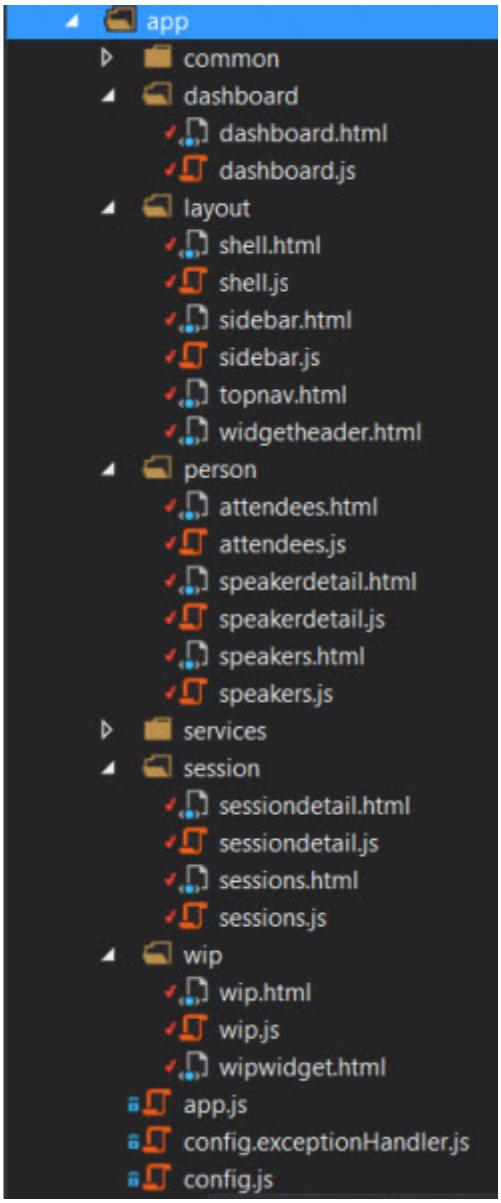
Capitolo 39: Progetto angolare - Struttura delle directory

Examples

Struttura della directory

Una domanda comune tra i nuovi programmatori angolari: "Quale dovrebbe essere la struttura del progetto?". Una buona struttura aiuta a uno sviluppo di applicazioni scalabile. Quando iniziamo un progetto abbiamo due scelte, **Ordina per tipo** (a sinistra) e **Ordina per caratteristica** (a destra). Il secondo è migliore, soprattutto nelle applicazioni di grandi dimensioni, il progetto diventa molto più facile da gestire.





Ordina per tipo (a sinistra)

L'applicazione è organizzata in base al tipo di file.

- **Vantaggio** : buono per le piccole app, per i programmatori che iniziano a utilizzare solo Angular ed è facile da convertire nel secondo metodo.
- **Svantaggio** - Anche per le piccole app inizia a essere più difficile trovare un file specifico. Ad esempio, una vista e il controller sono in due cartelle separate.

Ordina per caratteristica (a destra)

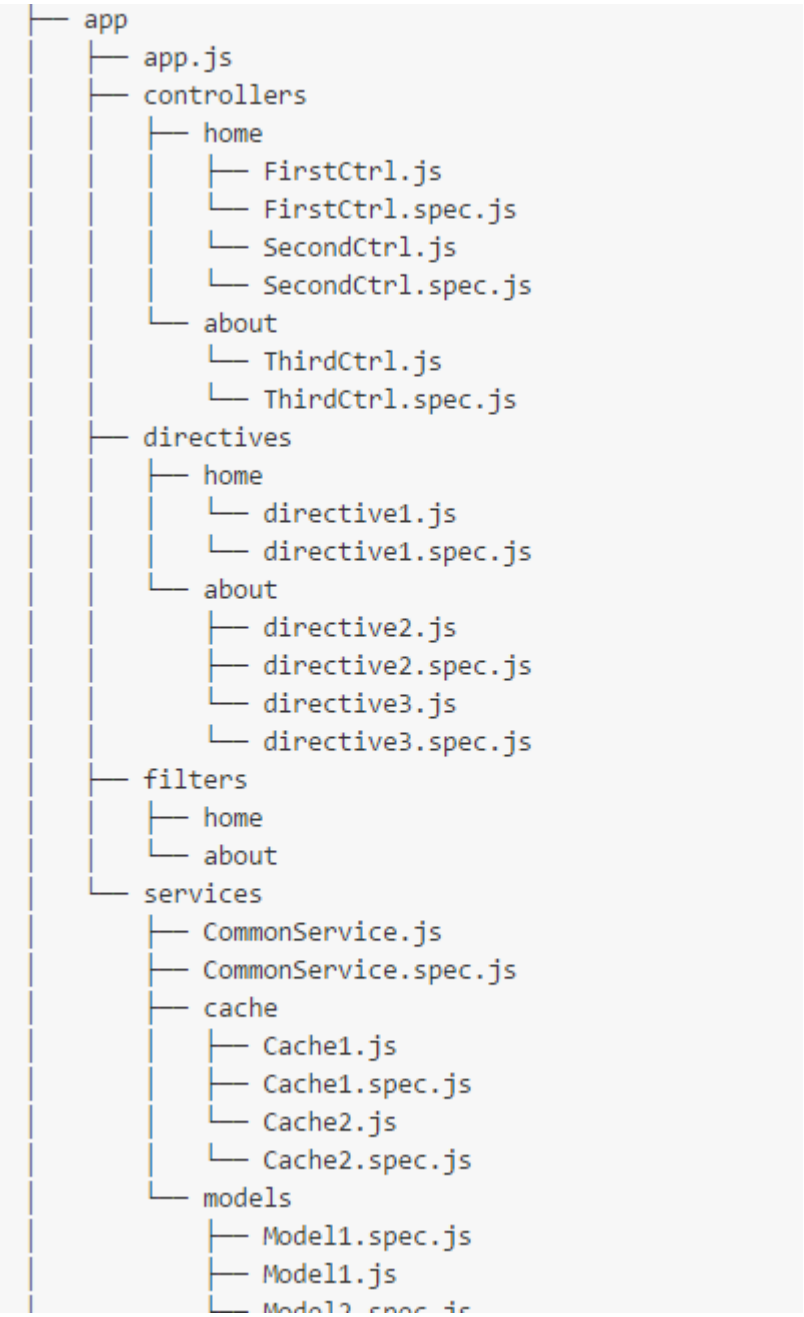
Il metodo di organizzazione suggerito in cui i file sono ordinati per tipo di funzionalità.

Tutte le viste di layout e i controller vanno nella cartella di layout, il contenuto di amministrazione va nella cartella di amministrazione e così via.

- **Vantaggio** : quando si cerca una sezione di codice che determina una determinata funzione, si trova in una cartella.
- **Svantaggio** - I servizi sono un po 'diversi in quanto "servono" molte funzionalità.

Puoi leggere di più su questo argomento su Angular Structure: Refactoring for Growth

La struttura dei file suggerita che combina entrambi i metodi sopra citati:



Credito a: Guida allo stile angolare

Leggi Progetto angolare - Struttura delle directory online:
https://riptutorial.com/it/angularjs/topic/6148/progetto-angolare---struttura-delle-directory

Capitolo 40: Promesse angolari con servizio $ q

Examples

Usando $ q.tutti per gestire più promesse

È possibile utilizzare la funzione `$q.all` per chiamare un metodo `.then` dopo che una serie di promesse è stata risolta correttamente e recuperare i dati risolti.

Esempio:

JS:

```
$scope.data = []

$q.all([
   $http.get("data.json"),
   $http.get("more-data.json"),
   ]).then(function(responses) {
     $scope.data = responses.map((resp) => resp.data);
   });
```

Il codice precedente esegue `$http.get` 2 volte per i dati nei file json locali, quando entrambi `get` metodo completato risolvono le promesse associate, quando tutte le promesse nell'array vengono risolte, il metodo `.then` inizia con entrambi i dati promessi all'interno delle `responses` argomento dell'array.

I dati vengono quindi mappati in modo che possano essere visualizzati sul modello, quindi possiamo mostrarli

HTML:

```
<ul>
   <li ng-repeat="d in data">
      <ul>
         <li ng-repeat="item in d">{{item.name}}: {{item.occupation}}</li>
      </ul>
   </li>
</ul>
```

JSON:

```
[{
  "name": "alice",
  "occupation": "manager"
}, {
  "name": "bob",
  "occupation": "developer"
```

```
}]
```

Usare il costruttore $ q per creare promesse

La funzione costruttore `$q` viene utilizzata per creare promesse da API asincrone che utilizzano callback per restituire risultati.

> $ q (funzione (risoluzione, rifiuto) {...})

La funzione di costruzione riceve una funzione invocata con due argomenti, `resolve` e `reject` che sono funzioni utilizzate per risolvere o rifiutare la promessa.

Esempio 1:

```
function $timeout(fn, delay) {
    return = $q(function(resolve, reject) {
         setTimeout(function() {
              try {
                 let r = fn();
                 resolve(r);
              }
              catch (e) {
                 reject(e);
              }
         }, delay);
     };
}
```

L'esempio precedente crea una promessa dall'API WindowTimers.setTimeout . Il framework AngularJS fornisce una versione più elaborata di questa funzione. Per l'utilizzo, consultare il riferimento all'API del servizio Timeout di AngularJS $.

Esempio 2:

```
$scope.divide = function(a, b) {
    return $q(function(resolve, reject) {
      if (b===0) {
        return reject("Cannot devide by 0")
      } else {
        return resolve(a/b);
      }
    });
}
```

Il codice precedente che mostra una funzione di divisione promessa, restituirà una promessa con il risultato o rifiuta con un motivo se il calcolo è impossibile.

È quindi possibile chiamare e utilizzare. `.then`

```
$scope.divide(7, 2).then(function(result) {
    // will return 3.5
}, function(err) {
```

```
    // will not run
})

$scope.divide(2, 0).then(function(result) {
    // will not run as the calculation will fail on a divide by 0
}, function(err) {
    // will return the error string.
})
```

Differire le operazioni usando $ q.defer

Possiamo usare `$q` per posticipare le operazioni al futuro mentre al momento abbiamo un oggetto promessa in sospeso, usando `$q.defer` creiamo una promessa che sarà risolta o respinta in futuro.

Questo metodo non è equivalente all'uso del costruttore `$q` , poiché usiamo `$q.defer` per promettere una routine esistente che può o non può restituire (o che ha mai restituito) una promessa.

Esempio:

```
var runAnimation = function(animation, duration) {
    var deferred = $q.defer();
    try {
        ...
        // run some animation for a given duration
        deferred.resolve("done");
    } catch (err) {
        // in case of error we would want to run the error hander of .then
        deferred.reject(err);
    }
    return deferred.promise;
}

// and then
runAnimation.then(function(status) {}, function(error) {})
```

1. Assicurati di restituire sempre un oggetto `deferred.promise` o rischiare un errore durante il `.then`
2. Assicurati di risolvere o rifiutare sempre il tuo oggetto posticipato o `.then` non funzionare e rischi una perdita di memoria

Usando promesse angolari con $ q servizio

`$q` è un servizio integrato che aiuta nell'esecuzione di funzioni asincrone e utilizza i loro valori di ritorno (o eccezioni) al termine dell'elaborazione.

`$q` è integrato con il meccanismo di osservazione del modello `$rootScope.Scope` , che significa propagazione più rapida della risoluzione o del rifiuto nei modelli ed evita inutili ridondazioni del browser, il che comporterebbe un'interfaccia utente tremolante.

Nel nostro esempio, chiamiamo la nostra factory `getMyData` , che restituisce un oggetto promessa.

Se l'oggetto è `resolved` , restituisce un numero casuale. Se viene `rejected` , restituisce un rifiuto con un messaggio di errore dopo 2 secondi.

In fabbrica angolare

```
function getMyData($timeout, $q) {
  return function() {
    // simulated async function
    var promise = $timeout(function() {
      if(Math.round(Math.random())) {
        return 'data received!'
      } else {
        return $q.reject('oh no an error! try again')
      }
    }, 2000);
    return promise;
  }
}
```

Utilizzo di promesse in chiamata

```
angular.module('app', [])
.factory('getMyData', getMyData)
.run(function(getData) {
  var promise = getData()
    .then(function(string) {
      console.log(string)
    }, function(error) {
      console.error(error)
    })
    .finally(function() {
      console.log('Finished at:', new Date())
    })
})
```

Per usare le promesse, iniettare `$q` come dipendenza. Qui abbiamo iniettato `$q` nella fabbrica `getMyData` .

```
var defer = $q.defer();
```

Viene `$q.defer()` una nuova istanza di differita chiamando `$q.defer()`

Un oggetto differito è semplicemente un oggetto che espone una promessa così come i metodi associati per risolvere quella promessa. È costruito usando la funzione `$q.deferred()` ed espone tre metodi principali: `resolve()` , `reject()` e `notify()` .

- `resolve(value)` - risolve la promessa derivata con il valore.
- `reject(reason)` - respinge la promessa derivata con la ragione.
- `notify(value)` : fornisce aggiornamenti sullo stato dell'esecuzione della promessa. Questo può essere chiamato più volte prima che la promessa sia risolta o respinta.

Proprietà

L'oggetto promessa associato è accessibile tramite la proprietà promessa. `promise` - {Promise} - promessa oggetto associato a questo rinviato.

Una nuova istanza di promessa viene creata quando viene creata un'istanza posticipata e può essere recuperata chiamando `deferred.promise` .

Lo scopo dell'oggetto `promise` è quello di consentire alle parti interessate di ottenere l'accesso al risultato dell'attività differita al suo completamento.

Metodi promettenti -

- `then(successCallback, [errorCallback], [notifyCallback])` - Indipendentemente da quando la promessa è stata o sarà risolta o rifiutata, quindi chiama uno dei callback di successo o di errore in modo asincrono non appena il risultato è disponibile. I callback vengono chiamati con un singolo argomento: il motivo del risultato o del rifiuto. Inoltre, il callback di notifica può essere chiamato zero o più volte per fornire un'indicazione di avanzamento, prima che la promessa sia risolta o respinta.

- `catch(errorCallback)` - abbreviazione di promise.then (null, errorCallback)

- `finally(callback, notifyCallback)` - consente di osservare l'adempimento o il rifiuto di una promessa, ma di farlo senza modificare il valore finale.

Una delle caratteristiche più potenti delle promesse è la capacità di concatenarle. Ciò consente ai dati di fluire attraverso la catena e di essere manipolati e mutati ad ogni passaggio. Questo è dimostrato con il seguente esempio:

Esempio 1:

```
// Creates a promise that when resolved, returns 4.
function getNumbers() {

  var promise = $timeout(function() {
    return 4;
  }, 1000);

  return promise;
}

// Resolve getNumbers() and chain subsequent then() calls to decrement
// initial number from 4 to 0 and then output a string.
getNumbers()
  .then(function(num) {
      // 4
      console.log(num);
      return --num;
  })
  .then(function (num) {
      // 3
      console.log(num);
      return --num;
  })
   .then(function (num) {
      // 2
```

```
    console.log(num);
    return --num;
})
.then(function (num) {
    // 1
    console.log(num);
    return --num;
})
.then(function (num) {
    // 0
    console.log(num);
    return 'And we are done!';
})
.then(function (text) {
    // "And we are done!"
    console.log(text);
});
```

Inserisci un valore semplice in una promessa utilizzando $ q.when ()

Se tutto ciò che serve è racchiudere il valore in una promessa, non è necessario utilizzare la sintassi lunga come qui:

```
//OVERLY VERBOSE
var defer;
defer = $q.defer();
defer.resolve(['one', 'two']);
return defer.promise;
```

In questo caso puoi scrivere:

```
//BETTER
return $q.when(['one', 'two']);
```

$ q.quando e il suo alias $ q.resolve

> Avvolge un oggetto che potrebbe essere un valore o una promessa in grado (di terze parti) in una promessa di $ q. Ciò è utile quando hai a che fare con un oggetto che potrebbe o non potrebbe essere una promessa, o se la promessa proviene da una fonte che non può essere considerata attendibile.
>
> - Riferimento API servizio AngularJS $ q - $ q.quando

Con il rilascio di AngularJS v1.4.1

Puoi anche utilizzare una `resolve` alias coerente con ES6

```
//ABSOLUTELY THE SAME AS when
return $q.resolve(['one', 'two'])
```

Evita questo anti-pattern

```
var myDeferred = $q.defer();

$http(config).then(function(res) {
   myDeferred.resolve(res);
}, function(error) {
   myDeferred.reject(error);
});

return myDeferred.promise;
```

Non è necessario produrre una promessa con `$q.defer` poiché il servizio $ http restituisce già una promessa.

```
//INSTEAD
return $http(config);
```

È sufficiente restituire la promessa creata dal servizio $ http.

Leggi Promesse angolari con servizio $ q online:
https://riptutorial.com/it/angularjs/topic/4379/promesse-angolari-con-servizio---q

Capitolo 41: provider

Sintassi

- costante (nome, valore);
- valore (nome, valore);
- factory (nome, $ getFn);
- servizio (nome, costruttore);
- fornitore (nome, fornitore);

Osservazioni

I provider sono oggetti singoli che possono essere iniettati, ad esempio, in altri servizi, controller e direttive. Tutti i fornitori sono registrati utilizzando diverse "ricette", in cui il `Provider` è il più flessibile. Tutte le ricette possibili sono:

- Costante
- Valore
- Fabbrica
- Servizio
- Provider

I servizi, le fabbriche e i provider sono tutti inizializzati in modo pigro, il componente viene inizializzato solo se l'applicazione dipende da esso.

I decoratori sono strettamente legati ai fornitori. I decoratori sono utilizzati per intercettare il servizio o la creazione della fabbrica al fine di cambiarne il comportamento o sostituirlo (parti di esso).

Examples

Costante

`Constant` è disponibile sia in fase di configurazione che di esecuzione.

```
angular.module('app',[])
  .constant('endpoint', 'http://some.rest.endpoint') // define
  .config(function(endpoint) {
    // do something with endpoint
    // available in both config- and run phases
  })
  .controller('MainCtrl', function(endpoint) {       // inject
    var vm = this;
    vm.endpoint = endpoint;                          // usage
  });
```

```
<body ng-controller="MainCtrl as vm">
  <div>endpoint = {{ ::vm.endpoint }}</div>
</body>
```

---

endpoint = http: //some.rest.endpoint

Valore

`Value` è disponibile sia nella configurazione che nelle fasi di esecuzione.

```
angular.module('app',[])
  .value('endpoint', 'http://some.rest.endpoint') // define
  .run(function(endpoint) {
    // do something with endpoint
    // only available in run phase
  })
  .controller('MainCtrl', function(endpoint) {    // inject
    var vm = this;
    vm.endpoint = endpoint;                       // usage
  });
```

```
<body ng-controller="MainCtrl as vm">
  <div>endpoint = {{ ::vm.endpoint }}</div>
</body>
```


---

endpoint = http: //some.rest.endpoint

Fabbrica

`Factory` è disponibile in fase di esecuzione.

> La ricetta Factory costruisce un nuovo servizio utilizzando una funzione con zero o più argomenti (dipendono da altri servizi). Il valore di ritorno di questa funzione è l'istanza di servizio creata da questa ricetta.
>
> Factory può creare un servizio di qualsiasi tipo, sia esso un primitivo, oggetto letterale, funzione, o anche un'istanza di un tipo personalizzato.

```
angular.module('app',[])
  .factory('endpointFactory', function() {
    return {
      get: function() {
        return 'http://some.rest.endpoint';
      }
    };
  })
  .controller('MainCtrl', function(endpointFactory) {
    var vm = this;
    vm.endpoint = endpointFactory.get();
  });
```

```
<body ng-controller="MainCtrl as vm">
  <div>endpoint = {{::vm.endpoint }}</div>
</body>
```

---

endpoint = http: //some.rest.endpoint

Servizio

`Service` è disponibile in fase di esecuzione.

La ricetta del servizio produce un servizio proprio come le ricette Valore o Fabbrica, ma lo fa *invocando un costruttore con il nuovo operatore* . Il costruttore può accettare zero o più argomenti, che rappresentano le dipendenze richieste dall'istanza di questo tipo.

```
angular.module('app',[])
  .service('endpointService', function() {
    this.get = function() {
      return 'http://some.rest.endpoint';
    };
  })
  .controller('MainCtrl', function(endpointService) {
    var vm = this;
    vm.endpoint = endpointService.get();
  });
```

```
<body ng-controller="MainCtrl as vm">
  <div>endpoint = {{::vm.endpoint }}</div>
</body>
```

---

endpoint = http: //some.rest.endpoint

Provider

`Provider` è disponibile sia in fase di configurazione che di esecuzione.

La ricetta del provider è sintatticamente definita come un tipo personalizzato che implementa un metodo `$get` .

È necessario utilizzare la ricetta Provider solo quando si desidera esporre un'API per la configurazione a livello di applicazione che deve essere eseguita prima dell'avvio dell'applicazione. Questo di solito è interessante solo per i servizi riutilizzabili il cui comportamento potrebbe dover variare leggermente tra le applicazioni.

```
angular.module('app',[])
  .provider('endpointProvider', function() {
    var uri = 'n/a';

    this.set = function(value) {
```

```
      uri = value;
    };

    this.$get = function() {
      return {
        get: function() {
          return uri;
        }
      };
    };
  })
  .config(function(endpointProviderProvider) {
    endpointProviderProvider.set('http://some.rest.endpoint');
  })
  .controller('MainCtrl', function(endpointProvider) {
    var vm = this;
    vm.endpoint = endpointProvider.get();
  });
```

```
<body ng-controller="MainCtrl as vm">
  <div>endpoint = {{::vm.endpoint }}</div>
</body>
```

---

endpoint = http: //some.rest.endpoint

Senza risultati di fase di `config` sarebbe

endpoint = n / a

Leggi provider online: https://riptutorial.com/it/angularjs/topic/5169/provider

Capitolo 42: Routing usando ngRoute

Osservazioni

`ngRoute` è un modulo incorporato che fornisce servizi di routing e deeplinking e direttive per app angolari.

La documentazione completa su `ngRoute` è disponibile su https://docs.angularjs.org/api/ngRoute

Examples

Esempio di base

Questo esempio mostra l'impostazione di una piccola applicazione con 3 percorsi, ciascuno con la propria vista e il proprio controller, usando la sintassi del `controllerAs` .

Configuriamo il nostro router con la funzione `.config` angolare

1. `$routeProvider` in `.config`
2. Definiamo i nostri nomi di percorso nel metodo `.when` con un oggetto di definizione del percorso.
3. Forniamo il metodo `.when` con un oggetto che specifica il nostro `template` o `templateUrl` , `controller` e `controllerAs`

app.js

```
angular.module('myApp', ['ngRoute'])
  .controller('controllerOne', function() {
    this.message = 'Hello world from Controller One!';
  })
  .controller('controllerTwo', function() {
    this.message = 'Hello world from Controller Two!';
  })
  .controller('controllerThree', function() {
    this.message = 'Hello world from Controller Three!';
  })
  .config(function($routeProvider) {
    $routeProvider
    .when('/one', {
      templateUrl: 'view-one.html',
      controller: 'controllerOne',
      controllerAs: 'ctrlOne'
    })
    .when('/two', {
      templateUrl: 'view-two.html',
      controller: 'controllerTwo',
      controllerAs: 'ctrlTwo'
    })
    .when('/three', {
      templateUrl: 'view-three.html',
      controller: 'controllerThree',
```

```
      controllerAs: 'ctrlThree'
    })
    // redirect to here if no other routes match
    .otherwise({
      redirectTo: '/one'
    });
  });
```

Quindi nel nostro HTML definiamo la nostra navigazione utilizzando gli `<a>` elementi con `href` , per il nome di una rotta di `helloRoute` che verrà instradato come `<a href="#/helloRoute">My route</a>`

Forniamo anche la nostra vista con un contenitore e la direttiva `ng-view` per iniettare i nostri percorsi.

index.html

```
<div ng-app="myApp">
  <nav>
    <!-- links to switch routes -->
    <a href="#/one">View One</a>
    <a href="#/two">View Two</a>
    <a href="#/three">View Three</a>
  </nav>
  <!-- views will be injected here -->
  <div ng-view></div>
  <!-- templates can live in normal html files -->
  <script type="text/ng-template" id="view-one.html">
    <h1>{{ctrlOne.message}}</h1>
  </script>

  <script type="text/ng-template" id="view-two.html">
    <h1>{{ctrlTwo.message}}</h1>
  </script>

  <script type="text/ng-template" id="view-three.html">
    <h1>{{ctrlThree.message}}</h1>
  </script>
</div>
```

Esempio di parametri di instradamento

Questo esempio estende l'esempio di base che passa i parametri nel percorso per poterli utilizzare nel controller

Per fare ciò abbiamo bisogno di:

1. Configura la posizione e il nome del parametro nel nome della rotta
2. `$routeParams` servizio `$routeParams` nel nostro controller

app.js

```
angular.module('myApp', ['ngRoute'])
  .controller('controllerOne', function() {
    this.message = 'Hello world from Controller One!';
  })
```

```
  .controller('controllerTwo', function() {
    this.message = 'Hello world from Controller Two!';
  })
  .controller('controllerThree', ['$routeParams', function($routeParams) {
    var routeParam = $routeParams.paramName

    if ($routeParams.message) {
        // If a param called 'message' exists, we show it's value as the message
        this.message = $routeParams.message;
    } else {
        // If it doesn't exist, we show a default message
        this.message = 'Hello world from Controller Three!';
    }
  }])
  .config(function($routeProvider) {
    $routeProvider
    .when('/one', {
      templateUrl: 'view-one.html',
      controller: 'controllerOne',
      controllerAs: 'ctrlOne'
    })
    .when('/two', {
      templateUrl: 'view-two.html',
      controller: 'controllerTwo',
      controllerAs: 'ctrlTwo'
    })
    .when('/three', {
      templateUrl: 'view-three.html',
      controller: 'controllerThree',
      controllerAs: 'ctrlThree'
    })
    .when('/three/:message', { // We will pass a param called 'message' with this route
      templateUrl: 'view-three.html',
      controller: 'controllerThree',
      controllerAs: 'ctrlThree'
    })
    // redirect to here if no other routes match
    .otherwise({
      redirectTo: '/one'
    });
  });
```

Quindi, senza apportare modifiche nei nostri modelli, aggiungendo solo un nuovo collegamento con un messaggio personalizzato, possiamo vedere il nuovo messaggio personalizzato nella nostra vista.

index.html

```
<div ng-app="myApp">
  <nav>
    <!-- links to switch routes -->
    <a href="#/one">View One</a>
    <a href="#/two">View Two</a>
    <a href="#/three">View Three</a>
    <!-- New link with custom message -->
    <a href="#/three/This-is-a-message">View Three with "This-is-a-message" custom message</a>
  </nav>
  <!-- views will be injected here -->
  <div ng-view></div>
```

```
  <!-- templates can live in normal html files -->
  <script type="text/ng-template" id="view-one.html">
    <h1>{{ctrlOne.message}}</h1>
  </script>

  <script type="text/ng-template" id="view-two.html">
    <h1>{{ctrlTwo.message}}</h1>
  </script>

  <script type="text/ng-template" id="view-three.html">
    <h1>{{ctrlThree.message}}</h1>
  </script>
</div>
```

Definizione del comportamento personalizzato per i singoli percorsi

Il modo più semplice di definire un comportamento personalizzato per i singoli percorsi sarebbe abbastanza facile.

In questo esempio lo usiamo per autenticare un utente:

1) `routes.js` : crea una nuova proprietà (come `requireAuth`) per qualsiasi percorso desiderato

```
angular.module('yourApp').config(['$routeProvider', function($routeProvider) {
    $routeProvider
        .when('/home', {
            templateUrl: 'templates/home.html',
            requireAuth: true
        })
        .when('/login', {
            templateUrl: 'templates/login.html',
        })
        .otherwise({
            redirectTo: '/home'
        });
}])
```

2) In un controller di livello superiore che non è associato a un elemento all'interno di `ng-view` (per evitare conflitti con angolare `$routeProvider`), verificare se `newUrl` ha la proprietà `requireAuth` e agire di conseguenza

```
angular.module('YourApp').controller('YourController', ['$scope', 'session', '$location',
    function($scope, session, $location) {

        $scope.$on('$routeChangeStart', function(angularEvent, newUrl) {

            if (newUrl.requireAuth && !session.user) {
                // User isn't authenticated
                $location.path("/login");
            }

        });
    }
]);
```

Leggi Routing usando ngRoute online: https://riptutorial.com/it/angularjs/topic/2391/routing-usando-ngroute

Capitolo 43: Servizi

Examples

Come creare un servizio

```
angular.module("app")
  .service("counterService", function(){

    var service = {
        number: 0
    };

    return service;
   });
```

Come usare un servizio

```
    angular.module("app")

        // Custom services are injected just like Angular's built-in services
        .controller("step1Controller", ['counterService', '$scope', function(counterService,
$scope) {
            counterService.number++;
            // bind to object (by reference), not to value, for automatic sync
            $scope.counter = counterService;
        })
```

Nel modello che usa questo controller scriveresti:

```
// editable
<input ng-model="counter.number" />
```

o

```
// read-only
<span ng-bind="counter.number"></span>
```

Naturalmente, nel codice reale si interagirebbe con il servizio utilizzando i metodi sul controller, che a sua volta delegherà al servizio. L'esempio sopra semplicemente incrementa il valore del contatore ogni volta che il controller viene utilizzato in un modello.

I servizi in Angularjs sono singoletti:

I servizi sono oggetti singleton creati solo una volta per app (da $ injector) e pigri (creati solo quando necessario).

> Un singleton è una classe che consente di creare una sola istanza di se stessa e

fornisce un accesso semplice e facile a detta istanza. Come affermato qui

Creare un servizio usando angular.factory

Definire innanzitutto il servizio (in questo caso utilizza lo schema di fabbrica):

```
.factory('dataService', function() {
    var dataObject = {};
    var service = {
        // define the getter method
        get data() {
            return dataObject;
        },
        // define the setter method
        set data(value) {
            dataObject = value || {};
        }
    };
    // return the "service" object to expose the getter/setter
    return service;
})
```

Ora puoi utilizzare il servizio per condividere i dati tra i controller:

```
.controller('controllerOne', function(dataService) {
    // create a local reference to the dataService
    this.dataService = dataService;
    // create an object to store
    var someObject = {
        name: 'SomeObject',
        value: 1
    };
    // store the object
    this.dataService.data = someObject;
})

.controller('controllerTwo', function(dataService) {
    // create a local reference to the dataService
    this.dataService = dataService;
    // this will automatically update with any changes to the shared data object
    this.objectFromControllerOne = this.dataService.data;
})
```

$ sce: disinfetta e rende il contenuto e le risorse nei modelli

$ sce ("Strict Contextual Escaping") è un servizio angolare incorporato che disinfetta automaticamente i contenuti e le fonti interne nei modelli.

l'iniezione **di** fonti **esterne** e **HTML grezzo** nel modello richiede il wrapping manuale di `$sce` .

In questo esempio creeremo un semplice filtro sanity $ sce: `.

dimostrazione

```
.filter('sanitizer', ['$sce', [function($sce) {
```

```
     return function(content) {
          return $sce.trustAsResourceUrl(content);
      };
}]);
```

Utilizzo nel modello

```
<div ng-repeat="item in items">

    // Sanitize external sources
    <ifrmae ng-src="{{item.youtube_url | sanitizer}}">

    // Sanitaize and render HTML
    <div ng-bind-html="{{item.raw_html_content| sanitizer}}"></div>

</div>
```

Come creare un servizio con dipendenze usando la 'sintassi dell'array'

```
angular.module("app")
  .service("counterService", ["fooService", "barService", function(anotherService,
barService){

    var service = {
        number: 0,
        foo: function () {
            return fooService.bazMethod(); // Use of 'fooService'
        },
        bar: function () {
            return barService.bazMethod(); // Use of 'barService'
        }
    };

    return service;
   }]);
```

Registrazione di un servizio

Il modo più comune e flessibile per creare un servizio utilizza la fabbrica API angular.module:

```
angular.module('myApp.services', []).factory('githubService', function() {
   var serviceInstance = {};
   // Our first service
   return serviceInstance;
});
```

La funzione di fabbrica del servizio può essere una funzione o un array, proprio come il modo in cui creiamo i controller:

```
// Creating the factory through using the
// bracket notation
angular.module('myApp.services', [])
.factory('githubService', [function($http) {
}]);
```

Per esporre un metodo sul nostro servizio, possiamo inserirlo come attributo sull'oggetto servizio.

```
angular.module('myApp.services', [])
    .factory('githubService', function($http) {
        var githubUrl = 'https://api.github.com';
        var runUserRequest = function(username, path) {
        // Return the promise from the $http service
        // that calls the Github API using JSONP
        return $http({
            method: 'JSONP',
            url: githubUrl + '/users/' +
            username + '/' +
            path + '?callback=JSON_CALLBACK'
        });
    }
// Return the service object with a single function
// events
return {
    events: function(username) {
    return runUserRequest(username, 'events');
    }
};
```

Differenza tra servizio e fabbrica

1) Servizi

Un servizio è una funzione di `constructor` che viene invocata una volta a runtime con `new` , proprio come farebbe con plain javascript con la sola differenza che `AngularJs` sta chiamando il `new` dietro le quinte.

C'è una regola del pollice da ricordare in caso di servizi

1. I servizi sono costruttori che vengono chiamati con `new`

Vediamo un semplice esempio in cui dovremmo registrare un servizio che utilizza il servizio `$http` per recuperare i dettagli dello studente e utilizzarlo nel controller

```
function StudentDetailsService($http) {
  this.getStudentDetails = function getStudentDetails() {
    return $http.get('/details');
  };
}

angular.module('myapp').service('StudentDetailsService', StudentDetailsService);
```

Abbiamo appena iniettato questo servizio nel controller

```
function StudentController(StudentDetailsService) {
  StudentDetailsService.getStudentDetails().then(function (response) {
      // handle response
    });
}
angular.module('app').controller('StudentController', StudentController);
```

Quando usare?

Utilizzare `.service()` ovunque si desideri utilizzare un costruttore. Di solito viene utilizzato per creare API pubbliche proprio come `getStudentDetails()` . Ma se non si desidera utilizzare un costruttore e si desidera utilizzare un semplice schema API, allora non c'è molta flessibilità in `.service()` .

2) Fabbrica

Anche se siamo in grado di ottenere tutte le cose usando `.factory()` che faremmo usando `.services()` , non rende `.factory()` "uguale a" `.service()` . È molto più potente e flessibile di `.service()`

A `.factory()` è un modello di progettazione che viene utilizzato per restituire un valore.

Ci sono due regole del pollice da ricordare in caso di fabbriche

1. Le fabbriche restituiscono valori
2. Fabbriche (possono) creare oggetti (Qualsiasi oggetto)

Vediamo alcuni esempi su cosa possiamo fare usando `.factory()`

Restituzione di oggetti letterali

Vediamo un esempio in cui factory è usato per restituire un oggetto usando un modello di modulo Revealing di base

```
function StudentDetailsService($http) {
  function getStudentDetails() {
    return $http.get('/details');
  }
  return {
    getStudentDetails: getStudentDetails
  };
}

angular.module('myapp').factory('StudentDetailsService', StudentDetailsService);
```

Utilizzo all'interno di un controller

```
function StudentController(StudentDetailsService) {
  StudentDetailsService.getStudentDetails().then(function (response) {
      // handle response
    });
}
angular.module('app').controller('StudentController', StudentController);
```

Chiusure di ritorno

Cos'è una chiusura?

Le chiusure sono funzioni che fanno riferimento a variabili utilizzate localmente, ma definite in un ambito che racchiude.

Di seguito è riportato un esempio di chiusura

```
function closureFunction(name) {
  function innerClosureFunction(age) { // innerClosureFunction() is the inner function, a
closure
    // Here you can manipulate 'age' AND 'name' variables both
  };
};
```

La parte *"meravigliosa"* è che può accedere al `name` che si trova nell'ambito genitore.

Consente di utilizzare l'esempio di chiusura sopra all'interno `.factory()`

```
function StudentDetailsService($http) {
  function closureFunction(name) {
  function innerClosureFunction(age) {
    // Here you can manipulate 'age' AND 'name' variables
    };
  };
};

angular.module('myapp').factory('StudentDetailsService', StudentDetailsService);
```

Utilizzo all'interno di un controller

```
function StudentController(StudentDetailsService) {
  var myClosure = StudentDetailsService('Student Name'); // This now HAS the
innerClosureFunction()
  var callMyClosure = myClosure(24); // This calls the innerClosureFunction()
};

angular.module('app').controller('StudentController', StudentController);
```

Creazione di costruttori / istanze

`.service()` crea costruttori con una chiamata a `new` come visto sopra. `.factory()` può anche creare costruttori con una chiamata a `new`

Vediamo un esempio su come ottenere questo

```
function StudentDetailsService($http) {
  function Student() {
    this.age = function () {
        return 'This is my age';
    };
  }
  Student.prototype.address = function () {
        return 'This is my address';
  };
  return Student;
};

angular.module('myapp').factory('StudentDetailsService', StudentDetailsService);
```

Utilizzo all'interno di un controller

```
function StudentController(StudentDetailsService) {
  var newStudent = new StudentDetailsService();

  //Now the instance has been created. Its properties can be accessed.

 newStudent.age();
 newStudent.address();

};

angular.module('app').controller('StudentController', StudentController);
```

Leggi Servizi online: https://riptutorial.com/it/angularjs/topic/1486/servizi

Capitolo 44: Servizio di distinzione rispetto alla fabbrica

Examples

Factory VS Service una volta per tutte

Per definizione:

I servizi sono fondamentalmente funzioni di costruzione. Usano la parola chiave "this".

Le fabbriche sono semplici funzioni quindi restituiscono un oggetto.

Sotto il cappuccio:

Le fabbriche chiamano internamente la funzione provider.

I servizi internamente chiamano la funzione di fabbrica.

Discussione:

Le fabbriche possono eseguire il codice prima di restituire il nostro oggetto letterale.

Ma allo stesso tempo, i Servizi possono anche essere scritti per restituire un oggetto letterale e per eseguire il codice prima di tornare. Sebbene ciò sia controproducente, i servizi sono progettati per funzionare come funzione di costruzione.

In effetti, le funzioni di costruzione in JavaScript possono restituire ciò che vogliono.

Quindi qual è il migliore?

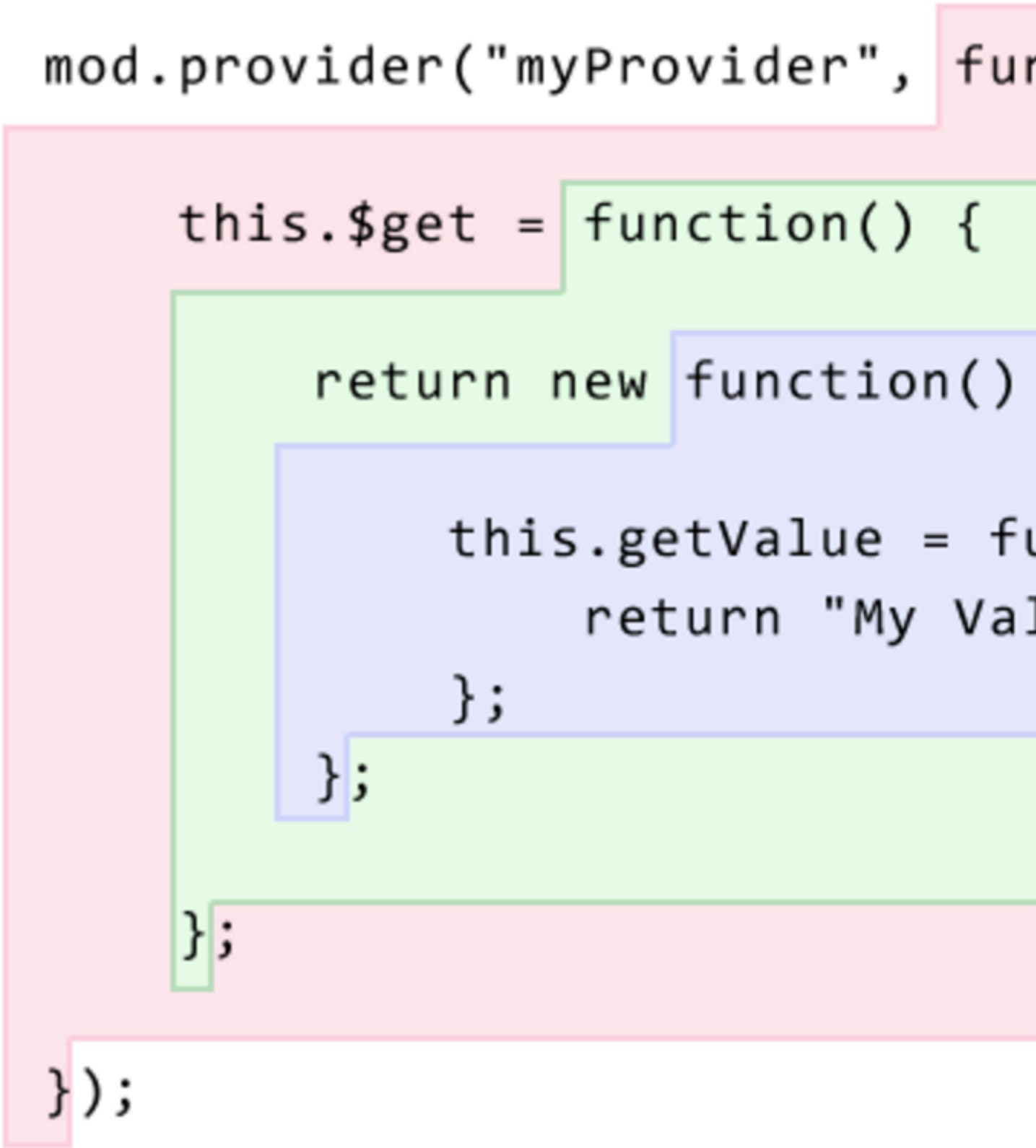
La sintassi dei servizi del costruttore è più vicina alla sintassi di classe di ES6. Quindi la migrazione sarà facile.

Sommario

Quindi, in breve, fornitore, fabbrica e servizio sono tutti fornitori.

Una fabbrica è un caso speciale di un fornitore quando tutto ciò di cui hai bisogno nel tuo provider è una funzione $ get (). Ti permette di scriverlo con meno codice.

Un servizio è un caso speciale di una fabbrica quando si desidera restituire un'istanza di un nuovo oggetto, con lo stesso vantaggio di scrivere meno codice.



Leggi Servizio di distinzione rispetto alla fabbrica online:
https://riptutorial.com/it/angularjs/topic/7099/servizio-di-distinzione-rispetto-alla-fabbrica

Capitolo 45: SignalR with AngularJs

introduzione

In questo articolo ci concentriamo su "Come creare un progetto semplice usando AngularJs e SignalR", in questa formazione è necessario conoscere "come creare app con angularjs", "come creare / utilizzare il servizio su angolare" e le conoscenze di base su SignalR "per questo raccomandiamo https://www.codeproject.com/Tips/590660/Introduction-to-SignalR) .

Examples

SignalR And AngularJs [ChatProject]

passaggio 1: Crea progetto

```
- Application
    - app.js
    - Controllers
        - appController.js
    - Factories
        - SignalR-factory.js
- index.html
- Scripts
    - angular.js
    - jquery.js
    - jquery.signalR.min.js
- Hubs
```

Uso della versione SignalR: signalR-2.2.1

Passaggio 2: Startup.cs e ChatHub.cs

Vai alla tua directory *"/ Hub"* e Aggiungi 2 file *[Startup.cs, ChatHub.cs]*

Startup.cs

```
using Microsoft.Owin;
using Owin;
[assembly: OwinStartup(typeof(SignalR.Hubs.Startup))]

namespace SignalR.Hubs
{
    public class Startup
    {
        public void Configuration(IAppBuilder app)
        {
            app.MapSignalR();
        }
    }
}
```

ChatHub.cs

```
using Microsoft.AspNet.SignalR;

namespace SignalR.Hubs
{
    public class ChatHub : Hub
    {
        public void Send(string name, string message, string time)
        {
            Clients.All.broadcastMessage(name, message, time);
        }
    }
}
```

fase 3: creare un'app angolare

Vai alla tua directory *"/ Applicazione"* e aggiungi il file *[app.js]*

app.js

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

fase 4: creare SignalR Factory

Vai alla tua directory *"/ Applicazioni / Fabbriche"* e aggiungi il file *[SignalR-factory.js]*

SignalR-factory.js

```
app.factory("signalR", function () {
    var factory = {};

    factory.url = function (url) {
        $.connection.hub.url = url;
    }

    factory.setHubName = function (hubName) {
        factory.hub = hubName;
    }

    factory.connectToHub = function () {
        return $.connection[factory.hub];
    }

    factory.client = function () {
        var hub = factory.connectToHub();
        return hub.client;
    }

    factory.server = function () {
        var hub = factory.connectToHub();
        return hub.server;
    }

    factory.start = function (fn) {
        return $.connection.hub.start().done(fn);
    }
```

```
    return factory;
});
```

passaggio 5: aggiorna app.js

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

app.run(function(signalR) {
    signalR.url("http://localhost:21991/signalr");
});
```

localhost: 21991 / signalr | **questo è il tuo SignalR Hub Urls**

passaggio 6: aggiungi controller

Vai alla directory *"/ Applicazione / Controllori"* e Aggiungi il file *[appController.js]*

```
app.controller("ctrl", function ($scope, signalR) {
    $scope.messages = [];
    $scope.user = {};

    signalR.setHubName("chatHub");

    signalR.client().broadcastMessage = function (name, message, time) {
        var newChat = { name: name, message: message, time: time };

        $scope.$apply(function() {
            $scope.messages.push(newChat);
        });
    };

    signalR.start(function () {
        $scope.send = function () {
            var dt = new Date();
            var time = dt.getHours() + ":" + dt.getMinutes() + ":" + dt.getSeconds();

            signalR.server().send($scope.user.name, $scope.user.message, time);
        }
    });
});
```

signalR.setHubName ("chatHub") | **[ChatHub] (classe pubblica)>** *ChatHub.cs*

Nota: non inserire *HubName* con maiuscolo, la **prima lettera** è inferiore.

signalR.client () | questo metodo tenta di connettersi agli hub e ottenere tutte le funzioni negli Hub, in questo esempio abbiamo "chatHub", per ottenere la funzione "broadcastMessage ()";

passaggio 7: aggiungi index.html nel percorso della directory

index.html

```
<!DOCTYPE html>
```

```
<html ng-app="app" ng-controller="ctrl">
<head>
    <meta charset="utf-8" />
    <title>SignalR Simple Chat</title>
</head>
<body>
    <form>
        <input type="text" placeholder="name" ng-model="user.name" />
        <input type="text" placeholder="message" ng-model="user.message" />
        <button ng-click="send()">send</button>

        <ul>
            <li ng-repeat="item in messages">
                <b ng-bind="item.name"></b> <small ng-bind="item.time"></small> :
{{item.message}}
            </li>
        </ul>
    </form>

    <script src="Scripts/angular.min.js"></script>
    <script src="Scripts/jquery-1.6.4.min.js"></script>
    <script src="Scripts/jquery.signalR-2.2.1.min.js"></script>
    <script src="signalr/hubs"></script>
    <script src="app.js"></script>
    <script src="SignalR-factory.js"></script>
</body>
</html
```

Risultato con l'immagine

Utente 1 (invia e ricevi)

Utente 2 (invia e ricevi)

Leggi SignalR with AngularJs online: https://riptutorial.com/it/angularjs/topic/9964/signalr-with-angularjs

Capitolo 46: sintesi del ciclo digest

Sintassi

- $ scope. $ watch (watchExpression, callback, [deep compare])
- $ Portata. $ Digest ()
- $ Portata. $ Applicare ([exp])

Examples

associazione dati bidirezionale

Angolare ha un po 'di magia sotto il suo cappuccio. abilita il DOM vincolante a variabili js reali.

Angular utilizza un ciclo, denominato " *ciclo digest* ", che viene chiamato dopo ogni modifica di una variabile, chiamando callback che aggiornano il DOM.

Ad esempio, la direttiva `ng-model` allega un eventup di `keyup` a questo input:

```
<input ng-model="variable" />
```

Ogni volta che si `keyup` evento `keyup` , viene `keyup` il *ciclo digest* .

Ad un certo punto, il *ciclo digest* itera su un callback che aggiorna il contenuto di questo intervallo:

```
<span>{{variable}}</span>
```

Il ciclo di vita di base di questo esempio, riassume (molto schematicamente) come funziona l'angolare:

1. Scansioni angolari html
    - direttiva `ng-model` crea un listener `keyup` sull'input
    - `expression` inside span aggiunge un callback al *ciclo digest*
2. L'utente interagisce con l'input
    - ascoltatore di `keyup` inizia il *ciclo di digestione*
    - *il ciclo di digest* chiama il callback
    - Gli aggiornamenti del callback comprendono i contenuti

$ digest e $ watch

L'implementazione del binding dei dati a due vie, per ottenere il risultato dell'esempio precedente, potrebbe essere eseguita con due funzioni principali:

- **$ digest** è chiamato dopo un'interazione dell'utente (vincolante DOM => variabile)
- **$ watch** imposta un callback da chiamare dopo le modifiche alle variabili (binding variable

=> DOM)

nota: questo è un esempio è una dimostrazione, non il codice angolare reale

---

```
<input id="input"/>
<span id="span"></span>
```

Le due funzioni di cui abbiamo bisogno:

```
var $watches = [];
function $digest(){
    $watches.forEach(function($w){
        var val = $w.val();
        if($w.prevVal !== val){
            $w.callback(val, $w.prevVal);
            $w.prevVal = val;
        }
    })
}
function $watch(val, callback){
    $watches.push({val:val, callback:callback, prevVal: val() })
}
```

Ora potremmo usare queste funzioni per collegare una variabile al DOM (l'angolare viene fornito con direttive incorporate che faranno questo per voi):

```
var realVar;
//this is usually done by ng-model directive
input1.addEventListener('keyup',function(e){
    realVar=e.target.value;
    $digest()
}, true);

//this is usually done with {{expressions}} or ng-bind directive
$watch(function(){return realVar},function(val){
    span1.innerHTML = val;
});
```

Fuori rotta, le implementazioni reali sono più complesse e supportano parametri come l' **elemento** a cui legarsi e **quale variabile** utilizzare

Un esempio in esecuzione può essere trovato qui: https://jsfiddle.net/azofxd4j/

l'albero $ scope

L'esempio precedente è abbastanza buono quando è necessario associare un singolo elemento html a una singola variabile.

In realtà, abbiamo bisogno di legare molti elementi a molte variabili:

```
<span ng-repeat="number in [1,2,3,4,5]">{{number}}</span>
```

Questa `ng-repeat` lega 5 elementi a 5 variabili chiamate `number` , con un valore diverso per ognuna di esse!

Il modo in cui l'angolare raggiunge questo comportamento sta usando un contesto separato per ogni elemento che ha bisogno di variabili separate. Questo contesto è chiamato scope.

Ogni ambito contiene proprietà, che sono le variabili legate al DOM, e le funzioni `$digest` e `$watch` sono implementate come metodi dell'ambito.

Il DOM è un albero e le variabili devono essere utilizzate in diversi livelli dell'albero:

```
<div>
    <input ng-model="person.name" />
    <span ng-repeat="number in [1,2,3,4,5]">{{number}} {{person.name}}</span>
</div>
```

Ma come abbiamo visto, il contesto (o l'ambito) delle variabili all'interno di `ng-repeat` è diverso dal contesto sopra di esso. Per risolvere questo - gli ambiti angolari implementa un albero.

Ogni ambito ha una serie di figli, e chiamando i suoi `$digest` metodo verrà eseguito tutti i dei suoi figli `$digest` metodo.

In questo modo - dopo aver modificato l'input - viene chiamato `$digest` per l'ambito div, che quindi esegue `$digest` per i suoi 5 figli - che aggiornerà il suo contenuto.

Un'implementazione semplice per un ambito potrebbe essere simile a questa:

```
function $scope(){
    this.$children = [];
    this.$watches = [];
}

$scope.prototype.$digest = function(){
    this.$watches.forEach(function($w){
        var val = $w.val();
        if($w.prevVal !== val){
            $w.callback(val, $w.prevVal);
          $w.prevVal = val;
        }
    });
    this.$children.forEach(function(c){
        c.$digest();
    });
}

$scope.prototype.$watch = function(val, callback){
    this.$watches.push({val:val, callback:callback, prevVal: val() })
}
```

nota: questo è un esempio è una dimostrazione, non il codice angolare reale

Leggi sintesi del ciclo digest online: https://riptutorial.com/it/angularjs/topic/3156/sintesi-del-ciclo-

digest

Capitolo 47: Stampare

Osservazioni

Crea una classe ng-hide nel file css. ng-show / hide non funzionerà senza la classe.

Più dettagli

Examples

Servizio di stampa

Servizio:

```
angular.module('core').factory('print_service', ['$rootScope', '$compile', '$http',
'$timeout','$q',
    function($rootScope, $compile, $http, $timeout,$q) {

        var printHtml = function (html) {
            var deferred = $q.defer();
            var hiddenFrame = $('<iframe style="display:
none"></iframe>').appendTo('body')[0];

            hiddenFrame.contentWindow.printAndRemove = function() {
                hiddenFrame.contentWindow.print();
                $(hiddenFrame).remove();
                deferred.resolve();
            };

            var htmlContent =   "<!doctype html>"+
                                "<html>"+
                                    '<head><link rel="stylesheet" type="text/css"
href="/style/css/print.css"/></head>'+
                                    '<body onload="printAndRemove();">' +
                                        html +
                                    '</body>'+
                                "</html>";

            var doc = hiddenFrame.contentWindow.document.open("text/html", "replace");
            doc.write(htmlContent);
            doc.close();
            return deferred.promise;
        };

        var openNewWindow = function (html) {
            var newWindow = window.open("debugPrint.html");
            newWindow.addEventListener('load', function(){
                $(newWindow.document.body).html(html);
            }, false);
        };

        var print = function (templateUrl, data) {

            $rootScope.isBeingPrinted = true;
```

```
            $http.get(templateUrl).success(function(template){
                var printScope = $rootScope.$new()
                angular.extend(printScope, data);
                var element = $compile($('<div>' + template + '</div>'))(printScope);
                var waitForRenderAndPrint = function() {
                    if(printScope.$$phase || $http.pendingRequests.length) {
                        $timeout(waitForRenderAndPrint, 1000);
                    } else {
                        // Replace printHtml with openNewWindow for debugging
                        printHtml(element.html());
                        printScope.$destroy();
                    }
                };
                waitForRenderAndPrint();
            });
        };

        var printFromScope = function (templateUrl, scope, afterPrint) {
            $rootScope.isBeingPrinted = true;
            $http.get(templateUrl).then(function(response){
                var template = response.data;
                var printScope = scope;
                var element = $compile($('<div>' + template + '</div>'))(printScope);
                var waitForRenderAndPrint = function() {
                    if (printScope.$$phase || $http.pendingRequests.length) {
                        $timeout(waitForRenderAndPrint);
                    } else {
                        // Replace printHtml with openNewWindow for debugging
                        printHtml(element.html()).then(function() {
                            $rootScope.isBeingPrinted = false;
                            if (afterPrint) {
                                afterPrint();
                            }
                        });
                    }
                };
                waitForRenderAndPrint();
            });
        };

        return {
            print : print,
            printFromScope : printFromScope
        }
    }
]);
```

Controller:

```
var template_url = '/views/print.client.view.html';
print_service.printFromScope(template_url,$scope,function(){
      // Print Completed
});
```

Leggi Stampare online: https://riptutorial.com/it/angularjs/topic/6750/stampare

Capitolo 48: Test unitari

Osservazioni

Questo argomento fornisce esempi per testare unitamente i vari costrutti in AngularJS. I test unitari vengono spesso scritti utilizzando Jasmine , un popolare framework di test guidato dal comportamento. Quando si testano i costrutti angolari, è necessario includere ngMock come dipendenza durante l'esecuzione dei test unitari.

Examples

Unit test un filtro

Codice filtro:

```
angular.module('myModule', []).filter('multiplier', function() {
  return function(number, multiplier) {
    if (!angular.isNumber(number)) {
      throw new Error(number + " is not a number!");
    }
    if (!multiplier) {
      multiplier = 2;
    }
    return number * multiplier;
  }
});
```

Il test:

```
describe('multiplierFilter', function() {
  var filter;

  beforeEach(function() {
    module('myModule');
    inject(function(multiplierFilter) {
      filter = multiplierFilter;
    });
  });

  it('multiply by 2 by default', function() {
    expect(filter(2)).toBe(4);
    expect(filter(3)).toBe(6);
  });

  it('allow to specify custom multiplier', function() {
    expect(filter(2, 4)).toBe(8);
  });

  it('throws error on invalid input', function() {
    expect(function() {
      filter(null);
    }).toThrow();
```

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

Correre!

Nota: nella chiamata in `inject` nel test, il filtro deve essere specificato in base al nome + *Filtro* . La causa di ciò è che ogni volta che si registra un filtro per il modulo, Angular lo registra con un `Filter` aggiunto al suo nome.

Unit test a component (1.5+)

Codice componente:

```
angular.module('myModule', []).component('myComponent', {
  bindings: {
    myValue: '<'
  },
  controller: function(MyService) {
    this.service = MyService;
    this.componentMethod = function() {
      return 2;
    };
  }
});
```

Il test:

```
describe('myComponent', function() {
  var component;

  var MyServiceFake = jasmine.createSpyObj(['serviceMethod']);

  beforeEach(function() {
    module('myModule');
    inject(function($componentController) {
      // 1st - component name, 2nd - controller injections, 3rd - bindings
      component = $componentController('myComponent', {
        MyService: MyServiceFake
      }, {
        myValue: 3
      });
    });
  });

  /** Here you test the injector. Useless. */

  it('injects the binding', function() {
    expect(component.myValue).toBe(3);
  });

  it('has some cool behavior', function() {
    expect(component.componentMethod()).toBe(2);
  });
});
```

Correre!

Unit test di un controller

Codice del controller:

```
angular.module('myModule', [])
  .controller('myController', function($scope) {
    $scope.num = 2;
    $scope.doSomething = function() {
      $scope.num += 2;
    }
  });
```

Il test:

```
describe('myController', function() {
  var $scope;
  beforeEach(function() {
    module('myModule');
    inject(function($controller, $rootScope) {
      $scope = $rootScope.$new();
      $controller('myController', {
        '$scope': $scope
      })
    });
  });
  it('should increment `num` by 2', function() {
    expect($scope.num).toEqual(2);
    $scope.doSomething();
    expect($scope.num).toEqual(4);
  });
});
```

Correre!

Unit test di un servizio

Codice di servizio

```
angular.module('myModule', [])
  .service('myService', function() {
    this.doSomething = function(someNumber) {
      return someNumber + 2;
    }
  });
```

Il test

```
describe('myService', function() {
  var myService;
  beforeEach(function() {
    module('myModule');
    inject(function(_myService_) {
      myService = _myService_;
    });
  });
```

```
  it('should increment `num` by 2', function() {
    var result = myService.doSomething(4);
    expect(result).toEqual(6);
  });
});
```

Correre!

Unit test di una direttiva

Codice della direttiva

```
angular.module('myModule', [])
  .directive('myDirective', function() {
    return {
      template: '<div>{{greeting}} {{name}}!</div>',
      scope: {
        name: '=',
        greeting: '@'
      }
    };
  });
```

Il test

```
describe('myDirective', function() {
  var element, scope;
  beforeEach(function() {
    module('myModule');
    inject(function($compile, $rootScope) {
      scope = $rootScope.$new();
      element = angular.element("<my-directive name='name' greeting='Hello'></my-directive>");
      $compile(element)(scope);
      /* PLEASE NEVER USE scope.$digest(). scope.$apply use a protection to avoid to run a
digest loop when there is already one, so, use scope.$apply() instead. */
      scope.$apply();
    })
  });

  it('has the text attribute injected', function() {
    expect(element.html()).toContain('Hello');
  });

  it('should have proper message after scope change', function() {
    scope.name = 'John';
    scope.$apply();
    expect(element.html()).toContain("John");
    scope.name = 'Alice';
    expect(element.html()).toContain("John");
    scope.$apply();
    expect(element.html()).toContain("Alice");
  });
});
```

Correre!

Leggi Test unitari online: https://riptutorial.com/it/angularjs/topic/1689/test-unitari

Capitolo 49: Trucchi e trappole per AngularJS

Examples

Il binding dei dati a due vie smette di funzionare

Uno dovrebbe avere in mente che:

1. L'associazione dei dati di Angular si basa sull'eredità prototipale di JavaScript, quindi è soggetta a ombreggiamenti variabili .
2. Un ambito figlio normalmente eredita prototipicamente dall'ambito principale. Un'eccezione a questa regola è una direttiva che ha un ambito isolato in quanto non eredita prototipicamente.
3. Ci sono alcune direttive che creano un nuovo ambito figlio: `ng-repeat` , `ng-switch` , `ng-view` , `ng-if` , `ng-controller` , `ng-include` , ecc.

Ciò significa che quando si tenta di associare in modo bidirezionale alcuni dati a una primitiva che si trova all'interno di un ambito figlio (o viceversa), le cose potrebbero non funzionare come previsto. Ecco un esempio di quanto sia facile *"rompere"* AngularJS.

Questo problema può essere facilmente evitato seguendo questi passaggi:

1. Avere un "." all'interno del tuo modello HTML ogni volta che leghi dei dati
2. Utilizzare la sintassi `controllerAs` in quanto promuove l'uso del binding a un oggetto "tratteggiato"
3. $ genitore può essere utilizzato per accedere alle variabili `scope` genitore piuttosto che all'ambito secondario. come dentro `ng-if` possiamo usare `ng-model="$parent.foo"` ..

Un'alternativa per quanto sopra è di associare `ngModel` a una funzione getter / setter che aggiornerà la versione cache del modello quando viene chiamata con argomenti, o la restituisce quando viene chiamata senza argomenti. Per utilizzare una funzione getter / setter, è necessario aggiungere `ng-model-options="{ getterSetter: true }"` all'elemento con l'attributo `ngModal` e chiamare la funzione getter se si desidera visualizzare il suo valore nell'espressione (Esempio di lavoro).

Esempio

Vista:

```
<div ng-app="myApp" ng-controller="MainCtrl">
    <input type="text" ng-model="foo" ng-model-options="{ getterSetter: true }">
    <div ng-if="truthyValue">
        <!-- I'm a child scope (inside ng-if), but i'm synced with changes from the outside
scope -->
        <input type="text" ng-model="foo">
    </div>
    <div>$scope.foo: {{ foo() }}</div>
</div>
```

controller:

```
angular.module('myApp', []).controller('MainCtrl', ['$scope', function($scope) {
    $scope.truthyValue = true;

    var _foo = 'hello'; // this will be used to cache/represent the value of the 'foo' model

    $scope.foo = function(val) {
        // the function return the the internal '_foo' varibale when called with zero
arguments,
        // and update the internal `_foo` when called with an argument
        return arguments.length ? (_foo = val) : _foo;
    };
}]);
```

Best Practice : è meglio mantenere i getter veloci perché è probabile che Angular li chiami più frequentemente rispetto ad altre parti del codice (riferimento).

Cose da fare quando si utilizza html5Mode

Quando si utilizza `html5Mode([mode])` è necessario che:

1. Specifica l'URL di base per l'applicazione con un `<base href="">` nella testa del tuo `index.html` .
2. È importante che il tag `base` venga prima di qualsiasi tag con richieste di URL. Altrimenti, questo potrebbe causare questo errore - `"Resource interpreted as stylesheet but transferred with MIME type text/html"` . Per esempio:

   ```
   <head>
       <meta charset="utf-8">
       <title>Job Seeker</title>

       <base href="/">

       <link rel="stylesheet" href="bower_components/bootstrap/dist/css/bootstrap.css" />
       <link rel="stylesheet" href="/styles/main.css">
   </head>
   ```

3. Se non si desidera specificare un tag `base` , configurare `$locationProvider` per non richiedere un tag `base` passando un oggetto definizione con `requireBase:false` a `$locationProvider.html5Mode()` questo modo:

   ```
   $locationProvider.html5Mode({
       enabled: true,
       requireBase: false
   });
   ```

4. Per supportare il caricamento diretto degli URL HTML5, è necessario abilitare la riscrittura degli URL lato server. Da AngularJS / Guida per gli sviluppatori / Utilizzo $ posizione

   > L'utilizzo di questa modalità richiede la riscrittura degli URL sul lato server, in pratica è necessario riscrivere tutti i collegamenti al punto di accesso

dell'applicazione (ad esempio `index.html`). Anche richiedere un tag `<base>` è importante in questo caso, poiché consente a Angular di distinguere tra la parte dell'URL che è la base dell'applicazione e il percorso che deve essere gestito dall'applicazione.

Un'eccellente risorsa per gli esempi di riscrittura delle richieste per varie implementazioni del server HTTP può essere trovata nelle domande frequenti sull'i-router - Come: Configurare il server per lavorare con html5Mode . Ad esempio, Apache

```
RewriteEngine on

# Don't rewrite files or directories
RewriteCond %{REQUEST_FILENAME} -f [OR]
RewriteCond %{REQUEST_FILENAME} -d
RewriteRule ^ - [L]

# Rewrite everything else to index.html to allow html5 state links
RewriteRule ^ index.html [L]
```

nginx

```
server {
    server_name my-app;

    root /path/to/app;

    location / {
        try_files $uri $uri/ /index.html;
    }
}
```

Esprimere

```
var express = require('express');
var app = express();

app.use('/js', express.static(__dirname + '/js'));
app.use('/dist', express.static(__dirname + '/../dist'));
app.use('/css', express.static(__dirname + '/css'));
app.use('/partials', express.static(__dirname + '/partials'));

app.all('/*', function(req, res, next) {
    // Just send the index.html for other files to support HTML5Mode
    res.sendFile('index.html', { root: __dirname });
});

app.listen(3006); //the port you want to use
```

7 Deadly Sins of AngularJS

Di seguito l'elenco di alcuni errori che gli sviluppatori fanno spesso durante l'uso delle funzionalità di AngularJS, alcune lezioni apprese e soluzioni a loro.

1. Manipolazione del DOM attraverso il controller

È legale, ma deve essere evitato. I controller sono i luoghi in cui si definiscono le dipendenze, si vincolano i dati alla vista e si sviluppa ulteriormente la logica aziendale. Puoi tecnicamente manipolare il DOM in un controller, ma ogni volta che hai bisogno di manipolazioni uguali o simili in un'altra parte della tua app, sarà necessario un altro controller. Pertanto, la migliore pratica di questo approccio è la creazione di una direttiva che includa tutte le manipolazioni e utilizzi la direttiva in tutta l'app. Quindi, il controller lascia intatta la vista e fa il suo lavoro. In una direttiva, la funzione di collegamento è il posto migliore per manipolare il DOM. Ha pieno accesso all'ambito e all'elemento, quindi utilizzando una direttiva, puoi anche trarre vantaggio dalla riusabilità.

```
link: function($scope, element, attrs) {
    //The best place to manipulate DOM
}
```

È possibile accedere agli elementi DOM nella funzione di collegamento in diversi modi, ad esempio il parametro `element` , il metodo `angular.element()` o puro Javascript.

2. Dati vincolanti in fase di transizione

AngularJS è famoso per il suo binding di dati bidirezionale. Tuttavia, a volte potresti riscontrare che i tuoi dati sono limitati a senso unico all'interno delle direttive. Fermati lì, AngularJS non è sbagliato ma probabilmente tu. Le direttive sono un po 'pericolose perché sono coinvolti gli ambiti figlio e gli ambiti isolati. Supponiamo che tu abbia la seguente direttiva con una transclusione

```
<my-dir>
    <my-transclusion>
    </my-transclusion>
</my-dir>
```

E all'interno della mia transclusione, hai alcuni elementi che sono legati ai dati nello scope esterno.

```
<my-dir>
    <my-transclusion>
        <input ng-model="name">
    </my-transclusion>
</my-dir>
```

Il codice sopra non funzionerà correttamente. Qui, la transclusione crea un ambito figlio e puoi ottenere la variabile nome, giusto, ma qualsiasi modifica apportata a questa variabile rimarrà lì. Quindi, puoi veramente accedere a questa variabile come **$ parent.name** . Tuttavia, questo utilizzo potrebbe non essere la migliore pratica. Un approccio migliore sarebbe il wrapping delle variabili all'interno di un oggetto. Ad esempio, nel controller è possibile creare:

```
$scope.data = {
    name: 'someName'
}
```

Quindi nella transclusione, puoi accedere a questa variabile tramite l'oggetto "dati" e vedere che il bind a due vie funziona perfettamente!

```
<input ng-model="data.name">
```

Non solo nelle transizioni, ma in tutta l'app, è una buona idea usare la notazione puntata.

3. Più direttive insieme

In realtà è legale utilizzare due direttive insieme all'interno dello stesso elemento, purché obbedisca alla regola: non possono esistere due ambiti isolati sullo stesso elemento. In generale, quando si crea una nuova direttiva personalizzata, si assegna un ambito isolato per facilitare il passaggio dei parametri. Supponendo che le direttive myDirA e myDirB abbiano ambiti isoleted e myDirC no, il seguente elemento sarà valido:

```
<input my-dir-a my-dirc>
```

mentre il seguente elemento causerà l'errore della console:

```
<input my-dir-a my-dir-b>
```

Pertanto, le direttive devono essere utilizzate con saggezza, prendendo in considerazione gli scopi.

4. Uso improprio di $ emit

$ emettono, $ broadcast e $ on, funzionano in un principio di ricevitore-ricevitore. In altre parole, sono un mezzo di comunicazione tra i controllori. Ad esempio, la riga seguente emette "someEvent" dal controller A, per essere intercettata dal controller interessato B.

```
$scope.$emit('someEvent', args);
```

E la seguente riga cattura il "someEvent"

```
$scope.$on('someEvent', function(){});
```

Finora tutto sembra perfetto. Ma ricorda che, se il controller B non è ancora stato richiamato, l'evento non verrà catturato, il che significa che sia i controller emettitore che quelli riceventi devono essere richiamati per farlo funzionare. Quindi, ancora una volta, se non sei sicuro di dover usare $ emit, costruire un servizio sembra un modo migliore.

5. Uso improprio di $ scope. $ Watch

$ scope. $ watch è usato per guardare un cambiamento variabile. Ogni volta che una variabile è cambiata, questo metodo viene invocato. Tuttavia, un errore comune è cambiare la variabile all'interno di $ scope. $ Watch. Ciò causerà incoerenza e un infinito ciclo $ digest a un certo punto.

```
$scope.$watch('myCtrl.myVariable', function(newVal) {
    this.myVariable++;
});
```

Quindi, nella funzione sopra, assicurati di non avere operazioni su myVariable e newVal.

6. Metodi vincolanti per le viste

Questo è uno dei peccati più mortali. AngularJS ha un'associazione a due vie e ogni volta che qualcosa cambia, le viste vengono aggiornate molte volte. Quindi, se si associa un metodo a un attributo di una vista, quel metodo potrebbe essere potenzialmente chiamato cento volte, cosa che fa impazzire anche durante il debug. Tuttavia, ci sono solo alcuni attributi che sono costruiti per il binding dei metodi, come ng-click, ng-blur, ng-on-change, ecc, che si aspettano metodi come paremeter. Ad esempio, supponi di avere la seguente vista nel tuo markup:

```
<input ng-disabled="myCtrl.isDisabled()" ng-model="myCtrl.name">
```

Qui si controlla lo stato disabilitato della vista tramite il metodo isDisabled. Nel controller myCtrl, hai:

```
vm.isDisabled = function(){
    if(someCondition)
        return true;
    else
        return false;
}
```

In teoria, può sembrare corretto ma tecnicamente ciò causerà un sovraccarico, poiché il metodo verrà eseguito innumerevoli volte. Per risolvere questo problema, dovresti associare una variabile. Nel controller, deve essere presente la seguente variabile:

```
vm.isDisabled
```

È possibile avviare nuovamente questa variabile nell'attivazione del controller

```
if(someCondition)
    vm.isDisabled = true
else
    vm.isDisabled = false
```

Se la condizione non è stabile, puoi associarla a un altro evento. Quindi dovresti associare questa variabile alla vista:

```
<input ng-disabled="myCtrl.isDisabled" ng-model="myCtrl.name">
```

Ora, tutti gli attributi della vista hanno ciò che si aspettano e i metodi verranno eseguiti solo quando necessario.

7. Non usando le funzionalità di Angular

AngularJS offre grande praticità con alcune delle sue funzionalità, non solo per semplificare il codice ma anche per renderlo più efficiente. Alcune di queste funzionalità sono elencate di seguito:

1. **angular.forOgni** per i loop (Attenzione, non puoi "spezzare", puoi solo evitare di entrare nel corpo, quindi considera le prestazioni qui).
2. **angular.element** per selettori DOM
3. **angular.copy** : utilizzare questo quando non si deve modificare l'oggetto principale
4. **Le convalide dei form** sono già fantastiche. Usa sporco, incontaminato, toccato, valido, richiesto e così via.
5. Oltre al debugger di Chrome, usa **il debug remoto anche** per lo sviluppo mobile.
6. E assicurati di usare **Batarang** . È un'estensione gratuita di Chrome in cui è possibile ispezionare facilmente gli ambiti
   .

Leggi Trucchi e trappole per AngularJS online:
https://riptutorial.com/it/angularjs/topic/3208/trucchi-e-trappole-per-angularjs

Capitolo 50: ui-router

Osservazioni

Cos'è l' `ui-router` ?

> Angular UI-Router è un framework di routing di applicazioni a pagina singola lato client per AngularJS.
>
> I framework di routing per SPA aggiornano l'URL del browser mentre l'utente naviga attraverso l'app. Al contrario, questo consente di modificare l'URL del browser per guidare la navigazione attraverso l'app, consentendo così all'utente di creare un segnalibro in una posizione profonda all'interno della SPA.
>
> Le applicazioni UI-Router sono modellate come un albero gerarchico di stati. UI-Router fornisce una macchina a stati per gestire le transizioni tra questi stati dell'applicazione in modo simile alla transazione.

Link utili

Puoi trovare la documentazione ufficiale dell'API qui . Per domande su `ui-router` VS `ng-router` , puoi trovare una risposta ragionevolmente dettagliata qui . Tenere presente che `ng-router` ha già rilasciato un nuovo aggiornamento ng-router chiamato `ngNewRouter` (compatibile con Angular 1.5 + / 2.0) che supporta gli stati proprio come l'ui-router. Puoi leggere ulteriori informazioni su `ngNewRouter` qui .

Examples

Esempio di base

app.js

```
angular.module('myApp', ['ui.router'])
  .controller('controllerOne', function() {
    this.message = 'Hello world from Controller One!';
  })
  .controller('controllerTwo', function() {
    this.message = 'Hello world from Controller Two!';
  })
  .controller('controllerThree', function() {
    this.message = 'Hello world from Controller Three!';
  })
  .config(function($stateProvider, $urlRouterProvider) {
    $stateProvider
      .state('one', {
        url: "/one",
        templateUrl: "view-one.html",
        controller: 'controllerOne',
        controllerAs: 'ctrlOne'
```

```
      })
      .state('two', {
        url: "/two",
        templateUrl: "view-two.html",
        controller: 'controllerTwo',
        controllerAs: 'ctrlTwo'
      })
      .state('three', {
        url: "/three",
        templateUrl: "view-three.html",
        controller: 'controllerThree',
        controllerAs: 'ctrlThree'
      });

      $urlRouterProvider.otherwise('/one');
  });
```

index.html

```
<div ng-app="myApp">
  <nav>
    <!-- links to switch routes -->
    <a ui-sref="one">View One</a>
    <a ui-sref="two">View Two</a>
    <a ui-sref="three">View Three</a>
  </nav>
  <!-- views will be injected here -->
  <div ui-view></div>
  <!-- templates can live in normal html files -->
  <script type="text/ng-template" id="view-one.html">
    <h1>{{ctrlOne.message}}</h1>
  </script>

  <script type="text/ng-template" id="view-two.html">
    <h1>{{ctrlTwo.message}}</h1>
  </script>

  <script type="text/ng-template" id="view-three.html">
    <h1>{{ctrlThree.message}}</h1>
  </script>
</div>
```

Visualizzazioni multiple

app.js

```
angular.module('myApp', ['ui.router'])
  .controller('controllerOne', function() {
    this.message = 'Hello world from Controller One!';
  })
  .controller('controllerTwo', function() {
    this.message = 'Hello world from Controller Two!';
  })
  .controller('controllerThree', function() {
    this.message = 'Hello world from Controller Three!';
  })
  .controller('controllerFour', function() {
    this.message = 'Hello world from Controller Four!';
```

```
  })
  .config(function($stateProvider, $urlRouterProvider) {
    $stateProvider
      .state('one', {
        url: "/one",
        views: {
          "viewA": {
            templateUrl: "view-one.html",
            controller: 'controllerOne',
            controllerAs: 'ctrlOne'
          },
          "viewB": {
            templateUrl: "view-two.html",
            controller: 'controllerTwo',
            controllerAs: 'ctrlTwo'
          }
        }
      })
      .state('two', {
        url: "/two",
        views: {
          "viewA": {
            templateUrl: "view-three.html",
            controller: 'controllerThree',
            controllerAs: 'ctrlThree'
          },
          "viewB": {
            templateUrl: "view-four.html",
            controller: 'controllerFour',
            controllerAs: 'ctrlFour'
          }
        }
      });

    $urlRouterProvider.otherwise('/one');
  });
```

index.html

```
<div ng-app="myApp">
  <nav>
    <!-- links to switch routes -->
    <a ui-sref="one">Route One</a>
    <a ui-sref="two">Route Two</a>
  </nav>
  <!-- views will be injected here -->
  <div ui-view="viewA"></div>
  <div ui-view="viewB"></div>
  <!-- templates can live in normal html files -->
  <script type="text/ng-template" id="view-one.html">
    <h1>{{ctrlOne.message}}</h1>
  </script>

  <script type="text/ng-template" id="view-two.html">
    <h1>{{ctrlTwo.message}}</h1>
  </script>

  <script type="text/ng-template" id="view-three.html">
    <h1>{{ctrlThree.message}}</h1>
  </script>
```

```
  <script type="text/ng-template" id="view-four.html">
    <h1>{{ctrlFour.message}}</h1>
  </script>
</div>
```

Utilizzo delle funzioni di risoluzione per caricare i dati

app.js

```
angular.module('myApp', ['ui.router'])
  .service('User', ['$http', function User ($http) {
    this.getProfile = function (id) {
      return $http.get(...) // method to load data from API
    };
  }])
  .controller('profileCtrl', ['profile', function profileCtrl (profile) {
    // inject resolved data under the name of the resolve function
    // data will already be returned and processed
    this.profile = profile;
  }])
  .config(['$stateProvider', '$urlRouterProvider', function ($stateProvider,
$urlRouterProvider) {
    $stateProvider
      .state('profile', {
        url: "/profile/:userId",
        templateUrl: "profile.html",
        controller: 'profileCtrl',
        controllerAs: 'vm',
        resolve: {
          profile: ['$stateParams', 'User', function ($stateParams, User) {
            // $stateParams will contain any parameter defined in your url
            return User.getProfile($stateParams.userId)
              // .then is only necessary if you need to process returned data
              .then(function (data) {
                return doSomeProcessing(data);
              });
          }]
        }
      }]);

      $urlRouterProvider.otherwise('/');
  });
```

profile.html

```
<ul>
  <li>Name: {{vm.profile.name}}</li>
  <li>Age: {{vm.profile.age}}</li>
  <li>Sex: {{vm.profile.sex}}</li>
</ul>
```

Visualizza la voce Wiki UI-Router su risolve qui .

Le funzioni di risoluzione devono essere risolte prima che `$stateChangeSuccess` evento `$stateChangeSuccess` , il che significa che l'interfaccia utente non verrà caricata fino a quando non

saranno terminate *tutte le* funzioni di risoluzione sullo stato. Questo è un ottimo modo per garantire che i dati siano disponibili per il controller e l'interfaccia utente. Tuttavia, puoi vedere che una funzione di risoluzione dovrebbe essere veloce per evitare di appendere l'interfaccia utente.

Viste nidificate / Stati

app.js

```
var app = angular.module('myApp',['ui.router']);

app.config(function($stateProvider,$urlRouterProvider) {

    $stateProvider

    .state('home', {
        url: '/home',
        templateUrl: 'home.html',
        controller: function($scope){
            $scope.text = 'This is the Home'
        }
    })

    .state('home.nested1',{
        url: '/nested1',
        templateUrl:'nested1.html',
        controller: function($scope){
            $scope.text1 = 'This is the nested view 1'
        }
    })

    .state('home.nested2',{
        url: '/nested2',
        templateUrl:'nested2.html',
        controller: function($scope){
            $scope.fruits = ['apple','mango','oranges'];
        }
    });

    $urlRouterProvider.otherwise('/home');

});
```

index.html

```
  <div ui-view></div>
   <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.5.8/angular.min.js"></script>
   <script src="angular-ui-router.min.js"></script>
   <script src="app.js"></script>
```

home.html

```
<div>
<h1> {{text}} </h1>
<br>
    <a ui-sref="home.nested1">Show nested1</a>
```

```
    <br>
    <a ui-sref="home.nested2">Show nested2</a>
    <br>

    <div ui-view></div>
</div>
```

nested1.html

```
<div>
<h1> {{text1}} </h1>
</div>
```

nested2.html

```
<div>
    <ul>
      <li ng-repeat="fruit in fruits">{{ fruit }}</li>
    </ul>
</div>
```

Leggi ui-router online: https://riptutorial.com/it/angularjs/topic/2545/ui-router

Capitolo 51: Uso di direttive integrate

Examples

Nascondi / Mostra elementi HTML

Questo esempio nasconde gli elementi html.

```
<!DOCTYPE html>
<html ng-app="myDemoApp">
  <head>
    <script src="https://code.angularjs.org/1.5.8/angular.min.js"></script>
    <script>

      function HideShowController() {
        var vm = this;
        vm.show=false;
        vm.toggle= function() {
          vm.show=!vm.show;
        }
      }

      angular.module("myDemoApp", [/* module dependencies go here */])
        .controller("hideShowController", [HideShowController]);
    </script>
  </head>
  <body ng-cloak>
    <div ng-controller="hideShowController as vm">
      <a style="cursor: pointer;" ng-show="vm.show" ng-click="vm.toggle()">Show Me!</a>
      <a style="cursor: pointer;" ng-hide="vm.show" ng-click="vm.toggle()">Hide Me!</a>
    </div>
  </body>
</html>
```

Dimostrazione dal vivo

Spiegazione passo passo:

1. `ng-app="myDemoApp"` , la direttiva ngApp dice angolare che un elemento DOM è controllato da uno specifico angular.module denominato "myDemoApp".
2. `<script src="[//angular include]">` include js angolare.
3. `HideShowController` funzione `HideShowController` è definita con un'altra funzione chiamata `toggle` che aiuta a nascondere l'elemento.
4. `angular.module(...)` crea un nuovo modulo.
5. `.controller(...)` Controller angolare e restituisce il modulo per il concatenamento;
6. `ng-controller` direttiva `ng-controller` è l'aspetto chiave di come angolare supporta i principi alla base del modello di progettazione Model-View-Controller.
7. `ng-show` direttiva `ng-show` mostra l'elemento HTML specificato se l'espressione fornita è vera.
8. `ng-hide` direttiva `ng-hide` nasconde l'elemento HTML specificato se l'espressione fornita è vera.
9. `ng-click` direttiva `ng-click` attiva una funzione di commutazione all'interno del controller

Leggi Uso di direttive integrate online: https://riptutorial.com/it/angularjs/topic/7644/uso-di-direttive-integrate

Capitolo 52: Utilizzo di AngularJS con TypeScript

Sintassi

- $ scope: ng.IScope - questo è il modo in typescript per definire il tipo per una particolare variabile.

Examples

Controller angolari in dattiloscritto

Come definito nella documentazione di AngularJS

> Quando un controller è collegato al DOM tramite la direttiva ng-controller, Angular crea un'istanza di un nuovo oggetto Controller, utilizzando la funzione di costruzione del controller specificata. Un nuovo ambito figlio verrà creato e reso disponibile come parametro iniettabile alla funzione di costruzione del Controller come $ scope.

I controller possono essere creati molto facilmente utilizzando le classi dattiloscritto.

```
module App.Controllers {
    class Address {
        line1: string;
        line2: string;
        city: string;
        state: string;
    }
    export class SampleController {
        firstName: string;
        lastName: string;
        age: number;
        address: Address;
        setUpWatches($scope: ng.IScope): void {
            $scope.$watch(() => this.firstName, (n, o) => {
                //n is string and so is o
            });
        };
        constructor($scope: ng.IScope) {
            this.setUpWatches($scope);
        }
    }
}
```

Il Javascript risultante è

```
var App;
(function (App) {
    var Controllers;
```

```
    (function (Controllers) {
        var Address = (function () {
            function Address() {
            }
            return Address;
        }());
        var SampleController = (function () {
            function SampleController($scope) {
                this.setUpWatches($scope);
            }
            SampleController.prototype.setUpWatches = function ($scope) {
                var _this = this;
                $scope.$watch(function () { return _this.firstName; }, function (n, o) {
                    //n is string and so is o
                });
            };
            ;
            return SampleController;
        }());
        Controllers.SampleController = SampleController;
    })(Controllers = App.Controllers || (App.Controllers = {}));
})(App || (App = {}));
//# sourceMappingURL=ExampleController.js.map
```

Dopo aver effettuato la classe controller, il modulo angular js sul controller può essere eseguito in modo semplice utilizzando la classe

```
app
 .module('app')
 .controller('exampleController', App.Controller.SampleController)
```

Uso del controller con la sintassi ControllerAs

Il Controller che abbiamo creato può essere istanziato e utilizzato usando il `controller as` Sintassi. Questo perché abbiamo messo la variabile direttamente sulla classe controller e non su `$scope` .

Usando `controller as someName` è necessario separare il controller da `$scope` Quindi, non è necessario iniettare $ scope come dipendenza nel controller.

Modo tradizionale:

```
// we are using $scope object.
app.controller('MyCtrl', function ($scope) {
  $scope.name = 'John';
});

<div ng-controller="MyCtrl">
  {{name}}
</div>
```

Ora, con `controller as` Sintassi :

```
// we are using the "this" Object instead of "$scope"
app.controller('MyCtrl', function() {
  this.name = 'John';
```

```
});

<div ng-controller="MyCtrl as info">
  {{info.name}}
</div>
```

Se istanziate una "classe" in JavaScript, potreste fare questo:

```
var jsClass = function () {
  this.name = 'John';
}
var jsObj = new jsClass();
```

Quindi, ora possiamo usare `jsObj` istanza di `jsObj` per accedere a qualsiasi metodo o proprietà di `jsClass` .

In angolare, facciamo lo stesso tipo di cosa. Usiamo il controller come sintassi per l'istanziazione.

Utilizzo di bundling / minification

Il modo in cui viene iniettato $ scope nelle funzioni di costruzione del controllore è un modo per dimostrare e utilizzare l'opzione di base dell'iniezione di dipendenza angolare, ma non è pronta per la produzione in quanto non può essere minimizzata. Questo perché il sistema di minificazione modifica i nomi delle variabili e l'iniezione delle dipendenze di anguar utilizza i nomi dei parametri per sapere cosa deve essere iniettato. Quindi per un esempio la funzione di costruzione di ExampleController viene ridotta al seguente codice.

```
function n(n){this.setUpWatches(n)
```

e `$scope` è cambiato in `n` !
per ovviare a ciò possiamo aggiungere un array di $ inject (`string[]`). In modo che il DI angolare sa cosa iniettare in quale posizione è la funzione di costruzione del controllore.
Quindi il dattiloscritto sopra diventa

```
module App.Controllers {
    class Address {
        line1: string;
        line2: string;
        city: string;
        state: string;
    }
    export class SampleController {
        firstName: string;
        lastName: string;
        age: number;
        address: Address;
        setUpWatches($scope: ng.IScope): void {
            $scope.$watch(() => this.firstName, (n, o) => {
                //n is string and so is o
            });
        };
        static $inject : string[] = ['$scope'];
        constructor($scope: ng.IScope) {
```

```
            this.setUpWatches($scope);
        }
    }
}
```

Perché la sintassi di ControllerAs?

Funzione controller

La funzione controller non è altro che una funzione di costruzione JavaScript. Quindi, quando una vista carica il `function context` della `function context` (**`this`**) viene impostato sull'oggetto controller.

Caso 1 :

```
this.constFunction = function() { ... }
```

Viene creato `controller object` , non in `$scope` . le viste non possono accedere alle funzioni definite sull'oggetto controller.

Esempio :

```
<a href="#123" ng-click="constFunction()"></a> // It will not work
```

Caso 2:

```
$scope.scopeFunction = function() { ... }
```

Viene creato `$scope object` , non `controller object` . le viste possono accedere solo alle funzioni definite sull'oggetto `$scope` .

Esempio :

```
<a href="#123" ng-click="scopeFunction()"></a> // It will work
```

Perché ControllerAs?

- **`ControllerAs`** sintassi **`ControllerAs`** rende molto più chiaro dove gli oggetti vengono manipolati. `oneCtrl.name` e `anotherCtrl.name` rende molto più facile identificare che si ha un `name` assegnato da più controller diversi per scopi diversi, ma se entrambi hanno usato lo stesso `$scope.name` e due diversi elementi HTML su una pagina che sono entrambi associati a **`{{name}}`** quindi è difficile identificare quale sia il controller.
- Nascondere lo `$scope` ed esporre i membri dal controller alla vista tramite un `intermediary object` . Impostando `this.*` , possiamo esporre solo ciò che vogliamo esporre dal controller

alla vista.

```
<div ng-controller="FirstCtrl">
    {{ name }}
    <div ng-controller="SecondCtrl">
        {{ name }}
        <div ng-controller="ThirdCtrl">
            {{ name }}
        </div>
    </div>
</div>
```

Qui, nel caso di cui sopra `{{ name }}` sarà molto confuso da usare e inoltre non sappiamo quale sia correlato a quale controller.

```
<div ng-controller="FirstCtrl as first">
    {{ first.name }}
    <div ng-controller="SecondCtrl as second">
        {{ second.name }}
        <div ng-controller="ThirdCtrl as third">
            {{ third.name }}
        </div>
    </div>
</div>
```

Perché $ scope?

- Usa `$scope` quando devi accedere a uno o più metodi di $ scope come `$watch` , `$digest` , `$emit` , `$http` ecc.
- limitare quali proprietà e / o metodi sono esposti a `$scope` , quindi passarli esplicitamente a `$scope` come necessario.

Leggi Utilizzo di AngularJS con TypeScript online:
https://riptutorial.com/it/angularjs/topic/3477/utilizzo-di-angularjs-con-typescript

Titoli di coda

S. No	Capitoli	Contributors
1	Iniziare con AngularJS	Abhishek Pandey, After Class, Andrés Encarnación, AnonymousNotReally, badzilla, Charlie H, Chirag Bhatia - chirag64, Community, daniellmb, David G., Devid Farinelli, Eugene, fracz, Franck Dernoncourt, Gabriel Pires, Gourav Garg , H. Pauwelyn, Igor Raush, jengeb, Jeroen, John F., Léo Martin, Lotus91, LucyMarieJ, M. Junaid Salaat, Maaz.Musa, Matt, Mikko Viitala, Mistalis, Nemanja Trifunovic, Nhan, Nico, pathe.kiran, Patrick, Pushpendra, Richard Hamilton, Stepan Suvorov, Stephen Leppik, Sunil Lama, superluminary, Syed Priom, timbo, Ven, vincentvanjoe, Yasin Patel, Ze Rubeus, Артем Комаров
2	$ richiesta http	CENT1PEDE, jaredsk, Liron Ilayev
3	Ambiti $ angolari	Abhishek Maurya, elliot-j, Eugene, jaredsk, Liron Ilayev, MoLow, Prateek Gupta, RamenChef, ryansstack, Tony
4	angularjs con filtro dati, impaginazione ecc	Paresh Maghodiya
5	Archiviazione delle sessioni	Rohit Jindal
6	Caricamento lento	Muli Yulzary
7	Come funziona il collegamento dei dati	Lucas L, Sasank Sunkavalli, theblindprophet
8	Compiti grunt	Mikko Viitala
9	componenti	Alon Eitan, Artem K., badzilla, BarakD, Hubert Grzeskowiak, John F., Juri, M. Junaid Salaat, Mansouri, Pankaj Parkar, Ravi Singh, sgarcia.dev, Syed Priom, Yogesh Mangaj
10	Condivisione dei dati	elliot-j, Grundy, Lex, Mikko Viitala, Mistalis, Nix, prit4fun, Rohit Jindal, sgarcia.dev, Sunil Lama
11	Controller	Adam Harrison, Aeolingamenfel, Alon Eitan, badzilla, Bon Macalindong, Braiam, chatuur, DerekMT12, Dr. Cool, Florian, George Kagan, Grundy, Jared Hooper, Liron Ilayev, M. Junaid Salaat, Mark Cidade, Matthew Green, Mike, Nad Flores,

		Praveen Poonia, RamenChef, Sébastien Deprez, sgarcia.dev, thegreenpizza, timbo, Und3rTow, WMios
12	Controller con ES6	Bouraoui KACEM
13	Convalida del modulo	Alon Eitan, fantarama, garyx, Mikko Viitala, Richard Hamilton, Rohit Jindal, shane, svarog, timbo
14	costanti	Sylvain
15	Debug	Aman, AWolf, Vinay K
16	decoratori	Mikko Viitala
17	direttiva ng-class	Dr. Cool
18	Direttive che utilizzano ngModelController	Nikos Paraskevopoulos
19	Direttive incorporate	Adam Harrison, Alon Eitan, Aron, AWolf, Ayan, Bon Macalindong, CENT1PEDE, Devid Farinelli, DillonChanis, Divya Jain, Dr. Cool, Eric Siebeneich, George Kagan, Grinn, gustavohenke, IncrediApp, kelvinelove, Krupesh Kotecha, Liron Ilayev, m.e.conroy, Maciej Gurban, Mansouri, Mikko Viitala, Mistalis, Mitul, MoLow, Naga2Raja, ngLover, Nishant123, Piet, redunderthebed, Richard Hamilton, svarog, tanmay, theblindprophet, timbo, Tomislav Stankovic, vincentvanjoe, Vishal Singh
20	Direttive personalizzate	Alon Eitan, br3w5, casraf, Cody Stott, Daniel, Everettss, Filipe Amaral, Gaara, Gavishiddappa Gadagi, Jinw, jkris, mnoronha, Pushpendra, Rahul Bhooteshwar, Sajal, sgarcia.dev, Stephan, theblindprophet, TheChetan, Yuri Blanc
21	eventi	CodeWarrior, Nguyen Tran, Rohit Jindal, RyanDawkins, sgarcia.dev, shaN, Shashank Vivek
22	filtri	Aeolingamenfel, developer033, Ed Hinchliffe, fracz, gustavohenke, Matthew Green, Nico
23	Filtri personalizzati	doodhwala, Pat, Sylvain
24	Filtri personalizzati con ES6	Bouraoui KACEM
25	Funzioni ausiliarie integrate	MoLow, Pranav C Balan, svarog
26	Il Sé o questa	It-Z, Jim

	variabile in un controller	
27	Iniezione di dipendenza	Andrea, badzilla, Gavishiddappa Gadagi, George Kagan, MoLow, Omri Aharon
28	Interceptor HTTP	G Akshay, Istvan Reiter, MeanMan, Mistalis, mnoronha
29	Migrazione verso Angular 2+	ShinDarth
30	moduli	Alon Eitan, Ankit, badzilla, Bon Macalindong, Matthew Green, Nad Flores, ojus kulkarni, sgarcia.dev, thegreenpizza
31	MVC angolare	Ashok choudhary, Gavishiddappa Gadagi, Jim
32	ng-repeat	Divya Jain, Jim, Sender, zucker
33	ng-style	Divya Jain, Jim
34	ng-view	Aayushi Jain, Manikandan Velayutham, Umesh Shende
35	Opzioni di collegamenti AngularJS (`=`, `@`, `&` ecc.)	Alon Eitan, Lucas L, Makarov Sergey, Nico, zucker
36	Preparati per la produzione - Grunt	JanisP
37	Profilazione e prestazioni	Ajeet Lakhani, Alon Eitan, Andrew Piliser, Anfelipe, Anirudha, Ashwin Ramaswami, atul mishra, Braiam, bwoebi, chris, Dania, Daniel Molin, daniellmb, Deepak Bansal, Divya Jain, DotBot, Dr. Cool, Durgpal Singh, fracz, Gabriel Pires, George Kagan, Grundy, JanisP, Jared Hooper, jhampton, John Slegers, jusopi, M22an, Matthew Green, Mistalis, Mudassir Ali, Nhan, Psaniko, Richard Hamilton, RyanDawkins, sgarcia.dev, theblindprophet, user3632710, vincentvanjoe, Yasin Patel, Ze Rubeus
38	Profilo delle prestazioni	Deepak Bansal
39	Progetto angolare - Struttura delle directory	jitender, Liron Ilayev
40	Promesse angolari con servizio $ q	Alon Eitan, caiocpricci2, ganqqwerty, georgeawg, John F., Muli Yulzary, Praveen Poonia, Richard Hamilton, Rohit Jindal, svarog

41	provider	Mikko Viitala
42	Routing usando ngRoute	Alon Eitan, Alvaro Vazquez, camabeh, DotBot, sgarcia.dev, svarog
43	Servizi	Abdellah Alaoui, Alvaro Vazquez, AnonDCX, DotBot, elliot-j, Flash, Gavishiddappa Gadagi, Hubert Grzeskowiak, Lex, Nishant123
44	Servizio di distinzione rispetto alla fabbrica	Deepak Bansal
45	SignalR with AngularJs	Maher
46	sintesi del ciclo digest	Alon Eitan, chris, MoLow, prit4fun
47	Stampare	ziaulain
48	Test unitari	daniellmb, elliot-j, fracz, Gabriel Pires, Nico, ronapelbaum
49	Trucchi e trappole per AngularJS	Alon Eitan, Cosmin Ababei, doctorsherlock, Faruk Yazıcı, ngLover, Phil
50	ui-router	George Kagan, H.T, Michael P. Bazos, Ryan Hamley, sgarcia.dev
51	Uso di direttive integrate	Gourav Garg
52	Utilizzo di AngularJS con TypeScript	Parv Sharma, Rohit Jindal