EBook Gratuito

APPRENDIMENTO knockout.js

Free unaffiliated eBook created from
Stack Overflow contributors.

#knockout.j

s

Sommario

Di ... 1

Capitolo 1: Iniziare con knockout.js ... 2

Osservazioni ... 2

Versioni ... 2

Examples ... 2

Installazione o configurazione ... 2

Includi come script ... 2

Utilizzando un CDN ... 3

Installa da npm ... 3

Installa da pergola ... 3

Installa da NuGet ... 3

Per iniziare: Hello world! ... 3

Creazione di un documento HTML e attivazione di knockout.js ... 3

Come funziona il file creato ... 4

Osservabili calcolati ... 5

Capitolo 2: Associazioni - Campi modulo ... 7

Examples ... 7

Clic ... 7

Opzioni ... 7

disabilitato abilitato ... 8

Sottoscrivi ... 8

Valore ... 9

Capitolo 3: Associazioni: testo e aspetto ... 11

Examples ... 11

Testo ... 11

CSS ... 11

Visibile ... 12

attr ... 12

HTML ... 12

Capitolo 4: Attacchi ... 14

Sintassi 14
Osservazioni 14
Che legame c'è 14
Sotto il cofano (breve panoramica) 14
Quando usare le parentesi 15
Examples 16
Se / se non 16
Per ciascuno 16
Con 17
Visibile 18
Capitolo 5: Binding personalizzati 19
Examples 19
Registrazione vincolante 19
Associazione di dissolvenza in dissolvenza in entrata / uscita 19
Il testo personalizzato sostituisce il binding 20
Sostituisci con associazione personalizzata con espressioni regolari 21
Capitolo 6: Debug di un'applicazione knockout.js 22
Examples 22
Verifica del contesto di associazione di un elemento DOM 22
Stampa di un contesto vincolante dal markup 23
Capitolo 7: Equivalenti di attacchi AngularJS 25
Osservazioni 25
Examples 25
ngShow 25
ngBin (markup ricci) 25
ngModel su input [tipo = testo] 26
ngHide 26
ngClass 26
Capitolo 8: Href vincolante 27
Osservazioni 27
Examples 27
Uso di attr binding 27

Gestore di binding personalizzato ... 27
Capitolo 9: Introduzione ai componenti ... 28
Osservazioni ... 28
Examples ... 28
Barra di avanzamento (Boostrap) ... 28
Capitolo 10: Lavorare con knockout foreach vincolanti con JSON ... 29
Examples ... 29
Lavorare con il ciclo annidato ... 29
Capitolo 11: osservabili ... 31
Examples ... 31
Creare un osservabile ... 31
Sottoscrizione esplicita agli osservabili ... 31
Capitolo 12: Richieste AJAX e vincolanti ... 32
Examples ... 32
Esempio di richiesta AJAX con rilegatura ... 32
"Caricamento sezione / notifica" durante la richiesta AJAX ... 32
Titoli di coda ... 34

Di

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

It is an unofficial and free knockout.js 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 knockout.js.

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 knockout.js

Osservazioni

Questa sezione fornisce una panoramica di cosa sia knockout.js e del perché uno sviluppatore potrebbe volerlo utilizzare.

Dovrebbe anche menzionare qualsiasi argomento di grandi dimensioni all'interno di knockout.js e collegarsi agli argomenti correlati. Poiché la documentazione di knockout.js è nuova, potrebbe essere necessario creare versioni iniziali di tali argomenti correlati.

Versioni

Versione	Gli appunti	Data di rilascio
3.4.2	Correzioni di bug	2017/03/06
3.4.1	Correzioni di bug	2016/11/08
3.4.0		2015/11/17
3.3.0		2015/02/18
3.2.0	Introdotto il binding dei `component`	2014/08/12
3.1.0		2014/05/14
3.0.0	Vedi anche: upgrade (da 2.x) note	2013/10/25
2.3.0	Ultima 2.x release	2013/07/08
2.0.0		2011-12-21
1.2.1	Ultima versione 1.x.	2011-05-22
1.0.0		2010-07-05

Examples

Installazione o configurazione

Knockout è disponibile sulla maggior parte delle piattaforme JavaScript o come script autonomo.

Includi come script

Puoi scaricare lo script dalla sua pagina di download , quindi includerlo nella tua pagina con un `script tag` standard

```
<script type='text/javascript' src='knockout-3.4.0.js'></script>
```

Utilizzando un CDN

Puoi anche includere l'eliminazione diretta da Microsoft CDN o CDNJS

```
<script type='text/javascript' src='//ajax.aspnetcdn.com/ajax/knockout/knockout-
3.4.0.js'></script>
```

O

```
<script type='text/javascript' src='//cdnjs.cloudflare.com/ajax/libs/knockout/3.4.0/knockout-
min.js'></script>
```

Installa da npm

```
npm install knockout
```

opzionalmente è possibile aggiungere il parametro `--save` per mantenerlo nel file `package.json`

Installa da pergola

```
bower install knockout
```

opzionalmente è possibile aggiungere il parametro `--save` per mantenerlo nel file `bower.json`

Installa da NuGet

```
Install-Package knockoutjs
```

Per iniziare: Hello world!

Creazione di un documento HTML e attivazione di knockout.js

Crea un file HTML e includi `knockout.js` tramite un tag `<script>` .

```
<!DOCTYPE html>
```

```
<html>
<head>
    <title>Hello world! (knockout.js)</title>
</head>
<body>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/knockout/3.4.0/knockout-
debug.js"></script>
</body>
</html>
```

Aggiungi un secondo tag `<script>` *sotto* lo script ad eliminazione diretta. In questo tag script, inizializzeremo un modello di visualizzazione e applicheremo *i binding di dati* al nostro documento.

```
<script>
  var ViewModel = function() {
    this.greeting = ko.observable("Hello");
    this.name = ko.observable("World");

    this.appHeading = ko.pureComputed(function() {
      return this.greeting() + ", " + this.name() + "!";
    }, this);
  };

  var appVM = new ViewModel();

  ko.applyBindings(appVM);
</script>
```

Ora, continua a creare una *vista* aggiungendo del codice HTML al corpo:

```
<section>
  <h1 data-bind="text: appHeading"></h1>
  <p>Greeting: <input data-bind="textInput: greeting" /></p>
  <p>Name: <input data-bind="textInput: name" /></p>
</section>
```

Quando il documento HTML viene aperto e gli script vengono eseguiti, vedrai una pagina che dice **Hello, World!** . Quando cambi le parole negli elementi `<input>` , il testo `<h1>` viene aggiornato automaticamente.

Come funziona il file creato

1. Una versione di debug di knockout viene caricata da una fonte esterna (cdnjs)

Codice:

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/knockout/3.4.0/knockout-
debug.js"></script>
```

2. Viene creata un'istanza viewmodel con proprietà *osservabili* . Ciò significa che knockout è in grado di rilevare le modifiche e aggiornare l'interfaccia utente di conseguenza.

Codice:

```
var appVM = new ViewModel();
```

3. Knockout controlla il DOM per gli attributi di associazione dei `data-bind` e aggiorna l'interfaccia utente utilizzando il viewmodel fornito.

Codice:

```
ko.applyBindings(appVM);
```

4. Quando incontra un binding di `text` , knockout usa il valore della proprietà così come è definito nel viewmodel associato per iniettare un nodo di testo:

Codice:

```
<h1 data-bind="text: appHeading"></h1>
```

Osservabili calcolati

Le osservabili calcolate sono funzioni che possono "guardare" o "reagire" ad altri osservabili sul modello di vista. L'esempio seguente mostra come visualizzare il numero totale di utenti e l'età media.

*Nota: l'esempio seguente può anche fare uso di **pureComputed ()** (introdotto in v3.2.0) poiché la funzione calcola semplicemente qualcosa in base ad altre proprietà del modello di vista e restituisce un valore.*

```
<div>
  Total Users: <span data-bind="text: TotalUsers">2</span><br>
  Average Age: <span data-bind="text: UsersAverageAge">32</span>
</div>
```

```
var viewModel = function() {

    var self = this;

    this.Users = ko.observableArray([
        { Name: "John Doe", Age: 30 },
        { Name: "Jane Doe", Age: 34 }
    ]);

    this.TotalUsers = ko.computed(function() {
        return self.Users().length;
    });

    this.UsersAverageAge = ko.computed(function() {
        var totalAge = 0;
        self.Users().forEach(function(user) {
            totalAge += user.Age;
        });
```

```
        return totalAge / self.TotalUsers();
    });
};

ko.applyBindings(viewModel);
```

Leggi Iniziare con knockout.js online: https://riptutorial.com/it/knockout-js/topic/799/iniziare-con-knockout-js

Capitolo 2: Associazioni - Campi modulo

Examples

Clic

Il bind di `click` può essere utilizzato con qualsiasi elemento DOM visibile per aggiungere un gestore di eventi, che invocherà una funzione JavaScript, quando si fa clic sull'elemento.

```
<button data-bind="click: onClick">Click me</button>
```

```
ko.applyBindings({
  onClick: function(data, event) {
    // data: the context of the element that triggered the event
    console.log(data);

    // event: the click event
    console.log(event);
  }
});
```

Opzioni

Utilizza questa associazione per creare opzioni per un elemento selezionato

```
<select data-bind="options: gasGiants"></select>

<script type="text/javascript">
    var viewModel = {
        gasGiants: ko.observableArray(['Jupiter', 'Saturn', 'Neptune', 'Uranus'])
    };
</script>
```

È inoltre possibile utilizzare le proprietà all'interno della matrice per la visualizzazione nell'elenco e per il salvataggio in viewModel: `optionsText` consente di visualizzare un testo personalizzato

`optionsValue` imposta la proprietà value della corrispondente `<option>`

`value` memorizza il valore dell'opzione selezionata in un osservabile di viewModel

```
<select data-bind="options: gasGiants, optionsText:'name', optionsValue:'id',
value:selectedPlanetId"></select>

<script type="text/javascript">
    var viewModel = {
        selectedPlanetId: ko.observable(),
        gasGiants: ko.observableArray([{name:'Jupiter', id:'0'},
                                       {name:'Saturn', id:'1'},
                                       {name:'Neptune', id:'2'},
                                       {name:'Uranus', id:'3'}])
    };
```

```
</script>
```

Per memorizzare i risultati di un elenco di selezione multipla, le opzioni di rilegatura possono essere combinati con le `selectedOptions` vincolante.

```
<select data-bind="options: gasGiants, selectedOptions: chosenGasGiants"
multiple="true"></select>

<script type="text/javascript">
    var viewModel = {
        gasGiants: ko.observableArray(['Jupiter', 'Saturn', 'Neptune', 'Uranus'])
        chosenGasGiants: ko.observableArray(['Jupiter','Saturn']) // Initial selection
    }; </script>
```

disabilitato abilitato

L'associazione disabilitata aggiunge un attributo `disabled` a un elemento html, impedendo che sia più modificabile o cliccabile. Questo è utile principalmente per gli elementi `<input>` , `<select>` , `<textarea>` , `<a>` e `<button>`

```
<input data-bind="disabled: disableInput"/>

<script type="text/javascript">
var viewModel = {
    disableInput: ko.observable(true);
};
</script>
```

L'inverso dell'associazione `disabled` è `enabled`

La visibilità può anche essere calcolata utilizzando le funzioni JavaScript. Qualsiasi osservabile utilizzato in queste funzioni deve essere chiamato con parentesi

```
<script type="text/javascript">
var viewModel = {
    disableInput: ko.observable(true);
};
</script>
```

o

```
<input data-bind="disabled: allValues().length>4"/>

<script type="text/javascript">
var viewModel = {
    allValues: ko.observableArray([1,2,3,4,5]);
};
</script>
```

Sottoscrivi

Gestore di eventi da invocare quando viene inviato un elemento DOM.

```
<form data-bind="submit: doSomething">
    <!-- form content here -->
    <button type="submit"></button>
</form>

<script type="text/javascript">
    var vm = {
        doSomething: function(data){
            //do something here
        };
    }
</script>
```

Knockout impedirà l'azione di invio predefinita del browser per tale modulo. Se vuoi che il tuo modulo venga inviato come un normale modulo HTML, devi solo restituire `true` nel gestore di invio.

Valore

Usa il valore vincolante per ottenere il valore di un elemento. Il valore vincolante può essere applicato a qualsiasi controllo di modulo, tuttavia esistono altri collegamenti che potrebbero essere più adatti per caselle di controllo, pulsanti di opzione e input di testo.

Nell'esempio seguente viene illustrato come applicare l'elemento di associazione a diversi campi di input del modulo e come compilare i valori predefiniti:

Definizione di ViewModel:

```
var MyViewModel = function(){
    var self = this;
  //Initialize valueOne
  self.valueOne = ko.observable();
  //Initialize valueTwo with a default value of "Value two"
  self.valueTwo = ko.observable("Value two");
  //Initialize the color dropdown, and by default, select the "blue" option
  self.color = ko.observable("blue");

  self.valueOne.subscribe(function(newValue){
      console.log("valueOne: " + newValue);
  });

  self.valueTwo.subscribe(function(newValue){
      console.log("valueTwo: " + newValue);
  });

  self.color.subscribe(function(newValue){
      console.log("color: " + newValue);
  });
}
```

Markup associato:

```
<input type="text" data-bind="value: valueOne" />
<input type="text" data-bind="value: valueTwo" />
```

```
<select data-bind="value: color">
  <option value="red">Red</option>
  <option value="green">Green</option>
  <option value="blue">Blue</option>
</select>
```

Nell'esempio precedente, quando un valore cambia, il nuovo valore verrà registrato nella console. I valori iniziali non innescano un evento di modifica.

Per impostazione predefinita, il valore vincolante definisce una modifica come una modifica al valore degli elementi e lo stato attivo viene trasferito a un altro elemento. Questo può essere modificato usando l'opzione valueUpdate:

```
<input type="text" data-bind="value: valueOne, valueUpdate: 'keyup'" />
```

L'esempio sopra cambierà l'aggiornamento del valore da attivare alla chiave su. Le opzioni disponibili sono input, keyup, keypress e afterkeydown.

Leggi Associazioni - Campi modulo online: https://riptutorial.com/it/knockout-js/topic/7101/associazioni---campi-modulo

Capitolo 3: Associazioni: testo e aspetto

Examples

Testo

Il bind del `text` può essere usato con qualsiasi elemento per aggiornarlo nel suo innerText.

```
<p>
  <span data-bind="text: greeting"></span>,
  <span data-bind="text: subject"></span>.
</p>
```

```
ko.applyBindings({
  greeting: ko.observable("Hello"),
  subject: ko.observable("world")
});
```

Il binding del `text` può anche essere utilizzato con elementi virtuali.

```
<p>
  <!--ko text: greeting--><!--/ko-->,
  <!--ko text: subject--><!--/ko-->.
</p>
```

CSS

Questa associazione applicherà la classe CSS fornita all'elemento. Le classi statiche vengono applicate quando le condizioni date vengono valutate in modo approssimativo su true. Le classi dinamiche usano il valore di un osservabile o calcolato.

page.html

```
<p data-bind="css: { danger: isInDanger }">Checks external expression</p>
<p data-bind="css: { danger: dangerLevel() > 10 }">Expression can be inline</p>
<p data-bind="css: { danger: isInDanger, glow: shouldGlow }">Multiple classes</p>
<p data-bind="css: dynamicObservable">Dynamic CSS class from observable</p>
<p data-bind="css: dynamicComputed">Dynamic CSS class from computed</p>
```

page.js

```
ko.applyBindings({
  isInDanger: ko.observable(true),
  dangerLevel: ko.observable(5),
  isHot: ko.observable(true),
  shouldGlow: ko.observable(true),
  dynamicObservable: ko.observable('highlighted'),
  dynamicComputed: ko.computed(function() {
        var customClass = "";
        if(dangerLevel() >= 15 ) {
```

```
            customClass += " danger";
        }
        if(dangerLevel() >= 10) {
            customClass += " glow";
        }
        if(dangerLevel() >= 5) {
            customClass += " highlighted";
        }
        return customClass;
    });
});
```

page.css

```
.danger { background: red; }
.glow { box-shadow: 5px 5px 5px gold; }
.highlighted { color: purple; }
```

Vedi anche: documentazione ufficiale .

Visibile

Può essere usato per mostrare / nascondere elementi DOM. È simile all'utilizzo `if` , ad eccezione di quella `visible` verrà comunque creato l'elemento e impostato `display:none` .

```
<div data-bind="visible: shouldShowMessage">
    You will see this message only when "shouldShowMessage" holds a true value.
</div>

<script type="text/javascript">
    var viewModel = {
        shouldShowMessage: ko.observable(true);
    };
</script>
```

attr

Usa il binding `attr` per applicare eventuali attributi aggiuntivi al tuo elemento. Più comunemente usato per impostare un href, src o qualsiasi attributo di dati.

```
<img data-bind="attr: { src: url, title: title }"/>
```

```
var viewModel = {
    url: ko.observable("images/example.png"),
    title: "example title"
};
```

HTML

Questo legame aggiorna il innerHTML dell'elemento usando `jQuery.html()` , se jQuery è stato referenziato, altrimenti, la logica di analisi di KO. Può essere utile se si recupera l'HTML da un'API, un feed RSS, ecc. Prestare attenzione all'utilizzo di questo tag con l'HTML di input

dell'utente.

page.html

```
<p>
  <span data-bind="html: demoLink"></span>
</p>
```

page.js

```
ko.applyBindings({
  demoLink: ko.observable("<a href='#'>Make a link</a>")
});
```

Leggi Associazioni: testo e aspetto online: https://riptutorial.com/it/knockout-js/topic/7103/associazioni--testo-e-aspetto

Capitolo 4: Attacchi

Sintassi

- `<!-- ko if:myObservable --><!-- /ko -->`
- `<i data-bind="if:myObservable"></i>`

Osservazioni

Che legame c'è

In sostanza, un binding o un binding di dati è un modo per collegare ViewModels ai tuoi Views (template) e viceversa. KnockoutJS utilizza l'associazione dati bidirezionale, il che significa che le modifiche al ViewModel influenzano la vista e le modifiche alla vista possono influenzare ViewModel.

Sotto il cofano (breve panoramica)

I binding sono solo plug-in (script) che ti consentono di risolvere un particolare compito. Questo compito è più frequente che cambiare markup (html) in base al ViewModel.

Ad esempio, un'associazione di `text` ti consente di visualizzare il testo e cambiarlo dinamicamente ogni volta che ViewModel cambia.

KnockoutJS è dotato di molti potenti binding e ti consente di estenderlo con i tuoi binding personalizzati.

E soprattutto i collegamenti non sono magici, funzionano secondo una serie di regole e ogni volta che non sei sicuro di ciò che fa un binding, quali parametri ci vogliono o quando aggiornerà la vista puoi fare riferimento al codice sorgente del binding.

Considera il seguente esempio di associazione personalizzata:

```
ko.bindingHandlers.debug = {
    init: function (element, valueAccessor, allBindingsAccessor, viewModel, bindingContext) {
        ko.computed(function () {
            var value = ko.unwrap(valueAccessor());

            console.log({
                value: value,
                viewModel: viewModel,
                bindingContext: bindingContext
            });
        }, null, { disposeWhenNodeIsRemoved: element });
    }
};
```

0. Un'associazione ha un nome - `debug` così puoi usare come segue:

```
data-bind="debug: 'foo'"
```

1. Il metodo `init` viene chiamato una volta quando viene avviata l'associazione. Il resto degli aggiornamenti è gestito da un anonimo calcolato che viene eliminato quando l' `element` viene rimosso.
2. Il binding stampa per gestire diverse cose: il valore passato nel nostro esempio questo valore è `foo` (questo valore può anche essere osservabile poiché il metodo `ko.unwrap` è usato per leggerlo), il viewModel corrente e il bindingContext corrente.
3. Ogni volta che il valore passato cambia, l'associazione stamperà le informazioni aggiornate sulla console.
4. Questa associazione non può essere utilizzata con elementi virtuali (nei commenti html), solo su elementi reali, poiché il flag `ko.virtualElements.allowedBindings.debug` non è impostato su true.

Quando usare le parentesi

Senza plug-in aggiuntivi, KnockoutJS avrà solo aggiornamenti live View per le proprietà sul ViewModel che sono *osservabili* (`observable` regolarmente, ma anche `computed` , `pureComputed` , `observableArray` , ecc.). Un osservabile sarebbe creato in questo modo:

```
var vm = { name: ko.observable("John") };
```

In questo caso, `vm.name` è una *funzione* con due "modi" separati:

1. Getter: `vm.name()` , senza argomenti, otterrà il valore corrente;
2. Setter: `vm.name("Johnnyboy")` , con un argomento, imposterà un nuovo valore.

Nei collegamenti dati incorporati è *sempre* possibile utilizzare il modulo getter e *talvolta* si possono effettivamente *omettere* le parentesi e il binding le "aggiunge" in modo efficace. Quindi questi sono equivalenti:

```
<p data-bind="text: name"></p> ... will work
<p data-bind="text: name()"></p> ... works too
```

Ma questo fallirà:

```
<p data-bind="text: 'Hello, ' + name + '!'"></p> ... FAILS!
```

Perché non appena si vuole "fare" qualcosa prima di passare un valore a un'associazione dati, compresi i confronti di valori, è necessario "ottenere" correttamente i valori per tutti gli osservabili, ad esempio:

```
<p data-bind="text: 'Hello, ' + name() + '!'"></p> ... works
```

Vedi anche questo Q & A per maggiori dettagli.

Examples

Se / se non

È possibile utilizzare il legame `if` per determinare se gli elementi figlio del nodo devono essere creati o meno.

```
<div class="product-info">
    <h2>  Product1  </h2>
    <img src="../products/product1.jpg"/>
    <span data-bind="if:featured">
        <span class="featured"></span>
    </span>
    <span data-bind="ifnot:inStock">
        <span class="out-of-stock"></span>
    </span>
</div>

<script>
    ko.applyBindings({
        product: {
            featured: ko.observable(true),
            inStock: ko.observable(false)
        }
    });
</script>
```

L'inverso di `if` binding è `ifnot`

```
<div data-bind="ifnot: someProperty">...</div>
```

è equivalente a

```
<div data-bind="if: !someProperty()">...</div>
```

A volte, non dovrai controllare la presenza di elementi senza dover creare un contenitore (in genere per gli elementi `<li>` in un elemento `<ul>` o `<option>` all'interno di `<select>`)

Knockout abilita questo con la sintassi del flusso di controllo senza contenitore in base ai tag di commento in questo modo:

```
<select>
  <option value="0">fixed option</option>
<!-- ko if: featured-->
  <option value="1">featured option</option>
<!-- /ko -->
</select>
```

Per ciascuno

Simile ai ripetitori utilizzati in altre lingue. Questa associazione ti consentirà di replicare un blocco

di html per ogni elemento in una matrice.

```
<div data-bind="foreach:contacts">
    <div class="contact">
        <h2 data-bind="text:name">
        <p data-bind="text:info">
    </div>
</div>

<script type="text/javascript">
    var contactViewModel = function (data) {
        this.name = ko.observable(data.name);
        this.info = ko.observable(data.info);
    };

    ko.applyBindings({
        contacts: [
            new contactViewModel({name:'Erik', info:'Erik@gmail.com'}),
            new contactViewModel({name:'Audrey', info:'Audrey@gmail.com'})
        ]
    });
</script>
```

Si noti che quando eseguiamo il looping del nostro contesto diventa l'elemento all'interno dell'array, in questo caso un'istanza di `contactViewModel` . All'interno di un `foreach` abbiamo anche accesso a

- `$parent` - il modello di visualizzazione che ha creato questa associazione
- `$root` - il modello di visualizzazione radice (potrebbe anche essere genitore)
- `$data` - i dati a questo indice dell'array
- `$index` - l' `$index` (osservabile) a base zero dell'oggetto reso

Con

Il binding `with` associa l'HTML all'interno del nodo associato a un contesto separato:

```
<div data-bind="with: subViewModel">
  <p data-bind="text: title"></p>
</div>
```

L'associazione `with` associazione può anche essere utilizzata senza un elemento contenitore in cui un elemento contenitore potrebbe non essere appropriato.

```
<!-- ko with: subViewModel -->
  <p data-bind="text: title"></p>
<!-- /ko -->
```

```
var vm = {
  subViewModel: ko.observable()
};

// Doesn't throw an error on the `text: title`; the `<p>` element
// isn't bound to any context (and even removed from the DOM)
ko.applyBindings(vm);
```

```
// Includes the `<p>` element and binds it to our new object
vm.subViewModel({ title: "SubViewModel" });
```

Il binding `with` ha molte somiglianze con il `template` o `foreach` binding.

Visibile

L'associazione `visible` nasconderà un elemento applicando `style="display: none;"` ad esso quando l'associazione valuta come falsa.

```
<input type="text" data-bind="textInput: name"> <span class="error" data-bind="visible:
isInvalid">Required!</span>

ko.applyBindings(new ViewModel());

function ViewModel(){
    var vm = this;
  vm.name = ko.observable("test");
  vm.isInvalid = ko.computed(function() {
      return vm.name().length == 0;
  });
}
```

jsFiddle

Leggi Attacchi online: https://riptutorial.com/it/knockout-js/topic/2249/attacchi

Capitolo 5: Binding personalizzati

Examples

Registrazione vincolante

I binding Cusom devono essere registrati estendendo l'attuale oggetto knockout bindingHandlers. Questo viene fatto aggiungendo una nuova proprietà all'oggetto.

```
ko.bindingHandlers.newBinding = {
    init: function(element, valueAccessor, allBindings, viewModel, bindingContext) {
    },
    update: function(element, valueAccessor, allBindings, viewModel, bindingContext) {
    }
};
```

Associazione di dissolvenza in dissolvenza in entrata / uscita

Questo esempio implementa un'associazione personalizzata che attiva la visibilità (simile all'attacco visibile esistente), ma utilizzerà l'API di fading di jQuery per animare la transizione da visibile a invisibile.

Definizione del binding personalizzato:

```
//Add a custom binding called "fadeVisible" by adding it as a property of ko.bindingHandlers
ko.bindingHandlers.fadeVisible = {
  //On initialization, check to see if bound element should be hidden by default
  'init': function(element, valueAccessor, allBindings, viewModel, bindingContext){
    var show = ko.utils.unwrapObservable(valueAccessor());
    if(!show){
      element.style.display = 'none';
    }
  },
  //On update, see if fade in/fade out should be triggered. Factor in current visibility
  'update': function(element, valueAccessor, allBindings, viewModel, bindingContext) {
    var show = ko.utils.unwrapObservable(valueAccessor());
    var isVisible = !(element.style.display == "none");

    if (show && !isVisible){
      $(element).fadeIn(750);
    }else if(!show && isVisible){
      $(element).fadeOut(750);
    }
  }
}
```

Markup di esempio con il binding fadeVisible:

```
<div data-bind="fadeVisible: showHidden()">
  Field 1: <input type="text" name="value1" />
  <br />
```

```
  Field 2: <input type="text" name="value2" />
</div>
<input data-bind="checked: showHidden" type="checkbox"/> Show hidden
```

Esempio di modello di visualizzazione:

```
var ViewModel = function(){
    var self = this;
    self.showHidden = ko.observable(false);
}


ko.applyBindings(new ViewModel());
```

Il testo personalizzato sostituisce il binding

Questo esempio è un binding personalizzato che sostituisce il testo ogni volta che viene aggiornato un valore di input. In questo caso, gli spazi saranno sostituiti con "+". È progettato per essere utilizzato insieme al valore esistente e mostra il binding con un oggetto letterale.

Esempio di markup con il binding replaceText:

```
<input type="text" data-bind="value: myField, replaceText: {value: myField, find:' ',
replace:'+'}" />
```

Definizione del binding personalizzato:

```
ko.bindingHandlers.replaceText = {

   //On update, grab the current value and replace text
  'update': function(element, valueAccessor, allBindings, viewModel, bindingContext) {

      //Get the current value of the input
      var val = ko.utils.unwrapObservable(valueAccessor().value());

      //Replace text using passed in values obtained from valueAccessor()
      //Note - Consider using something like string.js to do the find and replace
      var replacedValue = val.split(valueAccessor().find).join(valueAccessor().replace);

      //Set new value
      valueAccessor().value(replacedValue);
  }
}
```

Esempio di modello di visualizzazione:

```
var ViewModel = function(){
  var self = this;
  self.myField = ko.observable("this is a simple test");
}


ko.applyBindings(new ViewModel());
```

Sostituisci con associazione personalizzata con espressioni regolari

Definizione del binding personalizzato

```
function regExReplace(element, valueAccessor, allBindingsAccessor, viewModel, bindingContext)
{

  var observable = valueAccessor();
  var textToReplace = allBindingsAccessor().textToReplace || '';
  var pattern = allBindingsAccessor().pattern || '';
  var flags = allBindingsAccessor().flags;
  var text = ko.utils.unwrapObservable(valueAccessor());
  if (!text) return;
  var textReplaced = text.replace(new RegExp(pattern, flags), textToReplace);

  observable(textReplaced);
}

ko.bindingHandlers.regExReplace = {
  init: regExReplace,
  update: regExReplace
}
```

uso

ViewModel

```
ko.applyBindings({
  name: ko.observable(),
  num: ko.observable()
});
```

vista

```
<input type="text" data-bind="textInput : name, regExReplace:name, pattern:'(^[^a-zA-
Z]*)|(\\W)',flags:'g'" placeholder="Enter a valid name" />
<span data-bind="text : name"></span>
<br/>
<input class=" form-control " type="text " data-bind="textInput : num, regExReplace:num,
pattern: '[^0-9]',flags: 'g' " placeholder="Enter a number " />
<span data-bind="text : num"></span>
```

Leggi Binding personalizzati online: https://riptutorial.com/it/knockout-js/topic/6332/binding-personalizzati

Capitolo 6: Debug di un'applicazione knockout.js

Examples

Verifica del contesto di associazione di un elemento DOM

Molti bug nei legami dei dati knockout sono causati da proprietà non definite in un modello viewmodel. Knockout ha due metodi pratici per recuperare il contesto di bind di un elemento HTML:

```
// Returns the binding context to which an HTMLElement is bound
ko.contextFor(element);

// Returns the viewmodel to which an HTMLElement is bound
// similar to: ko.contextFor(element).$data
ko.dataFor(element);
```

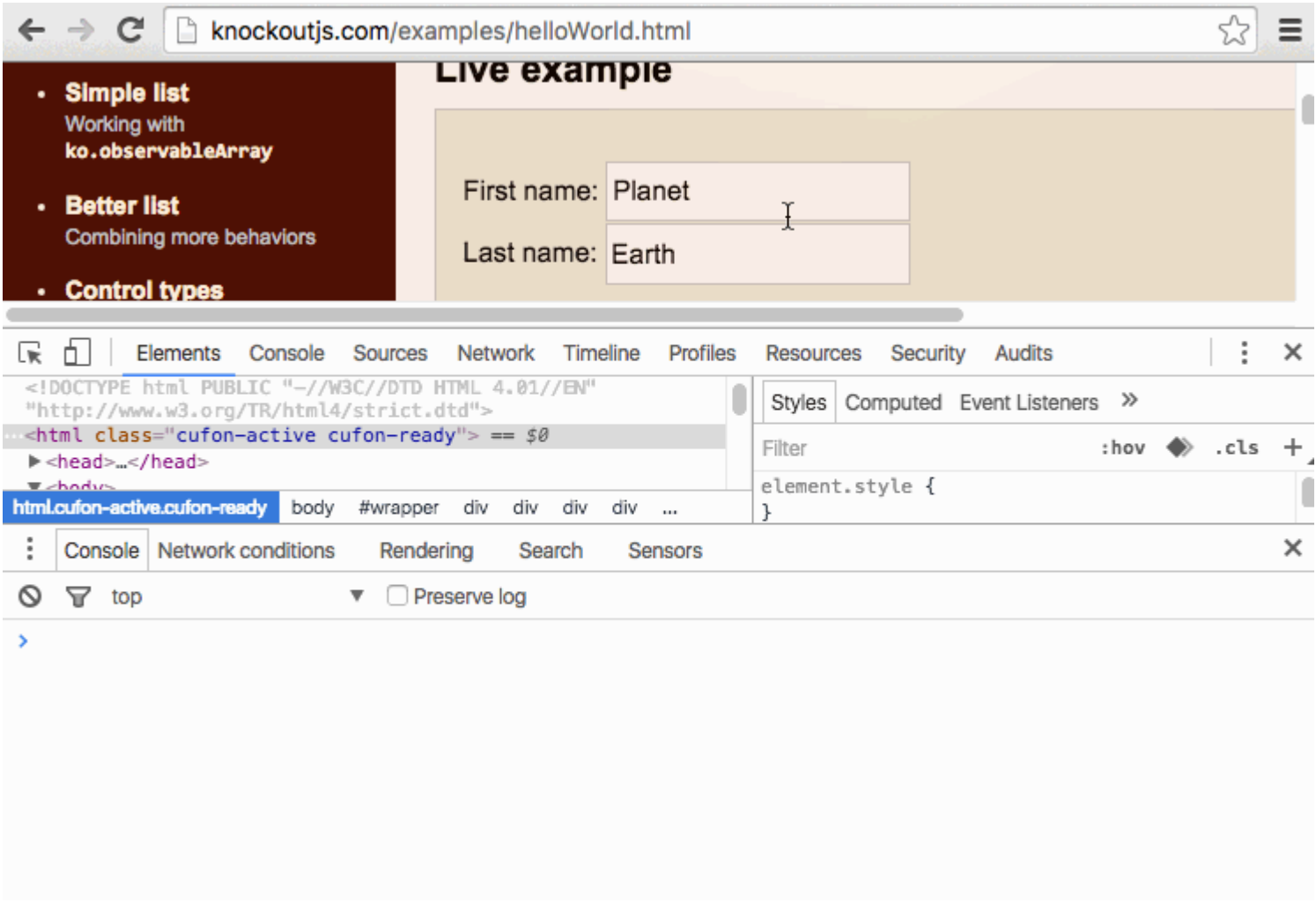
Per scoprire rapidamente il contesto vincolante di un elemento dell'interfaccia utente, ecco un utile trucco:

La maggior parte dei browser moderni memorizza l'elemento DOM attualmente selezionato in una variabile globale: `$0` (maggiori informazioni su questo meccanismo)

- Fai clic con il pulsante destro del mouse su un elemento nell'interfaccia utente e scegli *"inspect"* o *"inspect element"* nel menu di scelta rapida.
- digita `ko.dataFor($0)` nella console degli sviluppatori e premi `ko.dataFor($0)`

Esistono anche plug-in del browser che possono aiutare a trovare il contesto dell'oggetto.

Un esempio (provalo su esempio ciao mondo Knockout):



Stampa di un contesto vincolante dal markup

A volte è utile stampare un'associazione corrente direttamente dal markup. Un trucco accurato che consente di utilizzare un elemento DOM aggiuntivo con un binding non esistente (KO <3.0), un binding personalizzato o un'associazione che non è rilevante come `uniqueName` .

Considera questo esempio:

```
<tbody data-bind="foreach: people">
    <tr>
         <td data-bind="text: firstName"></td>
         <td data-bind="text: lastName"></td>
    </tr>
</tbody>
```

Se uno vorrebbe scoprire il contesto vincolante di ogni elemento nell'array di persone, si può scrivere:

```
<tbody data-bind="foreach: people">
    <span data-bind="uniqueName: console.log($data)"></span>
    <tr>
        <td data-bind="text: firstName"></td>
        <td data-bind="text: lastName"></td>
    </tr>
```

```
</tbody>
```

Leggi Debug di un'applicazione knockout.js online: https://riptutorial.com/it/knockout-js/topic/5066/debug-di-un-applicazione-knockout-js

Capitolo 7: Equivalenti di attacchi AngularJS

Osservazioni

Non tutto in AngularJS ha un equivalente KnockoutJS (ad esempio `ngCloack` o `ngSrc`). Esistono generalmente due soluzioni principali:

1. Utilizzare invece il collegamento generico `attr` o `event` .
2. Analogamente alle direttive personalizzate in AngularJS, puoi scrivere il tuo gestore di binding personalizzato se hai bisogno di qualcosa che non è incluso nella libreria di base.

Se preferisci la sintassi dell'associazione AngularJS, puoi prendere in considerazione l'utilizzo di Knockout.Punch che consente il binding in stile manubrio.

Examples

ngShow

Codice AngularJS per mostrare / nascondere un elemento in modo dinamico:

```
<p ng-show="SomeScopeProperty">This is conditionally shown.</p>
```

Equivalente KnockoutJS:

```
<p data-bind="visible: SomeScopeObservable">This is conditionally shown.</p>
```

ngBin (markup ricci)

Codice AngularJS per il rendering di testo normale:

```
<p>{{ ScopePropertyX }} and {{ ScopePropertyY }}</p>
```

Equivalente KnockoutJS:

```
<p>
  <!-- ko text: ScopeObservableX --><!-- /ko -->
  and
  <!-- ko text: ScopeObservableY --><!-- /ko -->
</p>
```

o:

```
<p>
  <span data-bind="text: ScopeObservableX"></span>
  and
  <span data-bind="text: ScopeObservableY"></span>
```

```
</p>
```

ngModel su input [tipo = testo]

Codice AngularJS per il bind a due vie su un `input` testo:

```
<input ng-model="ScopePropertyX" type="text" />
```

Equivalente KnockoutJS:

```
<input data-bind="textInput: ScopeObservableX" type="text" />
```

ngHide

Non esiste *un* legame equivalente *diretto* in KnockoutJS. Tuttavia, poiché nascondere è esattamente l'opposto di mostrare, possiamo semplicemente invertire l'esempio per l'equivalente di `ngShow` di Knockout .

```
<p ng-hide="SomeScopeProperty">This is conditionally shown.</p>
```

Equivalente KnockoutJS:

```
<p data-bind="visible: !SomeScopeObservable()">This is conditionally hidden.</p>
```

Il precedente esempio KnockoutJS presuppone che `SomeScopeObservable` sia osservabile, e poiché lo usiamo in un'espressione (a causa dell'operatore `!` fronte ad esso) non possiamo omettere il `()` alla fine.

ngClass

Codice AngularJS per classi dinamiche:

```
<p ng-class="{ highlighted: scopeVariableX, 'has-error': scopeVariableY }">Text.</p>
```

Equivalente KnockoutJS:

```
<p data-bind="css: { highlighted: scopeObservableX, 'has-error': scopeObservableY }">Text.</p>
```

Leggi Equivalenti di attacchi AngularJS online: https://riptutorial.com/it/knockout-js/topic/2408/equivalenti-di-attacchi-angularjs

Capitolo 8: Href vincolante

Osservazioni

Non esiste un vincolo `href` nella libreria core KnockoutJS, che è la ragione per cui tutti gli esempi mostrano *altre* funzionalità della libreria per ottenere lo stesso effetto.

Vedi anche questa domanda sull'overflow dello stack sullo stesso argomento .

Examples

Uso di attr binding

```
<a data-bind="attr: { href: myUrl }">link with dynamic href</a>
```

```
ko.applyBindings({
  myUrl: ko.observable("http://www.stackoverflow.com")
});
```

Poiché non esiste un binding `href` nativo in KnockoutJS, è necessario utilizzare una funzionalità diversa per ottenere collegamenti dinamici. L'esempio sopra mostra il legame `attr` incorporato per ottenere un collegamento dinamico.

Gestore di binding personalizzato

`href` collegamento `href` non è nativo di KnockoutJS, quindi per ottenere collegamenti dinamici utilizzare un gestore di binding personalizzato:

```
<a data-bind="href: myUrl">link with dynamic href</a>
```

```
ko.bindingHandlers['href'] = {
  update: function(element, valueAccessor) {
    element.href = ko.utils.unwrapObservable(valueAccessor());
  }
};
```

Leggi Href vincolante online: https://riptutorial.com/it/knockout-js/topic/6582/href-vincolante

Capitolo 9: Introduzione ai componenti

Osservazioni

I componenti consentono controlli / widget riutilizzabili rappresentati dalla propria vista (modello) e dal modello viewmodel. Sono stati aggiunti in Knockout 3.2. Ispirato da WebComponents, Knockout consente ai componenti di essere definiti come elementi personalizzati, consentendo l'uso di un markup autoesplicativo.

Examples

Barra di avanzamento (Boostrap)

Definizione del componente

```
ko.components.register('progress-bar', {
    viewModel: function(params) {
            var that = this;

        // progress is a numeric value between 0 and 100
        that.progress = params.progress;


        that.progressPercentual = ko.computed(function(){
            return '' + ko.utils.unwrapObservable(that.progress) + '%';
        })
    },
    template:
        '<div class="progress"> <div data-bind="attr:{\'aria-valuenow\':progress},
style:{width:progressPercentual}, text:progressPercentual" class="progress-bar"
role="progressbar" aria-valuenow="0" aria-valuemin="0" aria-valuemax="100" style="min-width:
2em;"></div> </div>'
});
```

Utilizzo HTML

```
<progress-bar params="progress:5"></progress-bar>
```

Leggi Introduzione ai componenti online: https://riptutorial.com/it/knockout-js/topic/6207/introduzione-ai-componenti

Capitolo 10: Lavorare con knockout foreach vincolanti con JSON

Examples

Lavorare con il ciclo annidato

Ecco la struttura JSON che useremo.

```
{
  "employees": [
    {
      "firstName": "John",
      "lastName": "Doe",
      "skills": [
        {
          "name": "javascript",
          "rating": 5
        }
      ]
    },
    {
      "firstName": "Anna",
      "lastName": "Smith",
      "skills": [
        {
          "name": "css",
          "rating": 5
        },
        {
          "name": "javascript",
          "rating": 5
        }
      ]
    },
    {
      "firstName": "Peter",
      "lastName": "Jones",
      "skills": [
        {
          "name": "html",
          "rating": 5
        },
        {
          "name": "javascript",
          "rating": 3
        }
      ]
    }
  ]
};
```

Questa struttura json può essere assegnata a una variabile o può essere una risposta di qualsiasi api.

Come possiamo vedere in questo JSON c'è un nodo esterno che contiene informazioni su di loro, e c'è un nodo interno che racconta le abilità di ciascun dipendente.

quindi qui creeremo un nidificato per ognuno utilizzando forca knockout. Ecco l'html

```
<ul id="employee" data-bind="foreach: employee">
   <li data-bind="text:firstName + ' ' + lastName">
   </li>
   <ul data-bind="foreach : skills">
        <li data-bind="text: name">
        </li>
      <ul>
        <li>
          Rating : <!-- ko text: rating --><!-- /ko -->
        </li>
      </ul>
   </ul>
</ul>
```

Qui nel codice HTML sopra ci sono due liste che contengono il ciclo foreach. anello esterno terrà il nodo esterno della struttura JSON che è dipendenti. Il ciclo interno mantiene le abilità di ciascun dipendente. All'interno di ciascun ciclo possiamo accedere alle proprietà del nodo corrispondente. Per un esempio possiamo accedere al nome e alla valutazione all'interno del ciclo di competenze non dall'esterno.

Di seguito è riportato il codice javascript.

```
var employeeViewModel = function(){
  var self = this;
  self.employee = ko.observableArray(employees);//here we can assign json
}
var viewModel = new employeeViewModel();
ko.applyBindings(viewModel);
```

Formare il punto di vista javascript non c'è molto codice. possiamo assegnare direttamente il nostro json a un observablearray che verrà utilizzato da Html.

Leggi Lavorare con knockout foreach vincolanti con JSON online:
https://riptutorial.com/it/knockout-js/topic/6961/lavorare-con-knockout-foreach-vincolanti-con-json

Capitolo 11: osservabili

Examples

Creare un osservabile

JS

```
// data model
var person = {
    name: ko.observable('Jack'),
    age: ko.observable(29)
};

ko.applyBindings(person);
```

HTML

```
<div>
    <p>Name: <input data-bind='value: name' /></p>
    <p>Age: <input data-bind='value: age' /></p>
    <h2>Hello, <span data-bind='text: name'> </span>!</h2>
</div>
```

Sottoscrizione esplicita agli osservabili

```
var person = {
  name: ko.observable('John')
};

console.log(person.name());

console.log('Update name');

person.name.subscribe(function(newValue) {
  console.log("Updated value is " + newValue);
});

person.name('Jane');
```

Leggi osservabili online: https://riptutorial.com/it/knockout-js/topic/6363/osservabili

Capitolo 12: Richieste AJAX e vincolanti

Examples

Esempio di richiesta AJAX con rilegatura

page.html

```
<div data-bind="foreach: blogs">
    <br />
    <span data-bind="text: entryPostedDate"></span>
    <br />
    <h3>
        <a data-bind="attr: { href: blogEntryLink }, text: title"></a>
    </h3>
    <br /><br />
    <span data-bind="html: body"></span>
    <br />
    <hr />
    <br />
</div>

<!--- include knockout and dependencies (Jquery) --->
<script type="text/javascript" src="blog.js"></script>
```

blog.js

```
function vm() {
    var self = this;

    // Properties
    self.blogs = ko.observableArray([]);

    // consists of entryPostedDate, blogEntryLink, title, body
    var blogApi = "/api/blog";

    // Load data
    $.getJSON(blogApi)
        .success(function (data) {
            self.blogs(data);
        });
    }
ko.applyBindings(new vm());
```

Si noti che JQuery è stato utilizzato (`$.getJSON(...)`) per eseguire la richiesta. JavaScript vanilla può eseguire lo stesso, anche se con più codice.

"Caricamento sezione / notifica" durante la richiesta AJAX

Blog.html

```
<div data-bind="visible: isLoading()">
```

```
    Loading...
</div>

<div data-bind="visible: !isLoading(), foreach: blogs">
    <br />
    <span data-bind="text: entryPostedDate"></span>
    <br />
    <h3>
        <a data-bind="attr: { href: blogEntryLink }, text: title"></a>
    </h3>
    <br /><br />
    <span data-bind="html: body"></span>
    <br />
    <hr />
    <br />
</div>

<!--- include knockout and dependencies (JQuery) --->
<script type="text/javascript" src="blog.js"></script>
```

blog.js

```
function vm() {
    var self = this;

    // Properties
    self.blogs = ko.observableArray([]);
    self.isLoading = ko.observable(true);

    // consists of entryPostedDate, blogEntryLink, title, body
    var blogApi = "/api/blog";

    // Load data
    $.getJSON(blogApi)
        .success(function (data) {
            self.blogs(data);
        })
        .complete(function () {
            self.isLoading(false); // on complete, set loading to false, which will hide
"Loading..." and show the content.
        });
}
ko.applyBindings(new vm());
```

Si noti che JQuery è stato utilizzato ($ .getJSON (...)) per eseguire la richiesta. JavaScript vanilla può eseguire lo stesso, anche se con più codice.

Leggi Richieste AJAX e vincolanti online: https://riptutorial.com/it/knockout-js/topic/7538/richieste-ajax-e-vincolanti

Titoli di coda

S. No	Capitoli	Contributors
1	Iniziare con knockout.js	ASindleMouat, aswallows, Community, Jeroen, Michael Best, Ray, Ross Vernal, Tyrsius, user3297291
2	Associazioni - Campi modulo	Knekki, Matthias Lloyd, Nick DeFazio, stackoverfloweth, Steffen Nieuwenhoven, tmg, user3297291
3	Associazioni: testo e aspetto	CreationEdge, Jeroen, Kritner, Nick DeFazio, stackoverfloweth, tmg
4	Attacchi	CreationEdge, dotnetom, Homer, Jeroen, Knekki, Kritner, Matthias Lloyd, natus, Nick DeFazio, Olga, stackoverfloweth, Steffen Nieuwenhoven, tmg, user3297291
5	Binding personalizzati	AldoRomo88, natus, Nick DeFazio
6	Debug di un'applicazione knockout.js	Adam Wolski, AldoRomo88, natus, user3297291
7	Equivalenti di attacchi AngularJS	Jeroen, Quango
8	Href vincolante	4444, Jeroen
9	Introduzione ai componenti	4444, AldoRomo88, ASindleMouat
10	Lavorare con knockout foreach vincolanti con JSON	suyesh
11	osservabili	Arun S, Norbert
12	Richieste AJAX e vincolanti	Kritner