eBook Gratuit

APPRENEZ knockout.js

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#knockout.j

s

Table des matières

À propos

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

Chapitre 1: Démarrer avec knockout.js

Remarques

Cette section fournit une vue d'ensemble de ce que knockout.js est et pourquoi un développeur peut vouloir l'utiliser.

Il devrait également mentionner tous les grands sujets dans knockout.js, et établir un lien avec les sujets connexes. Comme la documentation de knockout.js est nouvelle, vous devrez peut-être créer des versions initiales de ces rubriques connexes.

Versions

Version	Remarques	Date de sortie
3.4.2	Corrections de bugs	2017-03-06
3.4.1	Corrections de bugs	2016-11-08
3.4.0		2015-11-17
3.3.0		2015-02-18
3.2.0	Liaison de `component` introduite	2014-08-12
3.1.0		2014-05-14
3.0.0	Voir aussi: mise à niveau (à partir de 2.x) notes	2013-10-25
2.3.0	Dernière version 2.x	2013-07-08
2.0.0		2011-12-21
1.2.1	Dernière version 1.x	2011-05-22
1.0.0		2010-07-05

Examples

Installation ou configuration

Knockout est disponible sur la plupart des plates-formes JavaScript ou en tant que script autonome.

Inclure comme script

Vous pouvez télécharger le script depuis sa page de téléchargement , puis l'inclure dans votre page avec une `script tag` standard

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

Utiliser un CDN

Vous pouvez également inclure des knock-out du CDN Microsoft ou CDNJS

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

OU

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

Installer à partir de npm

```
npm install knockout
```

facultativement, vous pouvez ajouter le paramètre `--save` pour le conserver dans votre fichier `package.json`

Installer à partir de bower

```
bower install knockout
```

facultativement, vous pouvez ajouter le paramètre `--save` pour le conserver dans votre fichier `bower.json`

Installer depuis NuGet

```
Install-Package knockoutjs
```

Pour commencer: Bonjour tout le monde!

Créer un document HTML et activer knockout.js

Créez un fichier HTML et incluez `knockout.js` via une `<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>
```

Ajoutez une seconde `<script>` *sous* le script knockout. Dans cette balise de script, nous allons initialiser un modèle de vue et appliquer des *données liées* à notre document.

```
<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>
```

Maintenant, continuez à créer une *vue* en ajoutant du HTML au corps:

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

Lorsque le document HTML est ouvert et que les scripts sont exécutés, vous verrez une page indiquant **Bonjour, Monde!** . Lorsque vous modifiez les mots dans les éléments `<input>` , le texte `<h1>` est automatiquement mis à jour.

Comment le fichier créé fonctionne

1. Une version de débogage de knockout est chargée depuis une source externe (cdnjs)

Code:

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

2. Une instance viewmodel est créée et possède *des* propriétés *observables* . Cela signifie que KO peut détecter les modifications et mettre à jour l'interface utilisateur en conséquence.

Code:

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

3. Knockout vérifie les attributs de `data-bind` de `data-bind` du DOM et met à jour l'interface utilisateur à l'aide du modèle de vue fourni.

Code:

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

4. Lorsqu'il rencontre une liaison de `text` , le masquage utilise la valeur de la propriété telle qu'elle est définie dans le modèle de vue lié pour injecter un nœud de texte:

Code:

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

Observables Calculés

Les observables calculés sont des fonctions qui peuvent "regarder" ou "réagir" à d'autres observables du modèle de vue. L'exemple suivant montre comment afficher le nombre total d'utilisateurs et l'âge moyen.

*Remarque: L'exemple ci-dessous peut également utiliser **pureComputed ()** (introduit dans la version 3.2.0) car la fonction calcule simplement quelque chose en fonction d'autres propriétés du modèle de vue et renvoie une valeur.*

```
<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);
```

Lire Démarrer avec knockout.js en ligne: https://riptutorial.com/fr/knockout-js/topic/799/demarrer-avec-knockout-js

Chapitre 2: Déboguer une application knockout.js

Examples

Vérification du contexte de liaison d'un élément DOM

De nombreux bogues dans les liaisons de données à exclure sont causés par des propriétés non définies dans un modèle de vue. Knockout a deux méthodes pratiques pour récupérer le contexte de liaison d'un élément 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);
```

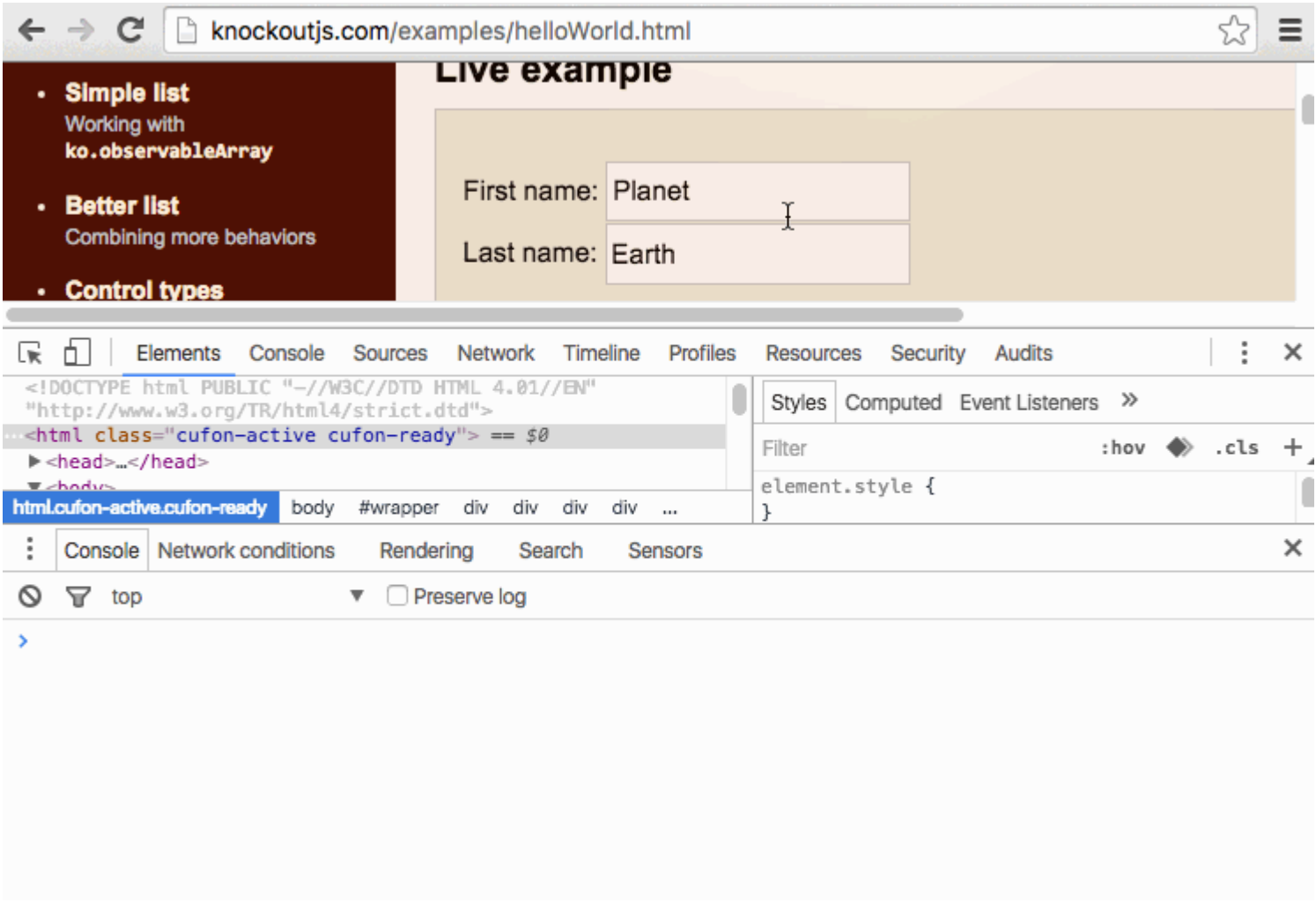
Pour trouver rapidement le contexte de liaison d'un élément d'interface utilisateur, voici une astuce pratique:

La plupart des navigateurs modernes stockent l'élément DOM actuellement sélectionné dans une variable globale: `$0` (plus sur ce mécanisme)

- Cliquez avec le bouton droit sur un élément de votre interface utilisateur et choisissez *"inspecter"* ou *"inspecter l'élément"* dans le menu contextuel.
- tapez `ko.dataFor($0)` dans la console du développeur et appuyez sur Entrée

Il existe également des plug-ins de navigateur qui peuvent aider à trouver le contexte de l'objet.

Un exemple (essayez-le sur Knockout hello world exemple):



Impression d'un contexte de liaison à partir du balisage

Parfois, il est utile d'imprimer une liaison en cours directement à partir du balisage. Une astuce qui permet d'utiliser un élément DOM supplémentaire avec une liaison non existante (KO <3.0), une liaison personnalisée ou une liaison non pertinente telle que `uniqueName` .

Considérez cet exemple:

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

Si l'on veut connaître le contexte de liaison de chaque élément du tableau de personnes, on peut écrire:

```
<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>
```

Lire Déboguer une application knockout.js en ligne: https://riptutorial.com/fr/knockout-js/topic/5066/deboguer-une-application-knockout-js

Chapitre 3: Demandes AJAX et liaison

Examples

Exemple de requête AJAX w / binding

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());
```

Notez que JQuery a été utilisé (`$.getJSON(...)`) pour effectuer la requête. vanilla JavaScript peut effectuer la même chose, mais avec plus de code.

"Loading section / notification" pendant la requête 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());
```

Notez que JQuery a été utilisé ($ .getJSON (...)) pour effectuer la requête. vanilla JavaScript peut effectuer la même chose, mais avec plus de code.

Lire Demandes AJAX et liaison en ligne: https://riptutorial.com/fr/knockout-js/topic/7538/demandes-ajax-et-liaison

Chapitre 4: Equivalents des liaisons AngularJS

Remarques

Tout dans AngularJS ne possède pas un équivalent KnockoutJS (par exemple `ngCloack` ou `ngSrc`). Deux solutions principales sont généralement disponibles:

1. Utilisez plutôt l' `attr` générique `attr` ou l' `event` .
2. Semblable aux directives personnalisées dans AngularJS, vous pouvez écrire votre propre gestionnaire de liaison personnalisé si vous avez besoin de quelque chose qui n'est pas inclus dans la bibliothèque de base.

Si vous préférez la syntaxe de liaison AngularJS, vous pouvez envisager d'utiliser Knockout.Punches, qui active la liaison de type guidon.

Examples

ngShow

Code AngularJS pour afficher / masquer dynamiquement un élément:

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

KnockoutJS équivalent:

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

ngBind (balisage bouclé)

Code AngularJS pour le rendu du texte brut:

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

KnockoutJS équivalent:

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

ou:

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

ngModel en entrée [type = text]

Code AngularJS pour une liaison bidirectionnelle sur une `input` texte:

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

KnockoutJS équivalent:

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

ngHide

Il n'y a pas de *liaison* équivalente *directe* dans KnockoutJS. Cependant, puisque cacher est le contraire de montrer, nous pouvons simplement inverser l'exemple pour l'équivalent `ngShow` de Knockout .

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

KnockoutJS équivalent:

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

L'exemple KnockoutJS ci-dessus suppose que `SomeScopeObservable` est une observable, et comme nous l'utilisons dans une expression (à cause de l'opérateur `!`), Nous ne pouvons pas omettre le `()` à la fin.

ngClass

Code AngularJS pour les classes dynamiques:

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

KnockoutJS équivalent:

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

Lire Equivalents des liaisons AngularJS en ligne: https://riptutorial.com/fr/knockout-js/topic/2408/equivalents-des-liaisons-angularjs

Chapitre 5: Fixations

Syntaxe

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

Remarques

Qu'est-ce qu'une liaison est

Essentiellement, une liaison ou une liaison de données permet de lier vos ViewModels à vos vues (modèles) et inversement. KnockoutJS utilise une liaison de données bidirectionnelle, ce qui signifie que les modifications apportées à votre ViewModel influencent la vue et les modifications apportées à votre vue peuvent influencer le ViewModel.

Sous le capot (bref aperçu)

Les liaisons ne sont que des plugins (scripts) qui vous permettent de résoudre une tâche particulière. Cette tâche modifie le plus souvent le balisage (html) en fonction de votre ViewModel.

Par exemple, une liaison de `text` vous permet d'afficher du texte et de le modifier dynamiquement chaque fois que votre ViewModel change.

KnockoutJS est livré avec de nombreuses liaisons puissantes et vous permet de l'étendre avec vos propres liaisons personnalisées.

Et surtout, les liaisons ne sont pas magiques, elles fonctionnent selon un ensemble de règles et chaque fois que vous n'êtes pas sûr de ce que fait une liaison, de ses paramètres ou de sa mise à jour, vous pouvez vous référer au code source de la liaison.

Prenons l'exemple suivant d'une liaison personnalisée:

```
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. Une liaison a un nom - `debug` pour que vous puissiez utiliser comme suit:

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

1. La méthode `init` est appelée une fois lorsque la liaison est lancée. Le reste des mises à jour est géré par un calculateur anonyme qui est disposé lorsque l' `element` est supprimé.
2. La liaison s'imprime pour consoler plusieurs choses: la valeur passée dans notre exemple cette valeur est `foo` (cette valeur peut également être observée depuis `ko.unwrap` méthode `ko.unwrap` est utilisée pour la lire), le viewModel actuel et bindingContext.
3. Chaque fois que la valeur transmise change, la liaison imprime les informations mises à jour sur la console.
4. Cette liaison ne peut pas être utilisée avec des éléments virtuels (dans les commentaires html), uniquement sur des éléments réels, car l'indicateur `ko.virtualElements.allowedBindings.debug` n'est pas défini sur true.

Quand utiliser les parenthèses

Sans aucun plugin supplémentaire, KnockoutJS n'aura que des mises à jour de View pour les propriétés du ViewModel qui sont *observables* (`observable` régulière, mais aussi `computed` , `pureComputed` , `observableArray` , etc.). Une observable serait créée comme ceci:

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

Dans ce cas, `vm.name` est une *fonction* avec deux "modes" distincts:

1. Getter: `vm.name()` , sans arguments, obtiendra la valeur actuelle;
2. Setter: `vm.name("Johnnyboy")` , avec un argument, définira une nouvelle valeur.

Dans les liaisons de données intégrées, vous pouvez *toujours* utiliser la forme de lecture, et vous pouvez *parfois omettre* les parenthèses, et la liaison les "ajoutera" effectivement pour vous. Donc, ceux-ci sont équivalents:

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

Mais cela échouera:

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

Parce que dès que vous voulez "faire" quelque chose avant de passer une valeur à une liaison de données, y compris des comparaisons de valeurs, vous devez "obtenir" correctement les valeurs pour toutes les observables, par exemple:

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

Voir aussi cette Q & R pour plus de détails.

Examples

Si / si

Vous pouvez utiliser la liaison `if` pour déterminer si les éléments enfants du nœud doivent être créés ou non.

```
<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'inverse de la liaison `if` est `ifnot`

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

est équivalent à

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

Parfois, vous ne voudrez pas contrôler la présence d'éléments sans avoir à créer un conteneur (généralement pour les éléments `<li>` dans les éléments `<ul>` ou `<option>` intérieur d'un élément `<select>`).

Knockout permet cela avec une syntaxe de flux de contrôle sans conteneur basée sur des balises de commentaire comme ceci:

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

Pour chaque

Semblable aux répéteurs utilisés dans d'autres langues. Cette liaison vous permettra de répliquer un bloc de HTML pour chaque élément d'un tableau.

```
<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>
```

Notez que lorsque nous parcourons notre contexte, il devient l'élément dans le tableau, en l'occurrence une instance de `contactViewModel` . Dans un `foreach` nous avons également accès à

- `$parent` - le modèle de vue qui a créé cette liaison
- `$root` - le modèle de vue racine (peut également être parent)
- `$data` - les données à cet index du tableau
- `$index` - l' `$index` (observable) de base de l'élément rendu

Avec

La liaison `with` lie le code HTML dans le noeud lié à un contexte distinct:

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

La liaison `with` peut également être utilisée sans élément de conteneur où un élément de conteneur peut ne pas être approprié.

```
<!-- 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" });
```

La liaison `with` a de nombreuses similitudes avec le `template` ou les liaisons `foreach` .

Visible

La liaison `visible` masquera un élément en appliquant `style="display: none;"` à elle lorsque la liaison est évaluée comme falsey.

```
<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

Lire Fixations en ligne: https://riptutorial.com/fr/knockout-js/topic/2249/fixations

Chapitre 6: Fixations personnalisées

Examples

Enregistrement obligatoire

Les liaisons Cusom doivent être enregistrées en étendant l'objet knockout bindingHandlers actuel.

Cela se fait en ajoutant une nouvelle propriété à l'objet.

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

Fondu de visibilité personnalisé

Cet exemple implémente une liaison personnalisée qui bascule la visibilité (similaire à la liaison visible existante), mais utilisera l'API d' évanouissement de jQuery pour animer la transition de visible à invisible.

Définition de liaison personnalisée:

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

Exemple de balisage avec la liaison 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
```

Modèle de vue échantillon:

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


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

Texte personnalisé remplace la liaison

Cet exemple est une liaison personnalisée qui remplace le texte chaque fois qu'une valeur d'entrée est mise à jour. Dans ce cas, les espaces seront remplacés par "+". Il est destiné à être utilisé avec la liaison de valeur existante et affiche une liaison avec un littéral d'objet.

Exemple de balisage avec la liaison replaceText:

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

Définition de liaison personnalisée:

```
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);
  }
}
```

Modèle de vue échantillon:

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


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

Remplacez par une liaison personnalisée par expression régulière

Définition de liaison personnalisée

```
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
}
```

Usage

ViewModel

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

Vue

```
<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>
```

Lire Fixations personnalisées en ligne: https://riptutorial.com/fr/knockout-js/topic/6332/fixations-personnalisees

Chapitre 7: Introduction des composants

Remarques

Les composants permettent des contrôles / widgets réutilisables représentés par leur propre vue (modèle) et leur propre modèle de vue. Ils ont été ajoutés dans Knockout 3.2. Inspiré des composants WebComposants, Knockout permet aux composants d'être définis en tant qu'éléments personnalisés, ce qui permet l'utilisation d'un balisage plus explicite.

Examples

Barre de progression (Boostrap)

Définition du composant

```
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>'
});
```

Utilisation HTML

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

Lire Introduction des composants en ligne: https://riptutorial.com/fr/knockout-js/topic/6207/introduction-des-composants

Chapitre 8: Les observables

Examples

Créer une observable

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>
```

Abonnement explicite aux observables

```
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');
```

Lire Les observables en ligne: https://riptutorial.com/fr/knockout-js/topic/6363/les-observables

Chapitre 9: Liaison Href

Remarques

Il n'y a pas de liaison `href` dans la bibliothèque principale de KnockoutJS, raison pour laquelle tous les exemples présentent d' *autres* fonctionnalités de la bibliothèque pour obtenir le même effet.

Voir aussi cette question Stack Overflow sur le même sujet .

Examples

Utilisation de la liaison attr

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

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

Comme il n'y a pas de liaison `href` native dans KnockoutJS, vous devez utiliser une fonctionnalité différente pour obtenir des liens dynamiques. L'exemple ci-dessus présente la liaison `attr` intégrée pour obtenir un lien dynamique.

Gestionnaire de liaison personnalisé

`href` liaison `href` n'est pas native à KnockoutJS, donc pour obtenir des liens dynamiques, utilisez un gestionnaire de liaison personnalisé:

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

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

Lire Liaison Href en ligne: https://riptutorial.com/fr/knockout-js/topic/6582/liaison-href

Chapitre 10: Liaisons - Champs de formulaire

Examples

Cliquez sur

La liaison par `click` peut être utilisée avec n'importe quel élément DOM visible pour ajouter un gestionnaire d'événement, qui invoquera une fonction JavaScript lorsque l'utilisateur clique sur un élément.

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

Les options

Utilisez cette liaison pour créer des options pour un élément de sélection

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

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

Vous pouvez également utiliser les propriétés à l' intérieur du tableau pour afficher dans la liste et pour enregistrer dans le viewmodel: `optionsText` permet un texte d'affichage personnalisé

`optionsValue` définit la propriété value de l' `<option>` correspondante `<option>`

`value` stocke la valeur de l'option sélectionnée dans une observable de la vueModel

```
<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>
```

Pour stocker les résultats d'une liste à sélection multiple, les options de liaison peuvent être combinées avec la liaison `selectedOptions` .

```
<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>
```

désactivé activé

La liaison désactivée ajoute un attribut `disabled` à un élément HTML, ce qui l'empêche d'être modifiable ou cliquable. Ceci est utile principalement pour les éléments `<input>` , `<select>` , `<textarea>` , `<a>` et `<button>`

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

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

L'inverse de la liaison `disabled` est `enabled`

La visibilité peut également être calculée à l'aide des fonctions JavaScript. Toutes les observables utilisées dans ces fonctions doivent être appelées avec des parenthèses

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

ou

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

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

soumettre

Gestionnaire d'événements à appeler lorsqu'un élément DOM est soumis.

```
<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 empêchera l'action de soumission par défaut du navigateur pour ce formulaire. Si vous souhaitez que votre formulaire soit soumis comme un formulaire HTML normal, vous retournez simplement `true` dans le gestionnaire de soumission.

Valeur

Utilisez la liaison de valeur pour obtenir la valeur d'un élément. La liaison de valeur peut être appliquée à n'importe quel contrôle de formulaire. Cependant, il existe d'autres liaisons mieux adaptées aux cases à cocher, aux boutons radio et aux entrées de texte.

L'exemple suivant illustre comment appliquer l'élément de liaison à plusieurs champs d'entrée de formulaire et comment renseigner les valeurs par défaut:

Définition de 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);
  });
}
```

Balisage associé:

```
<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>
```

Dans l'exemple ci-dessus, lorsqu'une valeur change, la nouvelle valeur sera consignée dans la console. Les valeurs initiales ne déclencheront pas d'événement de modification.

Par défaut, la liaison de valeur définit une modification en tant que modification de la valeur des éléments et le focus est transféré vers un autre élément. Cela peut être modifié à l'aide de l'option valueUpdate:

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

L'exemple ci-dessus changera la mise à jour de la valeur pour qu'elle se déclenche à la clé. Les options disponibles sont la saisie, la saisie au clavier, la pression sur le clavier et la réutilisation ultérieure.

Lire Liaisons - Champs de formulaire en ligne: https://riptutorial.com/fr/knockout-js/topic/7101/liaisons---champs-de-formulaire

Chapitre 11: Liaisons - Texte et apparence

Examples

Texte

La liaison de `text` peut être utilisée avec n'importe quel élément pour mettre à jour son 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")
});
```

La liaison de `text` peut également être utilisée avec des éléments virtuels.

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

CSS

Cette liaison appliquera la classe CSS fournie à l'élément. Les classes statiques sont appliquées lorsque les conditions données sont mal évaluées. Les classes dynamiques utilisent la valeur d'une observable ou calculée.

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; }
```

Voir aussi: documentation officielle .

Visible

Peut être utilisé pour afficher / masquer des éléments DOM. Semblable à l'utilisation de `if` , sauf que `visible` rendra toujours l'élément et définira l' `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>
```

Attirer

Utilisez la liaison `attr` pour appliquer des attributs supplémentaires à votre élément. Le plus couramment utilisé pour définir un href, src ou des attributs de données.

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

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

HTML

Cette liaison met à jour le innerHTML de l'élément en utilisant `jQuery.html()` , si jQuery a été référencé, sinon la propre logique d'analyse de KO. Cela peut être utile si vous récupérez du code HTML à partir d'une API, d'un flux RSS, etc. Veillez à utiliser cette balise avec l'entrée HTML

utilisateur.

page.html

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

page.js

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

Lire Liaisons - Texte et apparence en ligne: https://riptutorial.com/fr/knockout-js/topic/7103/liaisons---texte-et-apparence

Chapitre 12: Travailler avec knockout foreach liaison avec JSON

Examples

Travailler avec des boucles imbriquées

Voici la structure JSON que nous allons utiliser.

```
{
  "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
        }
      ]
    }
  ]
};
```

Cette structure json peut être assignée à une variable ou peut être une réponse de n'importe quelle api.

Comme nous pouvons le voir dans ce JSON, il existe un nœud externe qui contient des informations à leur sujet, et il existe un nœud interne qui raconte les compétences de chaque employé.

Nous allons donc créer un imbriqué pour chaque utilisation à l'aide de KO. Voici le 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>
```

Ici, dans le HTML ci-dessus, il y a deux listes qui contiennent la boucle foreach. la boucle externe contiendra le nœud externe de la structure json qui est des employés. Boucle intérieure détient les compétences de chaque employé. Dans chaque boucle, nous pouvons accéder aux propriétés du noeud correspondant. Par exemple, nous pouvons accéder au nom et à la notation à l'intérieur de la boucle de compétences, et non de l'extérieur.

Ci-dessous le code javascript.

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

Formez le point de vue javascript, il n'y a pas beaucoup de code. nous pouvons directement attribuer notre json à un tableau d'observation qui sera utilisé par HTML.

Lire Travailler avec knockout foreach liaison avec JSON en ligne:
https://riptutorial.com/fr/knockout-js/topic/6961/travailler-avec-knockout-foreach-liaison-avec-json

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec knockout.js	ASindleMouat, aswallows, Community, Jeroen, Michael Best, Ray, Ross Vernal, Tyrsius, user3297291
2	Déboguer une application knockout.js	Adam Wolski, AldoRomo88, natus, user3297291
3	Demandes AJAX et liaison	Kritner
4	Equivalents des liaisons AngularJS	Jeroen, Quango
5	Fixations	CreationEdge, dotnetom, Homer, Jeroen, Knekki, Kritner, Matthias Lloyd, natus, Nick DeFazio, Olga, stackoverfloweth, Steffen Nieuwenhoven, tmg, user3297291
6	Fixations personnalisées	AldoRomo88, natus, Nick DeFazio
7	Introduction des composants	4444, AldoRomo88, ASindleMouat
8	Les observables	Arun S, Norbert
9	Liaison Href	4444, Jeroen
10	Liaisons - Champs de formulaire	Knekki, Matthias Lloyd, Nick DeFazio, stackoverfloweth, Steffen Nieuwenhoven, tmg, user3297291
11	Liaisons - Texte et apparence	CreationEdge, Jeroen, Kritner, Nick DeFazio, stackoverfloweth, tmg
12	Travailler avec knockout foreach liaison avec JSON	suyesh