**EBook Gratis**

# APRENDIZAJE knockout.js

Free unaffiliated eBook created from
**Stack Overflow contributors.**

#knockout.j

S

# Tabla de contenido

# Acerca de

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

# Capítulo 1: Empezando con knockout.js

## Observaciones

Esta sección proporciona una descripción general de qué es knockout.js y por qué un desarrollador puede querer usarlo.

También debe mencionar cualquier tema grande en knockout.js y vincular a los temas relacionados. Dado que la Documentación para knockout.js es nueva, es posible que deba crear versiones iniciales de esos temas relacionados.

## Versiones

| Versión | Notas | Fecha de lanzamiento |
|---|---|---|
| 3.4.2 | Corrección de errores | 2017-03-06 |
| 3.4.1 | Corrección de errores | 2016-11-08 |
| 3.4.0 | | 2015-11-17 |
| 3.3.0 | | 2015-02-18 |
| 3.2.0 | Enlace de `component` introducidos | 2014-08-12 |
| 3.1.0 | | 2014-05-14 |
| 3.0.0 | Ver también: actualizar (desde 2.x) notas | 2013-10-25 |
| 2.3.0 | Último lanzamiento 2.x | 2013-07-08 |
| 2.0.0 | | 2011-12-21 |
| 1.2.1 | Último lanzamiento 1.x | 2011-05-22 |
| 1.0.0 | | 2010-07-05 |

## Examples

### Instalación o configuración

Knockout está disponible en la mayoría de las plataformas de JavaScript, o como un script independiente.

# Incluir como guion

Puede descargar el script desde su página de descarga , luego incluirlo en su página con una `script tag` estándar

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

## Usando un CDN

También puede incluir knockout desde 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>
```

# Instalar desde npm

```
npm install knockout
```

opcionalmente, puede agregar el parámetro `--save` para mantenerlo en su archivo `package.json`

# Instalar desde la glorieta

```
bower install knockout
```

opcionalmente, puede agregar el parámetro `--save` para mantenerlo en su archivo `bower.json`

# Instalar desde NuGet

```
Install-Package knockoutjs
```

### Comenzando: ¡Hola mundo!

## Creando un documento HTML y habilitando knockout.js

Cree un archivo HTML e incluya `knockout.js` través de una etiqueta `<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>
```

Agregue una segunda etiqueta `<script>` *debajo* del script knockout. En esta etiqueta de script, inicializaremos un modelo de vista y aplicaremos *enlaces de datos* a nuestro 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>
```

Ahora, continúe creando una *vista* agregando algo de HTML al cuerpo:

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

Cuando se abra el documento HTML y se ejecuten los scripts, verá una página que dice **¡Hola, Mundo!** . Cuando cambia las palabras en los elementos `<input>` , el texto `<h1>` se actualiza automáticamente.

---

# Cómo funciona el archivo creado

1. Una versión de depuración de knockout se carga desde una fuente externa (cdnjs)

Código:

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

2. Se crea una instancia de viewmodel que tiene propiedades *observables* . Esto significa que el knockout puede detectar cambios y actualizar la interfaz de usuario en consecuencia.

Código:

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

3. Knockout verifica el DOM para `data-bind` atributos de `data-bind` de `data-bind` y actualiza la interfaz de usuario usando el modelo de vista proporcionado.

Código:

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

4. Cuando encuentra un enlace de `text` , el knockout usa el valor de la propiedad tal como está definido en el modelo de vista enlazado para inyectar un nodo de texto:

Código:

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

## Observables calculados

Los observables calculados son funciones que pueden "ver" o "reaccionar" a otros observables en el modelo de vista. El siguiente ejemplo muestra cómo mostraría el número total de usuarios y la edad promedio.

*Nota: El ejemplo a continuación también puede usar **pureComputed ()** (introducido en v3.2.0) ya que la función simplemente calcula algo en función de otras propiedades del modelo de vista y devuelve un valor.*

```
<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);
```

Lea Empezando con knockout.js en línea: https://riptutorial.com/es/knockout-js/topic/799/empezando-con-knockout-js

# Capítulo 2: Depurando una aplicación knockout.js

## Examples

### Comprobando el contexto de enlace de un elemento DOM

Muchos errores en los enlaces de datos knockout son causados por propiedades indefinidas en un modelo de vista. Knockout tiene dos métodos prácticos para recuperar el contexto de enlace de 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);
```

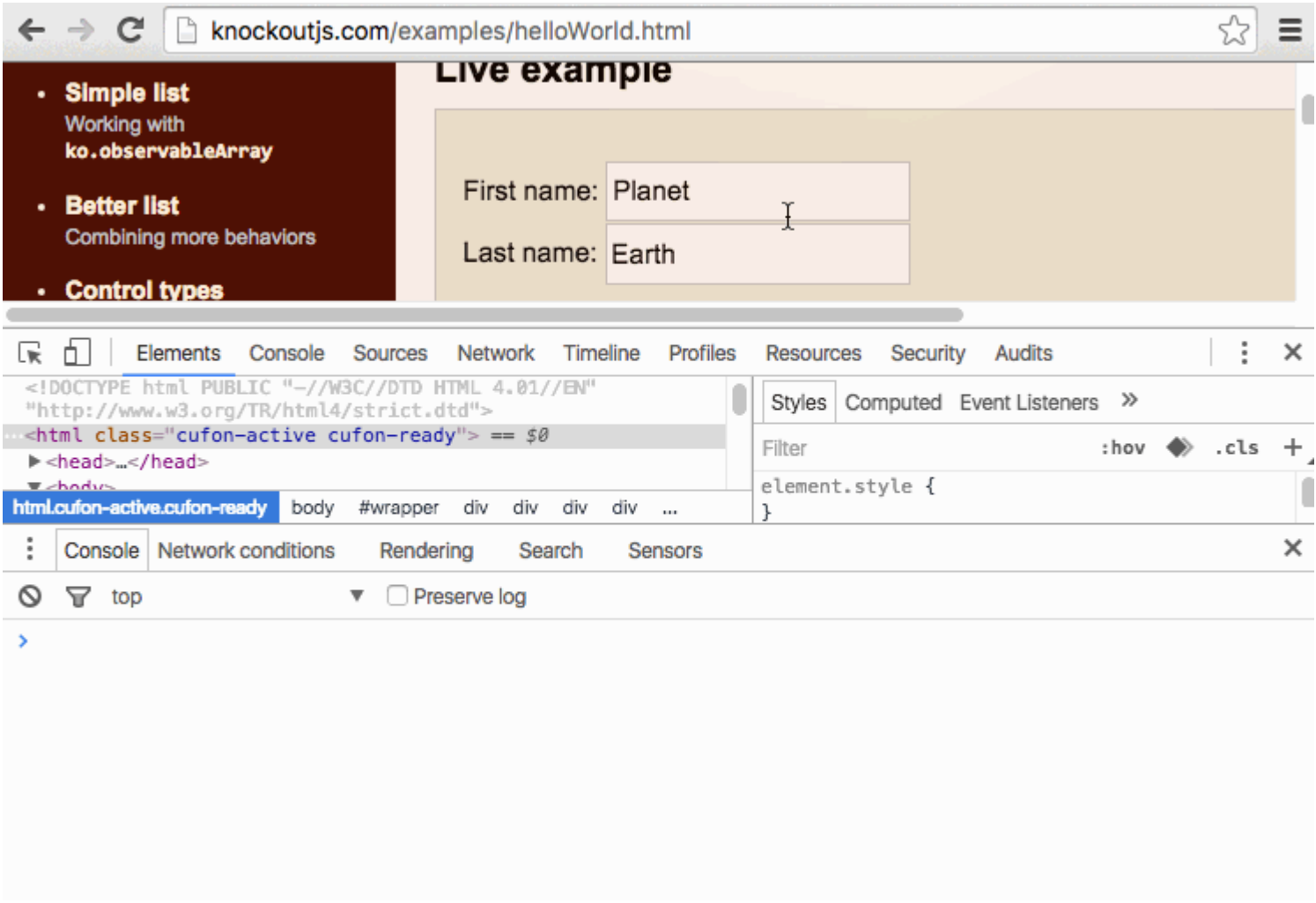
Para descubrir rápidamente el contexto de enlace de un elemento de UI, aquí hay un truco práctico:

La mayoría de los navegadores modernos almacenan el elemento DOM seleccionado actualmente en una variable global: `$0` ( más sobre este mecanismo )

- Haga clic derecho en un elemento de su interfaz de usuario y elija *"inspeccionar"* o *"inspeccionar elemento"* en el menú contextual.
- escriba `ko.dataFor($0)` en la consola del desarrollador y presione enter

También existen complementos de navegador que pueden ayudar a encontrar el contexto del objeto.

Un ejemplo (pruébalo en el ejemplo de Knockout hello world ):




## Imprimir un contexto de enlace desde el marcado

A veces es útil imprimir un enlace actual directamente desde el marcado. Un buen truco que permite usar un elemento DOM adicional con un enlace no existente (KO <3.0), un enlace personalizado o un enlace que no es relevante, como `uniqueName` .

Considera este ejemplo:

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

Si a uno le gustaría averiguar el contexto de enlace de cada elemento en la matriz de personas, puede escribir:

```
<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>
```

Lea Depurando una aplicación knockout.js en línea: https://riptutorial.com/es/knockout-js/topic/5066/depurando-una-aplicacion-knockout-js

# Capítulo 3: Encuadernaciones - campos de formulario

## Examples

### Hacer clic

El enlace de `click` se puede usar con cualquier elemento DOM visible para agregar un controlador de eventos, que invocará una función de JavaScript, cuando se `click` en el 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);
  }
});
```

### Opciones

Utilice este enlace para crear opciones para un elemento seleccionado

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

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

También puede usar propiedades dentro de la matriz para mostrar en la lista y para guardar en viewModel: `optionsText` permite un texto de visualización personalizado

`optionsValue` establece la propiedad de valor de la `<option>` correspondiente

`value` almacena el valor de la opción seleccionada en un observable del 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>
```

Para almacenar los resultados de una lista de selección múltiple, el enlace de opciones se puede combinar con el enlace de Opciones `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>
```

## deshabilitado / habilitado

El enlace deshabilitado agrega un atributo `disabled` a un elemento html, por lo que ya no se puede editar ni hacer clic. Esto es útil principalmente para los elementos `<input>` , `<select>` , `<textarea>` , `<a>` y `<button>`

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

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

El inverso del enlace `disabled` está `enabled`

La visibilidad también se puede calcular utilizando las funciones de JavaScript. Cualquier observable usado en estas funciones tiene que ser llamado entre paréntesis.

```
<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>
```

### enviar

El controlador de eventos se invoca cuando se envía 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 evitará la acción de envío predeterminada del navegador para ese formulario. Si desea que su formulario se envíe como un formulario HTML normal, simplemente devuelva `true` en el controlador de envío.

### Valor

Utilice el enlace de valores para obtener el valor de un elemento. El enlace de valores se puede aplicar a cualquier control de formulario, sin embargo, hay otros enlaces que pueden ser más adecuados para casillas de verificación, botones de radio y entradas de texto.

El siguiente ejemplo ilustra cómo aplicar el elemento de enlace a varios campos de entrada de formulario y cómo rellenar los valores predeterminados:

Definición 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);
  });
}
```

Marcado asociado:

```
<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>
```

En el ejemplo anterior, cuando un valor cambia, el nuevo valor se registrará en la consola. Los valores iniciales no activarán un evento de cambio.

De forma predeterminada, el enlace de valor define un cambio como un cambio en el valor de los elementos y el enfoque se transfiere a otro elemento. Esto se puede alterar usando la opción valueUpdate:

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

El ejemplo anterior cambiará la actualización de valor para activarse en la tecla arriba. Las opciones disponibles son entrada, keyup, keypress y afterkeydown.

Lea Encuadernaciones - campos de formulario en línea: https://riptutorial.com/es/knockout-js/topic/7101/encuadernaciones---campos-de-formulario

# Capítulo 4: Encuadernaciones - Texto y apariencia.

## Examples

### Texto

El enlace de `text` se puede utilizar con cualquier elemento para actualizar su texto interior.

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

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

El enlace de `text` también se puede utilizar con elementos virtuales.

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

### CSS

Este enlace aplicará la clase CSS suministrada al elemento. Las clases estáticas se aplican cuando las condiciones dadas se evalúan libremente como verdaderas. Las clases dinámicas utilizan el valor de un observable o computado.

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; }
```

Ver también: documentación oficial .

## Visible

Se puede utilizar para mostrar / ocultar elementos DOM. Similar a usar `if` , excepto que `visible` aún construirá el elemento y establecerá `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

Utilice el enlace `attr` para aplicar cualquier atributo adicional a su elemento. Más comúnmente utilizado para configurar un href, src o cualquier atributo de datos.

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

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

## HTML

Este enlace actualiza el innerHTML del elemento utilizando `jQuery.html()` , si jQuery ha sido referenciado, de lo contrario, la lógica de análisis de KO. Puede ser útil si recupera HTML de una API, fuente RSS, etc. Tenga en cuenta el uso de esta etiqueta con el HTML de entrada del usuario.

page.html

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

page.js

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

Lea Encuadernaciones - Texto y apariencia. en línea: https://riptutorial.com/es/knockout-js/topic/7103/encuadernaciones---texto-y-apariencia-

# Capítulo 5: Encuadernaciones personalizadas

## Examples

### Registro obligatorio

Los enlaces de Cusom deben registrarse extendiendo el objeto knockout bindingHandlers actual. Esto se hace agregando una nueva propiedad al objeto.

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

### Custom fade in / fade out enlace de visibilidad

Este ejemplo implementa un enlace personalizado que alterna la visibilidad (similar al enlace visible existente), pero utilizará la API de desvanecimiento de jQuery para animar la transición de visible a invisible.

Definición de encuadernación personalizada:

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

Marcado de muestra con el enlace 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
```

Modelo de vista de muestra:

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


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

## Texto personalizado reemplazar enlace

Este ejemplo es un enlace personalizado que reemplaza el texto cada vez que se actualiza un valor de entrada. En este caso, los espacios serán reemplazados con "+". Está destinado a ser usado junto con el enlace de valor existente, y muestra el enlace con un objeto literal.

Marcado de muestra con el enlace de reemplazo de texto:

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

Definición de encuadernación personalizada:

```
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);
  }
}
```

Modelo de vista de muestra:

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

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

## Reemplazar con el enlace personalizado expresión regular

### Definición de enlace personalizado

```
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()
});
```

*Ver*

```
<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>
```

Lea Encuadernaciones personalizadas en línea: https://riptutorial.com/es/knockout-js/topic/6332/encuadernaciones-personalizadas

# Capítulo 6: Equivalentes de los enlaces AngularJS

## Observaciones

No todo en AngularJS tiene un equivalente KnockoutJS (por ejemplo `ngCloack` o `ngSrc` ). Hay dos soluciones principales típicamente disponibles:

1. Utilice el `attr` genérico o el enlace de `event` lugar.
2. Al igual que las directivas personalizadas en AngularJS, puede escribir su propio controlador de enlace personalizado si necesita algo que no esté incluido en la biblioteca base.

Si prefiere la sintaxis de enlace de AngularJS puede considerar el uso de Knockout.Punches que habilita el enlace de estilo de manillar.

## Examples

### ngShow

Código AngularJS para mostrar / ocultar dinámicamente un elemento:

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

Equivalente a KnockoutJS:

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

### ngBind (marcado rizado)

Código AngularJS para renderizar texto plano:

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

Equivalente a 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 en la entrada [tipo = texto]

Código AngularJS para enlace bidireccional en una `input` texto:

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

Equivalente a KnockoutJS:

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

## ngOcultar

No hay *un* enlace equivalente *directo* en KnockoutJS. Sin embargo, dado que la ocultación es justo lo contrario de mostrar, podemos invertir el ejemplo para el equivalente `ngShow` de Knockout .

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

Equivalente a KnockoutJS:

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

El ejemplo anterior de KnockoutJS asume que `SomeScopeObservable` es un observable, y debido a que lo usamos en una expresión (debido al operador `!` Delante de él) no podemos omitir el `()` al final.

## clase ng

Código AngularJS para clases dinámicas:

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

Equivalente a KnockoutJS:

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

Lea Equivalentes de los enlaces AngularJS en línea: https://riptutorial.com/es/knockout-js/topic/2408/equivalentes-de-los-enlaces-angularjs

# Capítulo 7: Fijaciones

## Sintaxis

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

## Observaciones

# Lo que es un enlace

Esencialmente, un enlace o un enlace de datos es una forma de vincular sus ViewModels a sus Vistas (plantillas) y viceversa. KnockoutJS utiliza enlace de datos bidireccional, lo que significa que los cambios en su ViewModel influyen en la Vista y los cambios en su Vista pueden influir en el ViewModel.

## Bajo el capó (breve descripción)

Los enlaces son solo complementos (scripts) que le permiten resolver una tarea en particular. Esta tarea es más a menudo que no es cambiar el marcado (html) de acuerdo con su ViewModel.

Por ejemplo, un enlace de `text` permite mostrar texto y cambiarlo dinámicamente cada vez que cambie su ViewModel.

KnockoutJS viene con muchos enlaces poderosos y le permite extenderlo con sus propios enlaces personalizados.

Y lo más importante es que los enlaces no son mágicos, funcionan de acuerdo con un conjunto de reglas y, en cualquier momento que no esté seguro de qué hace el enlace, qué parámetros toma o cuándo actualizará la vista, puede consultar el código fuente del enlace.

Considere el siguiente ejemplo de un enlace personalizado:

```
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 enlace tiene un nombre: `debug` por lo que puede usar lo siguiente:

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

1. El método `init` se llama una vez cuando se inicia el enlace. El resto de las actualizaciones se manejan mediante un cálculo anónimo que se elimina cuando se elimina el `element` .
2. El enlace imprime en la consola varias cosas: el valor pasado en nuestro ejemplo, este valor es `foo` (este valor también puede observarse ya que el método `ko.unwrap` se usa para leerlo), el viewModel actual y bindingContext.
3. Siempre que el valor pasado cambie, el enlace imprimirá la información actualizada en la consola.
4. Este enlace no se puede usar con elementos virtuales (en comentarios html), solo en elementos reales, ya que el indicador `ko.virtualElements.allowedBindings.debug` no está establecido en verdadero.

# Cuándo usar paréntesis

Sin ningún complemento adicional, KnockoutJS solo tendrá actualizaciones de Vista en vivo para las propiedades en ViewModel que sean *observables* ( `observable` regulares, pero también `computed` , `pureComputed` , `observableArray` , etc.). Un observable se crearía así:

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

En este caso, `vm.name` es una *función* con dos "modos" separados:

1. Getter: `vm.name()` , sin argumentos, obtendrá el valor actual;
2. Setter: `vm.name("Johnnyboy")` , con un argumento, establecerá un nuevo valor.

En los enlaces de datos integrados, *siempre* puede utilizar el formulario de obtención, y en *ocasiones* puede *omitir* los paréntesis, y el enlace los "agregará" de manera efectiva. Así que estos son equivalentes:

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

Pero esto fallará:

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

Porque tan pronto como quiera "hacer" algo antes de pasar un valor a un enlace de datos, incluidas las comparaciones de valores, debe "obtener" correctamente los valores de todos los observables, por ejemplo:

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

Vea también este Q&A para más detalles.

# Examples

## Si / si no

Puede usar el enlace `if` para determinar si se deben crear o no los elementos secundarios del nodo.

```
<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>
```

El inverso de la unión `if` es `ifnot`

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

es equivalente a

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

A veces, no podrá controlar la presencia de elementos sin tener que crear un contenedor (normalmente para los elementos `<li>` en un elemento `<ul>` o `<option>` dentro de un `<select>` )

Knockout habilita esto con una sintaxis de flujo de control sin contenedor basada en etiquetas de comentarios como las siguientes:

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

## Para cada

Similar a los repetidores utilizados en otros idiomas. Este enlace le permitirá replicar un bloque de

html para cada elemento en una matriz.

```
<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>
```

Tenga en cuenta que cuando hacemos un bucle a través de nuestro contexto se convierte en el elemento dentro de la matriz, en este caso una instancia de `contactViewModel` . Dentro de un `foreach` también tenemos acceso a

- `$parent` : el modelo de vista que creó este enlace
- `$root` : el modelo de vista raíz (también podría ser padre)
- `$data` - los datos en este índice de la matriz
- `$index` - el índice basado en cero (observable) del elemento renderizado

## Con

El enlace `with` vincula el HTML dentro del nodo enlazado a un contexto separado:

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

El enlace `with` también se puede usar sin un elemento contenedor, donde un elemento contenedor puede no ser apropiado.

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

El enlace `with` tiene muchas similitudes con la `template` o `foreach` enlaces.

## Visible

El enlace `visible` ocultará un elemento aplicando `style="display: none;"` A ella cuando el enlace se evalúe como 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

Lea Fijaciones en línea: https://riptutorial.com/es/knockout-js/topic/2249/fijaciones

# Capítulo 8: Href vinculante

## Observaciones

No hay un enlace `href` en la biblioteca KnockoutJS central, por lo que todos los ejemplos muestran *otras* características de la biblioteca para obtener el mismo efecto.

Consulte también esta pregunta sobre el desbordamiento de pila en el mismo tema .

## Examples

### Usando attr binding

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

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

Como no hay un enlace `href` nativo en KnockoutJS, necesita usar una función diferente para obtener enlaces dinámicos. El ejemplo anterior muestra el enlace `attr` incorporado para obtener un enlace dinámico.

### Controlador de encuadernación personalizado

`href` enlace `href` no es nativo de KnockoutJS, así que para obtener enlaces dinámicos use un controlador de enlace personalizado:

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

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

Lea Href vinculante en línea: https://riptutorial.com/es/knockout-js/topic/6582/href-vinculante

# Capítulo 9: Introducción de componentes

## Observaciones

Los componentes permiten controles / widgets reutilizables representados por su propia vista (plantilla) y viewmodel. Fueron agregados en Knockout 3.2. Inspirado por WebComponents, Knockout permite que los componentes se definan como elementos personalizados, lo que permite el uso de un marcado más autoexplicativo.

## Examples

### Barra de progreso (Boostrap)

Definición 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>'
});
```

Uso de html

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

Lea Introducción de componentes en línea: https://riptutorial.com/es/knockout-js/topic/6207/introduccion-de-componentes

# Capítulo 10: Observables

## Examples

### Creando un 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>
```

### Suscripción explícita a 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');
```

Lea Observables en línea: https://riptutorial.com/es/knockout-js/topic/6363/observables

# Capítulo 11: Solicitudes y vinculación de AJAX.

## Examples

### Muestra de solicitud AJAX con enlace

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());
```

Tenga en cuenta que se usó JQuery ( `$.getJSON(...)` ) para realizar la solicitud. vainilla JavaScript puede realizar lo mismo, aunque con más código.

### "Cargando sección / notificación" durante la solicitud 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());
```

Tenga en cuenta que se usó JQuery ($ .getJSON (...)) para realizar la solicitud. vainilla JavaScript puede realizar lo mismo, aunque con más código.

Lea Solicitudes y vinculación de AJAX. en línea: https://riptutorial.com/es/knockout-js/topic/7538/solicitudes-y-vinculacion-de-ajax-

# Capítulo 12: Trabajando con knockout foreach vinculante con JSON

## Examples

### Trabajando con bucles anidados

Aquí está la estructura JSON que vamos a utilizar.

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

Esta estructura json puede asignarse a una variable o puede ser una respuesta de cualquier api.

Como podemos ver en este JSON, hay un nodo externo de empleados que contiene información sobre ellos, y hay un nodo interno que informa sobre las habilidades de cada empleado.

así que aquí vamos a crear un anidado para cada uno utilizando nocaut para cada uno. Aquí está el 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>
```

Aquí en el html anterior hay dos listas que contienen bucle foreach. El bucle externo mantendrá el nodo externo de la estructura json que es empleados. Bucle interior mantiene las habilidades de cada empleado. Dentro de cada bucle podemos acceder a las propiedades del nodo correspondiente. Para un ejemplo, podemos acceder al nombre y la calificación dentro del bucle de habilidades no desde afuera.

A continuación se muestra el código javascript.

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

Desde el punto de vista de javascript no hay mucho código. podemos asignar directamente nuestro json a un observblearray que será utilizado por Html.

Lea Trabajando con knockout foreach vinculante con JSON en línea: https://riptutorial.com/es/knockout-js/topic/6961/trabajando-con-knockout-foreach-vinculante-con-json

# Creditos

| S. No | Capítulos | Contributors |
|---|---|---|
| 1 | Empezando con knockout.js | ASindleMouat, aswallows, Community, Jeroen, Michael Best, Ray, Ross Vernal, Tyrsius, user3297291 |
| 2 | Depurando una aplicación knockout.js | Adam Wolski, AldoRomo88, natus, user3297291 |
| 3 | Encuadernaciones - campos de formulario | Knekki, Matthias Lloyd, Nick DeFazio, stackoverfloweth, Steffen Nieuwenhoven, tmg, user3297291 |
| 4 | Encuadernaciones - Texto y apariencia. | CreationEdge, Jeroen, Kritner, Nick DeFazio, stackoverfloweth, tmg |
| 5 | Encuadernaciones personalizadas | AldoRomo88, natus, Nick DeFazio |
| 6 | Equivalentes de los enlaces AngularJS | Jeroen, Quango |
| 7 | Fijaciones | CreationEdge, dotnetom, Homer, Jeroen, Knekki, Kritner, Matthias Lloyd, natus, Nick DeFazio, Olga, stackoverfloweth, Steffen Nieuwenhoven, tmg, user3297291 |
| 8 | Href vinculante | 4444, Jeroen |
| 9 | Introducción de componentes | 4444, AldoRomo88, ASindleMouat |
| 10 | Observables | Arun S, Norbert |
| 11 | Solicitudes y vinculación de AJAX. | Kritner |
| 12 | Trabajando con knockout foreach vinculante con JSON | suyesh |