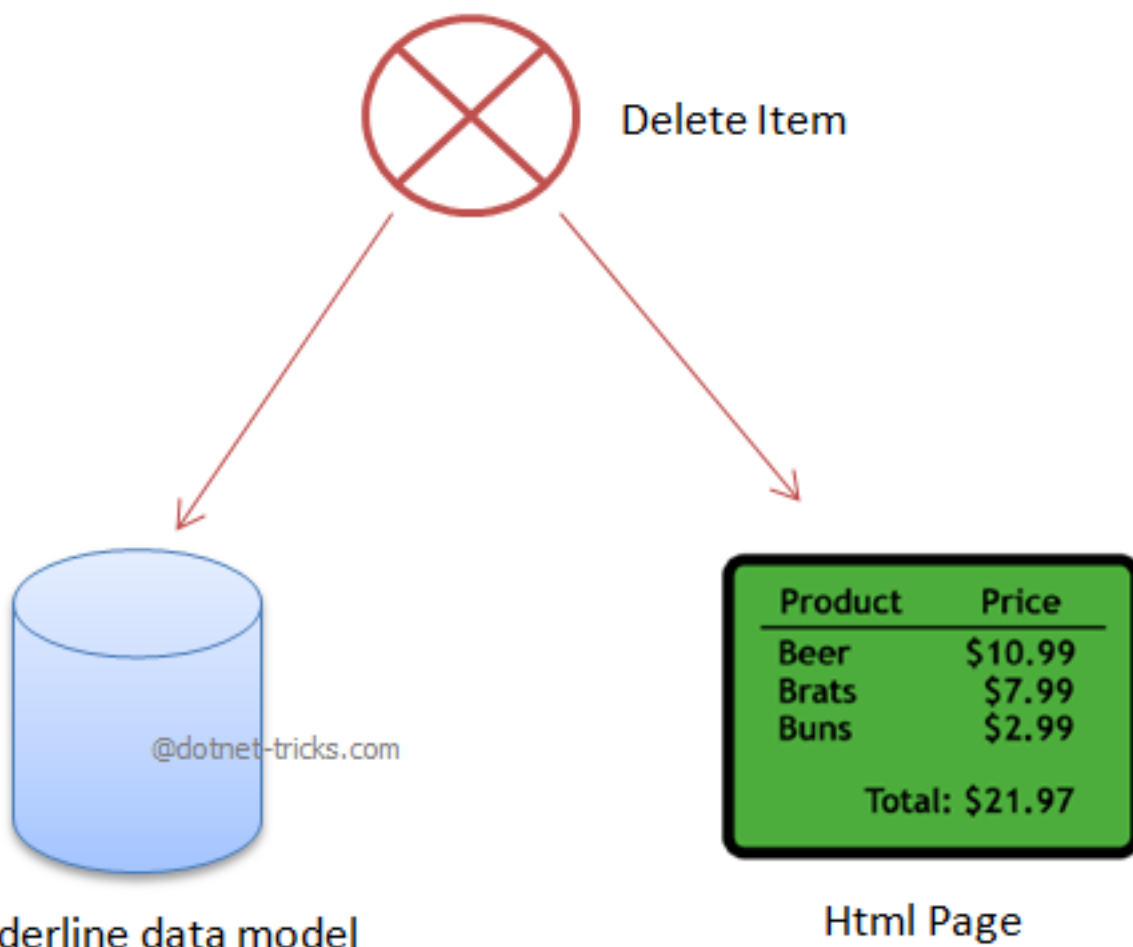


اگر از برنامه نویسی‌های پروژه‌های WPF درباره ویژگی‌های مهم الگوی MVVM بپرسید به احتمال زیاد اولین مطلبی که عنوان می‌شود این است که هنگام کار با الگوی MVVM در WPF باید از مباحث [data-binding](#) استفاده شود. به صورت خلاصه، [data-binding](#) مکانیزمی است که عناصر موجود در Xaml را به آبجکت‌های موجود در ViewModel یا سایر عناصر Xaml مقید می‌کند به طوری که با تغییر مقدار در آبجکت‌های ViewModel، عناصر View نیز خود را به روز می‌کنند یا با تغییر در مقادیر عناصر Xaml، آبجکت‌های متناظر در ViewModel نیز تغییر خواهند کرد (در صورت تنظیم Mode = TwoWay).

### Knockout.js چیست؟

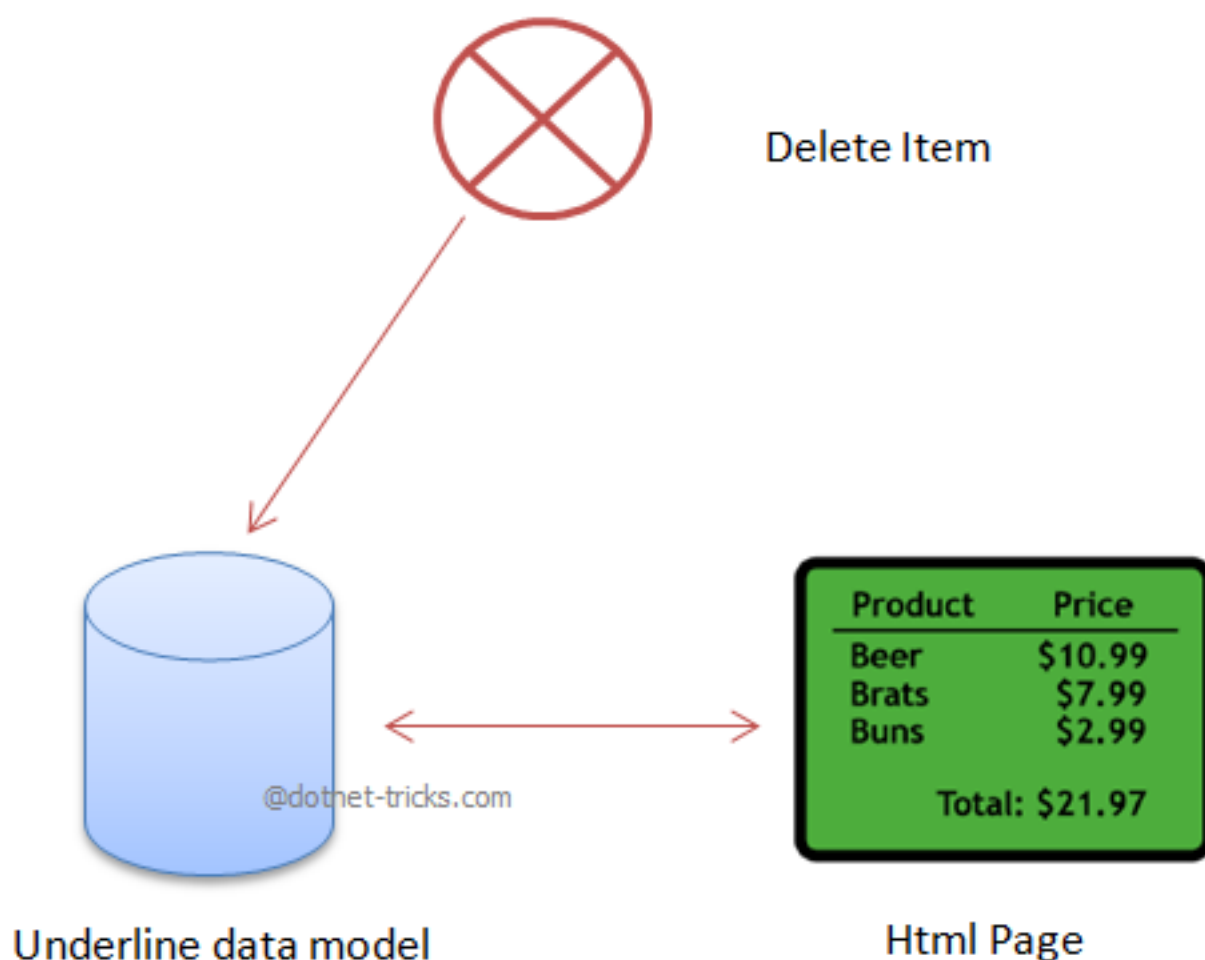
در یک جمله Knockout.js یک فریم ورک جاوا اسکریپت است که امکان پیاده سازی الگوی MVVM و مکانیزم [data-binding](#) را در پروژه‌های تحت وب به راحتی میسر می‌کند. به عبارت دیگر عناصر DOM را به [data-model](#) و آبجکت‌های [data-model](#) را به عناصر DOM مقید می‌کند، به طوری که با هر تغییر در مقدار یا وضعیت این عناصر یا آبجکت‌ها، تغییرات به موارد مقید شده نیز اعمال می‌گردد. به تصاویر زیر دقت کنید!

به روز رسانی [data-model](#) بدون استفاده از KO



**Before Knockout:** Manually tracking dependencies between HTML elements and their underlying data

به روز رسانی data-model با استفاده از KO



**After Knockout:** Automatically tracking dependencies between HTML elements and their underlying data

ویژگی‌های مهم KO

«ارائه یک راه حل بسیار ساده و واضح برای اتصال بخش‌های مختلف UI به data-model به روز رسانی خودکار عناصر و بخش‌های مختلف UI بر اساس تغییرات صورت گرفته در data-model به صورت کامل با کتابخانه و توابع javascript پیاده سازی شده است.

«حجم بسیار کم (سیزده کیلو بایت) بعد از فشرده سازی

«سازگار با تمام مرورگرهای جدید (IE 6+, Firefox 2+, Chrome, Safari, ...)

«امکان استفاده راحت بدون اعمال تغییرات اساسی در معماری پروژه هایی که در فاز توسعه هستند و بخشی از مسیر توسعه را طی کرده اند

...و»

آیا KO برای تکمیل JQuery در نظر گرفته شده است یا جایگزین؟

در اینکه JQuery بسیار محبوب است و در اکثر پروژه‌های تحت وب مورد استفاده است شکی وجود ندارد ولی این بدان معنی

نیست که با توجه به وجود JQuery و محبوبیت آن دیگر نیازی به KO احساس نمی‌شود. به عنوان یک مثال ساده : فرض کنید در یک قسمت از پروژه قصد داریم یک لیست از داده‌ها را نمایش دهیم. در پایین لیست تعداد آیتم‌های موجود در لیست مورد نظر نمایش داده می‌شود. یک دکمه Add داریم که امکان اضافه شدن آیتم جدید را در اختیار ما قرار می‌دهد. بعد از اضافه شدن یک مقدار، باید عددی که تعداد آیتم‌های لیست را نمایش می‌دهد به روز کنیم. خب اگر قصد داشته باشیم این کار را با JQuery انجام دهیم راه حل‌های زیر پیش رو است :

« به دست آوردن تعداد trهای جدول موجود؛

« به دست آوردن تعداد divهای موجود با استفاده از یک کلاس مشخص :css;

« یا حتی به دست آوردن تعداد آیتم‌های نمایشی در span هایی مشخص.

و البته سایر راه حل‌ها...

حال فرض کنید دکمه‌های دیگر نظیر Delete نیز مد نظر باشد که مراحل بالا تکرار خواهند شد. اما با استفاده از KO به راحتی می‌توانیم تعداد آیتم‌های موجود در یک آرایه را به یک عنصر مشخص bind کنیم به طور با هر تغییر در این مقدار، عنصر مورد نظر نیز به روز می‌شود یا به بیانی دیگر همواره تغییرات observe خواهند شد. برای مثال:

```
Number of items :<span data-bind="text: myList().count"></span>
```

در نتیجه برای کار با KO وابستگی مستقیم به استفاده از JQuery وجود ندارد ولی این امکان هست که بتوانیم هم از JQuery و هم از KO در کنار هم به راحتی استفاده کنیم و از قدرت‌های هر دو فریم ورک بهره ببریم و البته KO جایگزینی برای JQuery نخواهد بود. در پست بعد، شروع به کار با KO آموزش داده خواهد شد. ادامه دارد...

## نظرات خوانندگان

نویسنده: ابوالفضل رجب پور

تاریخ: ۱۳:۱۴ ۱۳۹۲/۰۶/۰۷

سلام و تشکر از آموزش خوبتون  
 KNOCKOUT در مقایسه با angular ، کدام مناسب تر هستند؟  
 آیا مقایسه این دو درست است؟  
 شنیدم روی ویژوال استودیو 2013 مایکروسافت پیش فرض آنگولار رو استفاده کرده. این خودش خیلی نقطه قوت هست و حتما روش فکر کرده مایکروسافت!  
 نظر شما چیه؟  
 به طور کل برای spa چه مجموعه فریم ورکی رو پیشنهاد میدید؟  
 مثلا ترکیب jquery + angular + requirejs چگونه؟

نویسنده: محسن خان

تاریخ: ۱۳:۴۲ ۱۳۹۲/۰۶/۰۷

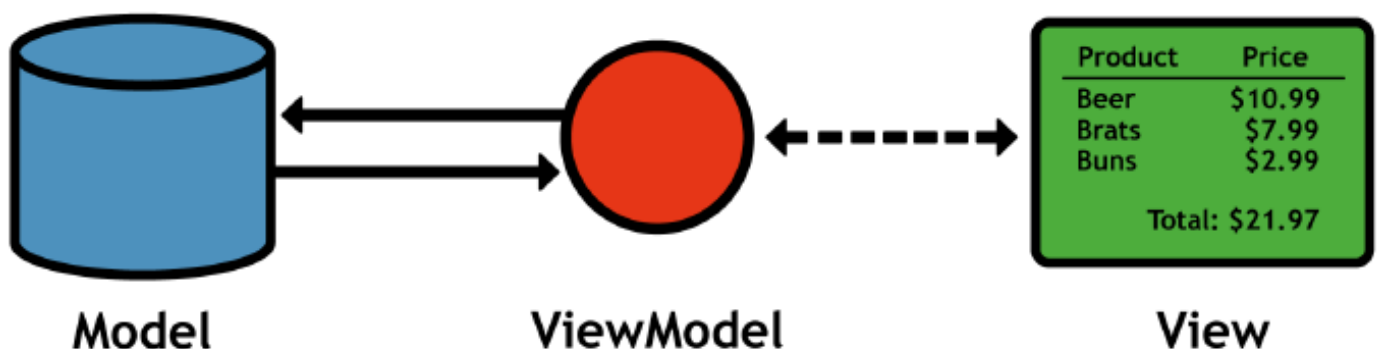
در این مطلب فقط بوت استرپ نگارش 2 در MVC 5 پیش فرض شده. قبلا ناک آوت در MVC4 بود جزو اسکریپت های پیش فرض.

نویسنده: آرمان فرقانی

تاریخ: ۱۴:۱۷ ۱۳۹۲/۰۷/۰۶

این سوال خوبی است. اما گمان نمی کنم بشود پاسخ دقیقی به بخش کدام مناسب تر است به طور کلی داد. شاید بتوانید بر اساس علاقه به MVC یا MVVM یکی را برگزینید. برای کسانی هم که می خواهند یکی را شروع کنند شاید Knockout برای شروع با توجه به داکيومنت و بخش آموزش جالب آن بهتر باشد. همچنین مقایسه هایی مانند این یا بحث هایی مانند این کمک کننده است برای انتخاب بین این فریم ورک ها. البته هر دو فریم ورک مدرن و مناسب برای بسیاری موارد هستند. نظر شخصی من این است اگر ASP.NET MVC کار می کنید Angular را به صورت راه حل کلی دنبال کنید چون کمی کسب مهارت و آشنایی با تمام مفاهیم آن نسبت به Knockout بیشتر طول می کشد. و زمانی که صرف یافتن گزینه بهتر بین این دو می کنید را برای مطالعه Knockout با استفاده از مقالات همین سایت یا بخش آموزش سایت رسمی آن اختصاص دهید. گمان نمی کنم از صرف وقت برای این دو پشیمان شوید. هر کدام شیرینی خاص خود را دارند.

در پست قبلی با مفاهیم و ویژگی‌های کلی KO آشنا شدید. KO از الگوی طراحی MVVM استفاده می‌کند. از آن جا که یکی از پیش نیازهای KO آشنایی اولیه با مفاهیم View و Model است نیاز به توضیح در این موارد نیست اما اگر به هر دلیلی با این مفاهیم آشنایی ندارید می‌توانید از اینجا شروع کنید. اما درباره ViewModel که کمی مفهوم متفاوتی دارد، این نکته قابل ذکر است که KO از ViewModel برای ارتباط مستقیم بین View و Model استفاده می‌کند، چیزی شبیه به منطق MVC با این تفاوت که ViewModel به جای Controller قرار خواهد گرفت.



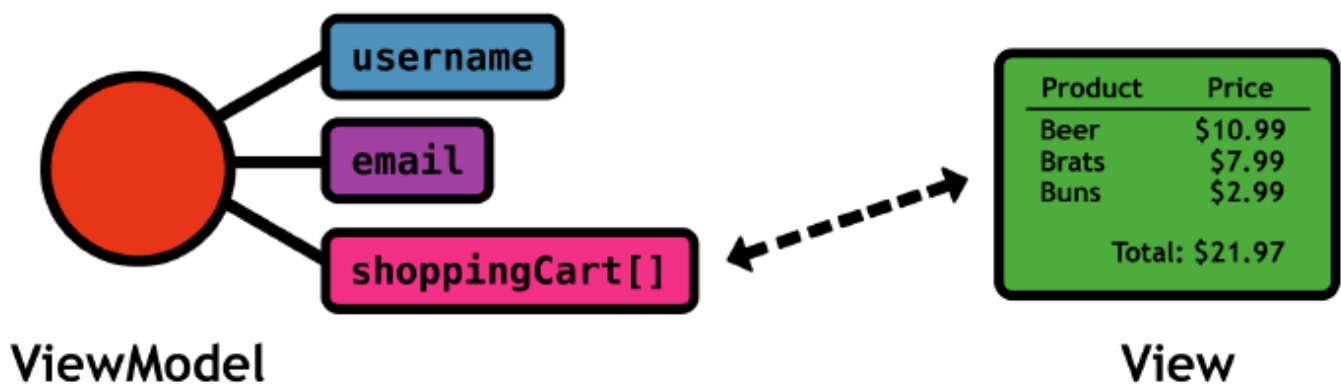
ابتدا باید به شرح برخی مفاهیم در KO بپردازم:

#### «Observable(قابل مشاهده کردن تغییرات)»

KO از Observable برای ردیابی و مشاهده تغییرات خواص ViewModel استفاده می‌کند. در واقع Observable دقیقاً شبیه به متغیرها در JavaScript عمل می‌کنند با این تفاوت که به KO اجازه می‌دهند که تغییرات این خواص را پیگیری کند و این تغییرات را به بخش‌های مرتبط View اعمال نماید. اما سوال این است که KO چگونه متوجه می‌شود که این تغییرات بر کدام قسمت در View تاثیر خواهند داشت؟ جواب این سوال در مفهوم Binding است.

#### «Binding»

برای اتصال بخش‌های مختلف View به Observableها باید از binding(مقید سازی) استفاده کنیم. بدون عملیات binding، امکان اعمال تغییرات Observableها بر روی عناصر HTML امکان پذیر نیست. برای مثال در شکل زیر یکی از خواص ViewModel را به View متناظر مقید شده است.



با کمی دقت در شکل بالا این نکته به دست می‌آید که می‌توان در یک ViewModel، فقط خواص مورد نظر را به عناصر HTML مقید کرد.

#### دانلود فایل‌های مورد نیاز

فایل‌های مورد نیاز برای KO رو می‌توانید از [اینجا](#) دانلود نمایید و به پروژه اضافه کنید. به صورت پیش فرض فایل‌های مورد نیاز KO، در پروژه‌های MVC 4 وجود دارد و نیاز به دانلود آن‌ها نیست و شما باید فقط مراحل BundleConfig را انجام دهید.

#### تعریف ViewModel

برای تعریف ViewModel و پیاده سازی مراحل Observable و binding باید به صورت زیر عمل نمایید:

```
<html lang='en'>
<head>
<title>Hello, Knockout.js</title>
<meta charset='utf-8' />
<link rel='stylesheet' href='style.css' />
</head>
<body>
<h1>Hello, Knockout.js</h1>
<script type='text/javascript' src='knockout-2.1.0.js'>
  <script type='text/javascript'>
    var personViewModel = {
      firstName: "Masoud",
      lastName: "Pakdel"
    };
    ko.applyBindings(personViewModel);
  </script>
</script>
</body>
</html>
```

مشاهده می‌کنید که ابتدا یک ViewModel به نام person ایجاد کردم همراه با دو خاصیت به نام‌های firstName و lastName. تابع applyBinding برای KO بدین معنی است که این آبجکت به عنوان یک ViewModel در این صفحه مورد استفاده قرار خواهد گرفت. اما برای مشاهده تغییرات باید یک عنصر HTML را به این ViewModel مقید (bind) کنیم.

#### مقید سازی عناصر HTML

برای مقید سازی عناصر HTML به ViewModel‌ها باید از data-bind attribute استفاده نماییم. برای مثال:

```
<p><span data-bind='text: firstName'></span>'s Shopping Cart</p>
```

اگر به `data-bind` در تگ `span` بالا توجه کنید خواهید دید که مقدار `text` در این تگ را به خاصیت `firstName` در `viewModel` این صفحه `bind` شده است. تا اینجا `KO` می‌داند که چه عنصر از `DOM` به کدام خاصیت از `ViewModel` مقید شده است اما هنوز دستور ردیابی تغییرات (`Observable`) را برای `KO` تعیین نکردیم.

### چگونه خواص را `Observable` کنیم

در پروژه‌های `WPF`، فقط در صورتی تغییرات خواص یک کلاس ردیابی می‌شوند که اولاً کلاس اینترفیس `INotifyPropertyChanged` را پیاده‌سازی کرده باشد ثانیاً، در متد `set` این خواص، متد `OnPropertyChanged` (البته این متد می‌تواند هر نام دیگری نیز داشته باشد) صدا زده شده باشد. نکته مهم و اساسی در `KO` نیز همین است که برای اینکه `KO` بتواند تغییرات هر خاصیت را مشاهده کند حتماً خواص مورد نظر باید `Observable` شوند. برای این کار کافیسیت به صورت عمل کنید:

```
var personViewModel = {
  firstName: ko.observable("Masoud"),
  lastName: ko.observable("Pakdel")
};
```

مزیت اصلی برای اینکه حتماً خواص مورد نظرتان `Observable` شوند این است که، در صورتی که مایل نباشید تغییرات یک خاصیت بر روی `View` اعمال شود کافیسیت از دستور بالا استفاده نکنید. درست مثل اینکه هرگز مقدار آن تغییر نکرده است.

### پیاده‌سازی متدهای `get` و `set`

همان‌طور که متوجه شدید، `Observable`ها متغیر نیستند بلکه تابع هستند در نتیجه برای دستیابی به مقدار یک `observable` کافیسیت آن را بدون پارامتر ورودی صدا بزنیم و برای تغییر در مقدار آن باید همان تابع را با مقدار جدید صدا بزنیم. برای مثال:

```
personViewModel.firstName() // Get
personViewModel.firstName("Masoud") // Set
```

البته این نکته را هم متذکر شوم که در `ViewModel`های خود می‌توانید توابع سفارشی مورد نیاز را بنویسید و از آن‌ها در جای مناسب استفاده نمایید (شبیه به مفاهیم `Command`ها در `WPF`)

### مقید سازی تعاملی

اگر با `WPF` آشنایی دارید می‌دانید که در این گونه پروژه‌ها می‌توان رویدادهای مورد نظر را به `Command`های خاص در `ViewModel` مقید کرد. در `KO` نیز این امر به آسانی امکان پذیر است که به آن `Interactive Bindings` می‌گویند. فقط کافیسیت در `data-bind` attribute از نام رویداد استفاده نماییم. مثال:

ابتدا یک `ViewModel` به صورت زیر خواهیم داشت:

```
function PersonViewModel() {
  this.firstName = ko.observable("Masoud");
  this.lastName = ko.observable("Pakdel");
  this.clickMe = function() {
    alert("this is test!");
  };
};
```

تنها نکته قابل ذکر تعریف تابع سفارشی به نام `clickMe` است که به نوعی معادل `Command` مورد نظر ما در `WPF` است. در عنصر `HTML` مورد نظر که در این جا `button` است باید `data-binding` به صورت زیر باشد:

```
<button data-bind='click: clickMe'>Click Me...</button>
```

در نتیجه بعد از کلیک بر روی `button` بالا تابع مورد نظر در `viewModel` اجرا خواهد شد. پس به صورت خلاصه:

ابتدا ViewModel مورد نظر را ایجاد نمایید؛  
سپس با استفاده از data-bind عملیات مقید سازی بین View و ViewModel را انجام دهید  
در نهایت با استفاده از Obsevable تغییرات خواص مورد نظر را ردیابی نمایید.

ادامه دارد...



## نظرات خوانندگان

نویسنده:

نوید

تاریخ:

۱۳:۲۳ ۱۳۹۲/۰۶/۰۶

اگر امکان داره کدهای مثالهای مربوطه رو هم بذارید . امیدوارم این سری آموزش رو ادامه بدین . با تشکر

نویسنده:

مسعود پاکدل

تاریخ:

۱۳:۴۲ ۱۳۹۲/۰۶/۰۶

در پست‌های بعدی که مفاهیم مهم و اصلی Knockout رو بررسی می‌کنیم حتما مثال‌های مربوطه قرار داده می‌شوند.

نویسنده:

دادخواه

تاریخ:

۱۶:۱۱ ۱۳۹۲/۰۶/۰۶

سلام

اگر از فریم ورک‌های KnockoutJS و یا AngularJS استفاده کنم. آیا نیاز هست که JQuery را نیز ضمیمه کنم و یا دیگر به JQuery نیازی نیست؟

آیا کارهایی که JQuery انجام می‌دهد را این دو فریم ورک و یا کلا فریم ورک‌های دیگر می‌توانند انجام دهند؟

تشکر

نویسنده:

محسن خان

تاریخ:

۱۶:۴۳ ۱۳۹۲/۰۶/۰۶

Knockout.js جایگزین JQuery یا MooTools نیست. در این کتابخانه animation یا مدیریت عمومی رخدادها، ساده سازی Ajax و مانند آن پیاده سازی نشده‌اند (هرچند Knockout.js امکان parse اطلاعات Ajax ایی دریافتی را دارد). هدف از Knockout.js ارائه مکملی برای سایر فناوری‌های وب جهت تولید برنامه‌های غنی و دسکتاپ مانند وب است. پشتیبانی خوبی از آن توسط میکروسافت صورت می‌گیرد چون [نویسنده‌اش](#) عضو تیم ASP.NET MVC است.

نویسنده:

سعید یزدانی

تاریخ:

۱۷:۵۲ ۱۳۹۲/۰۶/۱۸

سلام تشکر بابت مطالب ارزشمندی که گذاشتید

آیا میشه در view از چند view model استفاده کرد ؟

اگر میشه چطور باید در هنگام bind کردن به html صفحه از هم تفکیک کرد

نویسنده:

مسعود پاکدل

تاریخ:

۲۲:۱۵ ۱۳۹۲/۰۶/۱۸

ممنون دوست عزیز.

بله امکان پذیر است. باید از المان‌های تودرتو استفاده کنیم. به این صورت که المان ریشه با استفاده از with به model مربوطه مقید می‌شود و المان‌های داخلی به خواص مدل bind می‌شوند. برای مثال:

```
<div data-bind="text: teacher"> </div> //مقدد سازی به مدل اول
<p data-bind="with: student"> //مقدد سازی المان ریشه به مدل دوم
  Name: <span data-bind="text: name"> </span>, //مقدد سازی المان‌های داخلی به خواص
  Family: <span data-bind="text: family"> </span>
</p>

<script type="text/javascript">
  ko.applyBindings({
    teacher: "myTeacher",
    student: {
```

```
        name: "Masoud",  
        family: "Pakdel"  
    }  
});  
</script>
```

در ادامه مباحث قبلی، در این پست به بررسی سایر قابلیت‌های Observable ها در KO خواهیم پرداخت.

### Computed Observables

Computed Observables ها به واقع خواصی هستند که از ترکیب چند خاصیت دیگر به دست می‌آیند یا برای به دست آوردن مقادیر آن‌ها باید یک سری محاسبات را انجام داد. برای مثال به ViewModel زیر دقت کنید:

```
var personViewModel = {
  firstName: ko.observable("Masoud"),
  lastName: ko.observable("Pakdel"),
  this.fullName = ko.computed(function() {
    return this.firstName() + " " + this.lastName();
  }, this);
};
```

همان طور که مشخص است یک خاصیت به نام fullName ایجاد کردم که از ترکیب خواص firstName و lastName به دست آمده است. برای ایجاد این گونه خواص باید از دستور ko.compute استفاده شود که پارامتر ورودی آن یک تابع برای برگشت مقدار مورد نظر است. برای مقید کردن این خاصیت به کنترل مورد نظر نیز همانند قبل عمل خواهیم نمود:

```
<span data-bind='text: fullName'></span>
```

### آرایه ای از Observable

برای ردیابی و مشاهده تغییرات در یک آرایه باید از Observable array استفاده نماییم. برای درک بهتر موضوع یک مثال را پیاده سازی خواهیم کرد: در این مثال یک لیست از محصولات مورد نظر را داریم به همراه یک button برای اضافه کردن محصول جدید. بعد از کلیک بر روی دکمه مورد نظر، یک محصول جدید، به لیست اضافه خواهد شد و تغییرات لیست در لحظه مشاهده خواهد شد. ابتدا باید مدل مورد نظر را ایجاد کنیم.

```
function Product(name, price) {
  this.name = ko.observable(name);
  this.price = ko.observable(price);
}
```

برای ایجاد یک Observable Array باید از دستور ko.observableArray استفاده کنیم که ورودی آن نیز مجموعه ای از داده مورد نظر است:

```
this.shoppingCart = ko.observableArray([
  new Product("Beer", 10.99),
  new Product("Brats", 7.99),
  new Product("Buns", 1.49)
]);
```

در ابتدا یک لیست با سه مقدار خواهیم داشت. برای نمایش لیست، نیاز به یک جدول داریم که کد آن به صورت زیر خواهد بود:

```
<table>
<thead><tr>
<th>Product</th>
<th>Price</th>
</tr></thead>
<tbody data-bind='foreach: shoppingCart'>
<tr>
```

```
<td data-bind='text: name'></td>
<td data-bind='text: price'></td>
</tr>
</tbody>
</table>
```

یک توضیح: همانطور که می‌بینید در تگ <tbody> از دستور foreach برای پیمایش لیست مورد نظر (shoppingCart) استفاده شده است. برای مقید سازی تگ‌های <td> به مقادیر ViewModel از data-bind attribute استفاده شده است. حال نیاز به یک button داریم تا با کلیک با بر روی آن یک product جدید به لیست اضافه خواهد شد.

```
<button data-bind='click: addProduct'>Add Beer</button>
```

در ViewModel یک تابع جدید به نام addProduct ایجاد می‌کنیم:

```
this.addProduct = function() {
    this.shoppingCart.push(new Product("More Beer", 10.99));
};
```

از دستور push برای اضافه کردن یک آیتم به لیست اضافه می‌شود. تا اینجا کدهای ViewModel به صورت زیر خواهد بود:

```
function PersonViewModel()
{
    this.firstName = ko.observable("John");
    this.lastName = ko.observable("Smith");
    this.checkout = function () {
        alert("Trying to checkout");
    };
    this.fullName = ko.computed(function(){
        return this.firstName() + " " + this.lastName();
    }, this);

    this.shoppingCart = ko.observableArray([
        new Product("Beer", 10.99),
        new Product("Brats", 7.99),
        new Product("Buns", 1.49)
    ]);

    this.addProduct = function () {
        this.shoppingCart.push(new Product("More beer", 10.99));
    };
};
```

### [دریافت سورس مثال تا اینجا](#)

در این مرحله قصد داریم که یک button نیز برای حذف آیتم از لیست ایجاد کنیم. در ابتدا یک تابع جدید به نام removeProduct به صورت زیر ایجاد خواهیم کرد:

```
this.removeProduct = function(product) {
    self.shoppingCart.remove(product);
};
```

با کمی دقت متوجه خواهید شد که به جای this از self استفاده شده است. در واقع self چیزی نیست جز یک اشاره گر به viewModel جاری. اگر از this استفاده کنید با یک TypeError مواجه خواهید شد و برای جلوگیری از این خطا باید در ابتدای ViewModel دستور زیر را بنویسیم:

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

و در کدهای HTML جدول مورد نظر نیز باید تغییرات زیر را اعمال کنیم:

```
<tr>
  <td data-bind='text: name'></td>
  <td data-bind='text: price'></td>
  <td><button data-bind='click:
    $root.removeProduct'>Remove</button></td>
</tr>
```

به ازای هر محصول یک button داریم که البته رویداد کلیک آن به تابع removeProduct عنصر جاری مقید شده است (\$root به عنصر جاری در لیست اشاره می‌کند).

دستور remove در لیست باعث حذف کامل آیتم از لیست خواهد شد و در خیلی موارد این مورد برای ما خوشایند نیست زیرا حذف یک آیتم از لیست باید در سمت سرور نیز انجام شود نه صرفاً در سمت کلاینت، در نتیجه می‌توانیم از دستور destroy استفاده کنیم. استفاده از این دستور باعث خواهد شد که عنصر مورد نظر در لیست نمایش داده نشود ولی به صورت واقعی از لیست حذف نشده است (این کار را با تغییر در مقدار خاصیت \_destroy هر عنصر انجام می‌دهد) ادامه دارد...

[دریافت سورس مثال](#)

## نظرات خوانندگان

نویسنده:

ناصر

تاریخ:

۱۳۹۲/۰۶/۰۹ ۱۲:۲۳

تشکر. در قسمت آخر اشاره به حذف شدن آیتم در سمت سرور کردید.  
برای این کار، چطور باید درخواستی رو به سرور به صورت AJAX ارسال کرد؟

نویسنده:

محسن خان

تاریخ:

۱۳۹۲/۰۶/۰۹ ۲۲:۴

متد [ko.toJSON](#) می‌تونه ViewModel رو به JSON تبدیل کنه. بعد jQuery Ajax رو فراخوانی کنید تا به سرور ارسال بشه.

نویسنده:

رحیمی

تاریخ:

۱۳۹۲/۰۶/۱۳ ۱۱:۱۸

با سلام  
من مشکلی که با foreach پیدا کردم این بود که توی ie8 اجرا نمیشد با جستجو و پیدا کردن یه اسکریپت به نام modernizr توی ie8 هم کار کرد اما مشکل این هست که یک سطر اضافه انجام میشه که در واقع همون چیزی که درون تگ مربوط به foreach هست رو هم آخرین سطر میاره

نویسنده:

رحیمی

تاریخ:

۱۳۹۲/۰۶/۱۳ ۱۲:۲۳

سلام  
اشکال کار رو پیدا کردم همش از یک کامای آخر بود  
یعنی وقتی ایت‌ها رو معرفی می‌کردم آخرین ایت‌هم بعدش کاما بود برای همین یک ابجکت خالی هم تکرار میشد

### مقید سازی رویداد کلیک

Click Binding روشی است برای اضافه کردن یک گرداننده رویداد در زمانی که قصد داریم یک تابع جاوااسکریپتی را در هنگام کلیک بر روی المان مورد نظر فراخوانی کنیم. از این مقید سازی عموماً در عناصر button و input و تگ a استفاده می‌شود. اما در حقیقت در تمام عناصر غیر پنهان صفحه مورد استفاده قرار می‌گیرد.

```
<div>
  Number Of Clicks <span data-bind="text: numberOfClicks"></span> times
  <button data-bind="click: clickMe">Click me</button>
</div>

<script type="text/javascript">
  var viewModel = {
    numberOfClicks : ko.observable(0),
    clickMe: function() {
      var previousCount = this.numberOfClicks();
      this.numberOfClicks(previousCount + 1);
    }
  };
</script>
```

رویداد کلیک button در کد بالا به تابعی با نام clickMe مقید شده است. این تابع در viewModel جاری صفحه تعریف شده است و در بدنه آن تعداد کلیک‌های قبلی را به علاوه یک خواهد کرد. از آنجا که تگ span در بالای صفحه به تعداد کلیک‌ها مقید شده است در نتیجه همواره مقدار این تگ به روز خواهد بود.

**\*نکته اول:** اگر قصد داشته باشیم که عنصر جاری در viewModel را به گرداننده رویداد پاس دهیم چه باید کرد؟

هنگام فراخوانی رویدادها، KO به صورت پیش فرض مقدار جاری مدل را به عنوان اولین پارامتر به این گرداننده پاس می‌دهد. این روش مخصوصاً در هنگامی که قصد اجرای عملیاتی خاص بر روی تک تک عناصر یک مجموعه را داشته باشید (مثل حلقه foreach) بسیار مفید خواهد بود.

```
<ul data-bind="foreach: places">
  <li>
    <span data-bind="text: $data"></span>
    <button data-bind="click: $parent.removePlace">Remove</button>
  </li>
</ul>

<script type="text/javascript">
  function MyViewModel() {
    var self = this;
    self.places = ko.observableArray(['Tehran', 'Esfahan', 'Shiraz']);

    self.removePlace = function(place) {
      self.places.remove(place)
    }
  }
  ko.applyBindings(new MyViewModel());
</script>
```

در تابع removePlace می‌بینید که مقدار آیتم جاری در لیست به عنوان اولین آرگومان به این تابع پاس داده می‌شود، در نتیجه می‌دانیم که کدام عنصر را باید از لیست مورد نظر حذف کنیم. برای به دست آوردن آیتم جاری در لیست از \$parent یا \$root می‌توان استفاده کرد.

همان طور که پست قبل توضیح داده شد؛ برای اینکه بتوانیم از یک viewModel به مجموعه از عناصر در یک حلقه foreach مقید کنیم امکان استفاده از اشاره گر this میسر نیست. در نتیجه بهتر است در ابتدای viewModel مقدار این اشاره گر را در یک متغیر معمولی (در اینجا به نام self است) ذخیره کنیم و از این پس این متغیر را برای اشاره به عناصر viewModel به کار ببریم. در اینجا self به عنوان یک alias برای this خواهد بود.

**\*نکته دوم:** دسترسی به عنصر رویداد

در بعضی مواقع نیاز است در حین فراخوانی رویداد، عنصر رویداد DOM به عنوان فرستنده در اختیار تابع گرداننده قرار گیرد. خبر خوش این است که KO به صورت پیش فرض این عنصر را نیز به عنوان پارامتر دوم به توابع گرداننده رویداد پاس می‌دهد. برای مثال:

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

<script type="text/javascript">
  var viewModel = {
    myFunction: function(data, event) {
      if (event.shiftKey) {
        // ...
      } else {
        // ...
      }
    }
  };
  ko.applyBindings(viewModel);
</script>
```

تابع myFunction در مثال بالا دارای دو پارامتر است. پارامتر دوم در این تابع به عنوان عنصر فرستنده رویداد مورد استفاده قرار خواهد گرفت. بدین ترتیب در توابع event Handler می‌توان به راحتی اطلاعات مورد نیاز درباره آبجکت رویداد را به دست آورد.

**\*نکته سوم:** به صورت پیش فرض KO از اجرای عملیات پیش فرض رویدادها جلوگیری به عمل می‌آورد. این به این معنی است که اگر برای رویداد کلیک تگ a یک تابع گرداننده تعریف کرده باشید، بعد از کلیک بر روی این المان؛ مرورگر فقط این تابع تعریف شده توسط شما را فراخوانی خواهد کرد و دیگر عملیات راهبری به صفحه مورد نظر در خاصیت href صورت نخواهد گرفت. اگر به هر دلیلی قصد داشته باشیم که این رفتار صورت نگیرد کافیست در انتهای تابع گرداننده رویداد مقدار true برگشت داده شود.

**\*نکته چهارم:** مفهوم clickBubble

ابتدا به کد زیر دقت کنید:

```
<div data-bind="click: myDivHandler">
  <button data-bind="click: myButtonHandler">
    Click me
  </button>
</div>
```

همان طور که مشاهده می‌کنید در کد بالا برای عنصر button یک رویداد کلیک تعریف شده است. از طرف دیگر این button درون تگ div قرار دارد که برای این تگ نیز این رویداد کلیک با تابع گرداننده متفاوتی تعریف شده است. نکته این جاست که به صورت پیش فرض بعد از فراخوانی رویداد کلیک عنصر داخلی، رویداد کلیک عنصر خارجی نیز فراخوانی خواهد شد. به این رفتار event bubbling می‌گویند. اگر قصد داشته باشیم که این رفتار را غیر فعال کنیم (یعنی با کلیک بر روی button، رویداد کلیک تگ div اجرا نشود باید مقدار خاصیت clickBubble رویداد عنصر داخلی را برابر false قرار دهیم) به صورت زیر:

```
<div data-bind="click: myDivHandler">
  <button data-bind="click: myButtonHandler, clickBubble: false">
    Click me
  </button>
</div>
```



## نظرات خوانندگان

نویسنده: مهرداد اشکانی  
تاریخ: ۱۵:۵۷ ۱۳۹۲/۰۷/۰۳

عالی بود دوست عزیز خیلی استفاده کردیم

نویسنده: سعید  
تاریخ: ۲۳:۵۸ ۱۳۹۲/۱۰/۱۶

سلام ضمن تشکر از اراپه آموزش فارسی مفیدتون که باعث میشه زمان کمتری برای درک مفهوم صرف بشه اما لطفا در صورت امکان روند را بایک فرآیند عملی قابل درک توضیح دهید چون بعضی اصطلاحات فارسی نمیتونه گویا باشد مثلا در نکته دوم جملات برای من واضح نبود و چون مثال عملی نیست مجبورم که به مطالب زبان اصلی مراجعه کنم تا مفهوم را بهتر درک کنم. من آموزش قبلیتون که با مثال بود را بخوبی درک کردم و از کاراتون تشکر میکنم

با پیشرفت HTML 5 و پدید آمدن چارچوب‌های مختلف JavaScript توسعه‌ی نرم افزارهای تک صفحه‌ای تحت وب (Single Page Applications) محبوب شده است. اخیراً مطالب خوبی در رابطه با AngularJS در وبسایت جاری منتشر شده است. KnockoutJS توسط Microsoft معرفی شد و در قالب پیشفرض پروژه‌های SPA قرار گرفت، بنابراین احتمالاً این سوال برای افرادی مطرح شده است که تفاوت بین KnockoutJS و AngularJS چیست؟ می‌توان پاسخ داد این مقایسه ممکن نیست.

KnockoutJS: یک پیاده‌سازی مستقل JavaScript از الگوی MVVM با امکانات Databinding می‌باشد. Knockout یک کتابخانه‌ی Databinding است نه یک کتابخانه‌ی SPA

AngularJS: طبق معرفی در [این مطلب](#) AngularJS فریم ورکی متن باز و نوشته شده به زبان جاوا اسکریپت است. هدف از به وجود آمدن این فریم ورک، توسعه هر چه ساده‌تر SPAها با الگوی طراحی MVC و تست پذیری هر چه آسان‌تر آنها است. این فریم ورک توسط یکی از محققان Google در سال 2009 به وجود آمد. بعدها این فریم ورک تحت مجوز MIT به صورت متن باز در آمد و اکنون گوگل آن را حمایت می‌کند و توسط هزاران توسعه دهنده در سرتاسر دنیا، توسعه داده می‌شود.

بنابراین شاید بهتر باشد ذکر شود AngularJS یک Presentation Framework مخصوص برنامه‌های وب تک صفحه‌ای می‌باشد در حالی که KnockoutJS کتابخانه‌ای با تمرکز بر Databinding می‌باشد، بنابراین مقایسه‌ی این‌ها چندان صحیح نیست.

اگر قصد بر بررسی گزینه‌های دیگر در کنار Angular باشد، می‌توان از [Durandal](#) نام برد. Durandal یک چارچوب SPA می‌باشد، این چارچوب بر فراز [RequireJS](#)، jQuery و Knockout توسعه پیدا کرده است. (سابقاً برای routing از SammyJS استفاده می‌کرد که در نسخه‌های اخیر از موتور خودش استفاده می‌کند).



jQuery

Require

Knockout

jQuery /  
jqLite

Durandal از Knockout جهت Databinding و از RequireJS برای مدیریت وابستگی‌ها استفاده می‌کند. Angular همه‌ی امکانات بالا را مستقل پیاده سازی کرده و حتی نیازی به jQuery ندارد. اگر jQuery وجود داشته باشد Angular از آن استفاده می‌کند در غیر این صورت از jQuery Lite یا jqLite استفاده می‌کند. jqLite پیاده سازی توابع متداول jQuery برای دستکاری DOM می‌باشد. اطلاعات بیشتر در [اینجا](#)

بنابراین با استفاده تنها از KnockoutJS نمی‌توان یک برنامه‌ی کامل SPA توسعه داد ، در کنار آن نیاز به کتابخانه‌های دیگری مثل jQuery برای مدیریت درخواست‌های AJAX و استفاده از دیگر API ها ، Sammy برای routing و RequireJS برای مدیریت وابستگی‌ها می‌باشد.

در Knockout و در نتیجه Durandal عمل Databinding به این صورت است :

```
// JavaScript
var vm = {
  firstName = ko.observable('John')
};
ko.applyBindings(vm);
```

```
<!-- HTML -->
<input data-bind="value:firstName"/>
```

در Angular :

```
// JavaScript
// Inside of a personController
this.firstName = 'John';
```

در Angular همچنین از یک روش [Controller As](#) استفاده می‌شود :

```
<!-- HTML -->
<div ng-controller="personController as vm">
  <input ng-model="vm.firstName"/>
</div>
```

اگر تنها نیاز به یک کتابخانه‌ی Databinding باشد ، Knockout گزینه‌ی مناسبی است ، به خوبی از عمل مقید سازی داده‌ها پشتیبانی می‌کند و Syntax خوش دستی دارد اما اگر نیاز به چارچوبی برای توسعه‌ی پروژه‌های SPA می‌باشد می‌توان از Angular یا Durandal استفاده کرد.

مقایسه‌ی Knockout با Angular همانند مقایسه‌ی موتور بنز با ماشین پورشه می‌باشد.

[مطالعه‌ی بیشتر](#)

## نظرات خوانندگان

نویسنده: محسن خان  
تاریخ: ۱۳۹۲/۱۰/۱۱ ۱:۱۱

برای مطالعه بیشتر: [سری 8 قسمتی AngularJS vs Knockout](#)

نویسنده: mohammad sepahvand  
تاریخ: ۱۳۹۲/۱۱/۲۱ ۱۰:۲۹

به نظر من مقایسه angular و knockout آنقدر هم احمقانه نیست. اگر بخواهیم فقط هم از data binding استفاده کنیم angular خیلی از knockout خوش دست‌تر و ساده‌تر است. تازگی angular بیشتر modular شده و بنابراین مقایسه این دو مانند مقایسه موتور بنز با خود پورشه نیست، چون اگر تنها نیازمان data-binding است لزومی ندارد از module های دیگر angular مانند ng-animate, ng-route ..... استفاده کنیم و حتی نیازی نیست آن اسکریپت‌ها را در پروژه خود include کنیم.

نویسنده: شاهین کیاست  
تاریخ: ۱۳۹۲/۱۱/۲۱ ۱۰:۴۳

در واقع زمانی که تنها از ماژول Data binding استفاده می‌شود یعنی به عنوان مثال تنها از موتور بنز استفاده شده .

نویسنده: خیام  
تاریخ: ۱۳۹۲/۱۱/۲۱ ۱۳:۴۳

حالا که زحمت مقایسه AngularJS و knockout رو انجام دادین ، بهتر بود Angular رو با یک فریم ورک قویتری مثل Ember مقایسه کنید و از این دو سخن بگید ؟ نظر شما در مورد این دو چی هست ؟

نویسنده: محسن خان  
تاریخ: ۱۳۹۲/۱۱/۲۱ ۱۷:۳

[فاکتورهایی را که باید حین انتخاب یک فریم‌ورک JavaScript MVC در نظر داشت](#)

نویسنده: شاهین کیاست  
تاریخ: ۱۳۹۲/۱۱/۲۱ ۱۸:۲۱

مقایسه از این قبیل [زیاد است](#)

اگر نگاهی به جامعه کاربری استفاده کننده کنیم به طور مثال در Stackoverflow با تگ Angular حدود 25 هزار سوال پرسیده شده در حالی که با تگ Backbone حدود 14 هزار سوال پرسیده شده. Angular امکانات کاملی برای توسعه‌ی SPA در بر دارد.

نویسنده: سعید رضایی  
تاریخ: ۱۳۹۲/۱۲/۲۰ ۱۶:۴۳

با عرض سلام.  
angularjs با مرورگر ie 11 به پایین مشکل داره..

نویسنده: وحید نصیری  
تاریخ: ۱۳۹۲/۱۲/۲۰ ۱۷:۰

خیر؛ با IE 9 به بعد مشکلی ندارد. با IE8 هم کار می‌کند ولی یک سری نکات خاص خودش را دارد. اطلاعات بیشتر را در [مستندات رسمی آن در مورد IE](#) مطالعه کنید.

## Custom Binding در KO

در پست‌های قبلی ([^](#) و [^](#) و [^](#)) با انواع مقید سازی در KO آشنا شدید. اما در پیاده سازی، محدود به این نوع‌هایی، click، value، text و ... نیستیم؛ بلکه می‌توانیم نوع مورد نظر برای عملیات مقید سازی را بنابر نیاز خود بسازیم که به آن‌ها Custom Binding گفته می‌شود. Custom Binding یکی از امکانات قدرتمند موجود در KO است و مورد اصلی استفاده آن در طراحی کامپوننت‌ها و ویجت‌ها می‌باشد.

### مکانیزم پیاده سازی Custom Binding

برای شروع باید binding مورد نظر، به خاصیت ko.bindingHandlers رجیستر شود. سپس با تعیین کردن و شخصی سازی دو تابع init و update می‌توان نوع مقید سازی مورد نظر را تعریف کرد.  
«init: این تابع فقط یک بار آن هم به ازای هر عنصری که عملیات مقید سازی را شامل می‌شود، فراخوانی خواهد شد.  
«update: این تابع برای تعیین نوع عمل مورد انتظار در هنگام تغییر کردن مقدار عنصر DOM استفاده می‌شود.  
برای مثال:

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

### پارامترهای توابع:

هر دو تابع بالا دقیقاً دارای پنج پارامتر یکسان هستند که در زیر به تفصیل شرح داده شده‌اند:  
«element: برای دسترسی مستقیم به عنصر DOMی که شامل مقید سازی است، می‌توان از این پارامتر استفاده کرد.  
«valueAccessor: این پارامتر تابعی است که امکان دسترسی به هر آنچه را که به binding مورد نظر پاس داده باشیم، در اختیار ما قرار می‌دهد. برای مثال اگر observable را پاس داده باشیم، خروجی این تابع دقیقاً همان observable خواهد بود. اگر از یک عبارت یا expression استفاده کرده باشیم خروجی این تابع برابر با حاصل آن عبارت خواهد بود.  
«allBindingsAccessor: برای پیدا کردن لیست تمام عناصری است که به یک data-bind attribute مشترک اشاره می‌کنند.  
«viewModel: برای دسترسی به viewModel عنصر مقید شده استفاده می‌شود. در knockout نسخه 3 به بعد این گزینه منسوخ شده است. به جای آن باید از پارامتر bindingContext.\$data یا bindingContext.\$rowData استفاده کرد.  
«bindingContext: این پارامتر شی binding Context را که عنصر مورد نظر به آن مقید شده است، شامل می‌شود. این آبجکت شامل خواص \$parent و \$parents و \$root است.

### یک مثال ساده:

```
ko.bindingHandlers.jqButton= {
  init: function(element, valueAccessor) {
    var options = valueAccessor() || {};
    $(element).button(options);
  }
};
```

و روش استفاده از آن در عناصر DOM:

```
<button data-bind="click: greet, jqButton: { icons: { primary: 'ui-icon-gear' } }">Test</button>
```

[دموی این مثال](#)

**استفاده از تابع update :**

فرض کنید قصد داریم که با تغییر در مقدار یک متغیر، تغییرات مورد نظرمان در عنصر مقید شده نیز مشاهده شود. در این حالت باید از تابع update استفاده نمود. به مثال زیر دقت کنید:

```
ko.bindingHandlers.flash= {
  update: function(element, valueAccessor) {
    ko.utils.unwrapObservable(valueAccessor());
    $(element).hide().fadeIn(500);
  }
};
```

نکته : دستور ko.utils.unwrapObservable خاصیت مورد نظر را از حالت observe بودن خارج می‌کند.

[دموی این مثال](#)

ادامه دارد...

## نظرات خوانندگان

نویسنده:

موحدی نیا

تاریخ:

۱۰:۴۵ ۱۳۹۳/۰۲/۰۴

با سلام و تشکر از مطالب مفیدی که تو سایت قرار میدید من یه پروژه case study رو چند روزی هست که شروع کردم و بدون مشکل کارم رو ادامه میدادم تا اینکه به ویو ویرایش مشخصات افراد رسیدم. سه تا از فیلدهای مربوط به افراد شامل کشور، استان و شهر میشه که تو View مربوط به افراد جدید این سه تا DropDownList با استفاده از Knockout پر میشن. بطوری که DropDownListهای مربوط به استان و شهر خالی هستن و با انتخاب کشور، استان پر میشه و با انتخاب استان، شهر پر میشه. مشکل اینجاست که تو View ویرایش DropDownListهای استان و شهر در بارگذاری اولیه فرم پر نمیشن ولی با تغییر مقادیر کشور، استانها در DropDownList خودش پر میشه و این کار برای شهر هم به خوبی انجام میشه. حالا میخام ببینم که چطور میشه این مشکل رو حل کرد

نویسنده:

مسعود پاکدل

تاریخ:

۱۲:۴۸ ۱۳۹۳/۰۲/۰۴

برای اینکه بتوان پاسخ به سوال شما را بدون حدس گمان و به صورت قطعی بیان کنم لطفا کدهای مورد نظر را قرار بدید!

<< [باگ را بدون باگ گزارش کن](#) >><< [آناتومی یک گزارش خطای خوب](#) >>

## پیاده سازی Extender

همان طور که در [پست‌های و مثال‌های قبلی](#) مشاهده شد با استفاده از Ko.Observable توانستیم عملیات مقید سازی را به کمک ویژگی‌های خواندن و نوشتن ساده، پیاده سازی نماییم. اما قصد داریم در طی عملیات نوشتن به جای یک tracking ساده تغییرات، بتوانیم یک سری عملیات مشخص را نیز اجرا نماییم. چیزی شبیه به AOP دنیای back-end. یعنی بتوانیم کد اصلی برنامه را در هنگام عملیات خواندن و نوشتن خاصیت‌ها، با یک سری کد مورد نظر مزین نماییم. برای این کار مفهوم extender در KO تعبیه شده است.

برای ساخت یک extender کافیت تابع مورد نظر را به عنوان آرگومان به شی ko.extenders پاس دهیم. پارامتر اول این تابع، شی observable شده مورد نظر و پارامتر دوم آن، شی option برای انجام یک سری تنظیمات یا فرستادن مقادیر مورد نظر به تابع است. خروجی این تابع نیز می‌تواند یک شی observable یا حتی یک شی [computed/observable](#) نیز باشد. یک مثال ساده برای extenderها به صورت زیر است:

```
ko.extenders.logOpt = function(target, option) {
    target.subscribe(function(newValue) {
        console.log(option + ": " + newValue);
    });
    return target;
};
```

در مثال بالا با ایجاد یک extender برای شی target که خود آن به عنوان آرگومان به تابع پاس داده می‌شود، به ازای هر تغییر در مقدار شی target، مقدار جدید نیز در console نمایش داده خواهد شد. مقدار چاپ شده در console برابر است با مقدار شی option + مقدار جدید شی target به ازای هر تغییر.

برای استفاده از این extender کافیت آن را در هنگام تعریف تابع observable برای خواص، به صورت زیر فراخوانی نمایید:

```
this.firstName = ko.observable("Masoud").extend({logOpt: "my first name"});
```

مقدار 'my first name' همان مقدار پاس داده شده در قالب شی option است. در نتیجه خروجی console به صورت زیر خواهد شد:

```
my first name : Masoud
```

## پیاده سازی یک extender جهت اعلام هشدار برای مقادیر منفی

برای اینکه هنگام ورود داده‌ها توسط کاربر، بتوانیم با ورود مقادیر منفی یک هشدار (تغییر رنگ ورودی) اعلام کنیم، می‌توان به صورت زیر عمل نمود:

```
ko.extenders.negativeValueWarn = function (target, option) {
    target.hasWarning = ko.observable();

    function warn(newValue) {
        if(newValue && newValue.substring) {
            newValue = parseFloat(newValue);
        }
        target.hasWarning(newValue < 0 ? true : false);
    }

    warn(target());
    target.subscribe(warn);
};
```



```
return target;
};
```

تابع warn با در اختیار داشتن مقدار جدید و بررسی منفی یا مثبت بودن آن نتیجه را به تابع set شی hasWarning ارسال می‌کند.

**یاد آوری :** در KO برای انتساب مقدار جدید به خواصی که به صورت observable تعریف شده اند به صورت زیر:

فراخوانی به صورت تابع و پاس دادن مقدار جدید به آن => target(NewValue)

و برای به دست آوردن این مقادیر از اشیای Observable به صورت زیر عمل می‌نماییم:

فراخوانی به صورت تابع بدون آرگومان => target()

### [خروجی مثال بالا](#)

#### پیاده سازی یک extender برای انتساب مقادیر Boolean به Radio Button ها

برای اینکه radio button ها نیز بتوانند فقط با مقادیر Boolean مقدار دهی شوند و از طرفی در هنگام عملیات مقید سازی و ارسال نتایج در قالب شی Json به سرور، بدون هیچ گونه تغییر و محاسبات مقادیر مورد نظر به صورت true/false (از نوع Boolean) باشند می‌توان به صورت زیر عمل نمود:

```
ko.extenders["booleanValue"] = function (target) {
    target.formattedValue = ko.computed({
        read: function () {
            if (target() === true) return "True";
            else if (target() === false) return "False";
        },
        write: function (newValue) {
            if (newValue) {
                if (newValue === "False") target(false);
                else if (newValue === "True") target(true);
            }
        }
    });
    target.formattedValue(target());
    return target;
};
```

در کد بالا یک sub-observable به نام formattedValue ایجا شده است و همان طور که ملاحظه می‌نمایید از نوع computed می‌باشد. در تابع read آن (هنگام عملیات مقید سازی برای خواندن مقادیر) اگر مقدار مورد نظر برابر با true از نوع boolean بود مقدار True (به صورت string) و اگر برابر با false بود مقدار False برگشت داده می‌شود. هنگام عملیات write بر عکس عمل خواهد شد.

با فرض اینکه کدهای Html صفحه به صورت زیر است:

```
<span>Do you want fries with that?</span>
<label>
    <input type="radio" name="question" value="True"
        data-bind="value: myValue.formattedValue" /> Yes
</label>
<label>
    <input type="radio" name="question" value="False"
        data-bind="value: myValue.formattedValue" /> No
</label>
```

Json Object مورد نظر که مقادیر boolean در آن به صورت true یا false است و بعد از عملیات مقید سازی در هنگام انتساب مقادیر، آن‌ها را تبدیل به True یا False برای المان‌های Html می‌کند. و در هنگام ورود اطلاعات توسط کاربر و انتساب آن‌ها به شی Json ، مقادیر تبدیل به true یا false از نوع boolean خواهند شد. برای استفاده از آن کافیت به صورت زیر عمل نمایید:

```
this.myValue= ko.observable(false).extend({ booleanValue: null });
```