عنوان: **آموزش Knockout.Js #3**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۶/۰۹ ۸:۴۵
آدرس: www.dotnettips.info
برچسب‌ها: KnockoutJS

در ادامه مباحث قبلی ، در این پست به بررسی سایر قابلیت‌های Observableها در KO خواهم پرداخت.

Computed Observables

Computed Observablesها به واقع خواصی هستندکه 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 تایع 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 صرفا 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: رحیمي
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 اخرین سطر میاره

نویسنده: رحیمي
تاریخ: ۱۳۹۲/۰۶/۱۳ ۱۲:۲۳

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