آموزش KnockoutJS ـ آموزش foreach binding در KnockoutJS

این binding به ازای هر یک از ورودی های موجود در یک آرایه، بخشی از markup را کپی می کند و هر یک از بخش های کپی شده ی این markup را به آیتم آرایه ای متناظر مقید می کند. این binding در نمایش دادن لیست ها یا جدول ها کاربرد ویژه ای دارد.

فرض کنید که آرایه ی شما یک آرایه ی قابل مشاهده باشد، هر زمان که شما بخواهید ورودی های آرایه را اضافه یا حذف کنید و یا اینکه آن ها را مجددا مرتب کنید، در این صورت این binding با وارد کردن یا حذف کردن کپی های بیشتری از این markup یا مرتب سازی مجدد المان های موجود DOM بدون آنکه دیگر المان های DOM تحت تاثیر قرار گیرند، برای انجام منطبق سازی، رابط کاربری را آپدیت می کند.

این عملیات نسبت به تولید مجدد کل خروجی foreach بعد از هر بار تغییر آرایه سرعت بسیار بیشتری دارد.

در صورت نیاز می توانید به صورت دلخواه هر یک از اعداد foreach binding را همراه با دیگر binding های جریان کنترل مانند if و with به صورت تو در تو استفاده کنید.

مثال 1 : ایجاد تکرار در یک آرایه

در این مثال برای ایجاد یک جدول read-only همراه با ردیفی برای هر یک از ورودی های آرایه از foreach استفاده شده است.

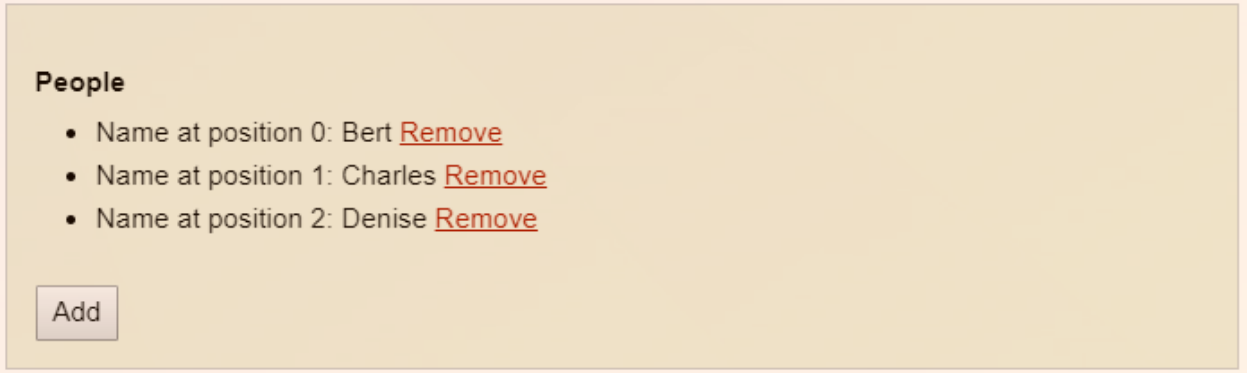
```
<table>
  <thead>
    <tr><th>First name</th><th>Last name</th></tr>
  </thead>
  <tbody data-bind="foreach: people">
    <tr>
      <td data-bind="text: firstName"></td>
      <td data-bind="text: lastName"></td>
    </tr>
  </tbody>
</table>

<script type="text/javascript">
  ko.applyBindings({
    people: [
      { firstName: 'Bert', lastName: 'Bertington' },
      { firstName: 'Charles', lastName: 'Charlesforth' },
      { firstName: 'Denise', lastName: 'Dentiste' }
    ]
```

```
});
</script>
```

مثال 2 : مثالی زنده با قابلیت حذف و اضافه کردن

این مثال نشان می دهد که اگر آرایه ی شما قابل مشاهده باشد، در اینصورت رابط کاربری با تغییرات اعمال شده بر روی این آرایه هماهنگ خواهد بود.



Source code: View

```
<h4>People</h4>
<ul data-bind="foreach: people">
  <li>
    Name at position <span data-bind="text: $index"> </span>:
    <span data-bind="text: name"> </span>
    <a href="#" data-bind="click: $parent.removePerson">Remove</a>
  </li>
</ul>
<button data-bind="click: addPerson">Add</button>
```

Source code: View model

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

  self.people = ko.observableArray([
    { name: 'Bert' },
    { name: 'Charles' },
    { name: 'Denise' }
  ]);
```

```
  self.addPerson = function() {
    self.people.push({ name: "New at " + new Date() });
  };

  self.removePerson = function() {
    self.people.remove(this);
  }
}

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

پارامترها

- پارامتر اصلی

آرایه ای که می خواهید بر روی آن عملیات تکرار را انجام دهید را وارد کنید. این binding به ازای هر یک از ورودی ها بخشی از markup را خروجی می دهد.

به عنوان جایگزین می توانید یک شیء لفظی جاوا اسکریپت را با مشخصه ای به نام data وارد کنید. به گونه ای که این مشخصه آرایه ای است که شما می خواهید عملیات تکرار را بر روی آن انجام دهید. این شیء لفظی ممکن است مشخصه های دیگری هم داشته باشد، مانند afterAdd و includeDestroyed _ جزئیات بیشتر در رابطه با option های بیشتر این چنینی و مثال هایی در رابطه با کاربرد آن ها در ادامه آمده است.

اگر آرایه ای که شما وارد می کنید قابل مشاهده باشد، در اینصورت foreach binding با اضافه کردن یا حذف کردن بخش های متناظر markup در DOM نسبت به تمامی تغییراتی که در آینده بر محتوای این آرایه اعمال می شود، واکنش نشان خواهد داد.

- پارامترهای اضافی
- ندارد

نکته 1 : اشاره به هر یک از ورودی های آرایه با استفاده از $data

همانطور که در مثال های بالا نشان داده شد binding های داخل بلوک foreach می توانند به مشخصه های آرایه های ورودی اشاره داشته باشند. برای مثال، مثال 1 در هریک از آرایه های ورودی به مشخصه های firstName و lastName اشاره دارد.

اما در صورتی که بخواهید به خود آرایه ی ورودی اشاره کنید (نه فقط به مشخصه های آن)، باید چه کار کنید؟ در این صورت می توانید از مشخصه ی ویژه ی متنی (special context property) $data داخل یک بلوک foreach (در مثال زیر این کار بر روی "the current item" انجام شده است)استفاده کنید. برای مثال :

```
<ul data-bind="foreach: months">
```

```
  <li>
    The current item is: <b data-bind="text: $data"></b>
  </li>
</ul>

<script type="text/javascript">
  ko.applyBindings({
    months: [ 'Jan', 'Feb', 'Mar', 'etc' ]
  });
</script>
```

در صورت تمایل در زمان اشاره به مشخصه های هر یک از ورودی ها می توانید از $data به صورت یک پیشوند استفاده کنید. برای مثال می توانید مثال 1 را مانند زیر بازنویسی کنید:

```
<td data-bind="text: $data.firstName"></td>
```

اما الزامی برای انجام این کار نیست. زیرا به هر حال به صورت پیش فرض firstName داخل متن $data ارزیابی خواهد شد.

نکته 2 : استفاده از $index, $parent و دیگر مشخصه های متنی

همانطور که در مثال 2 می بینید می توان برای اشاره به شاخص مبتنی بر صفر آیتم آرایه ای فعلی از $index استفاده کرد. در اینجا $index یک observable بوده و هر زمان که شاخص آیتم تغییر کند (مثلا اگر آیتم ها به آرایه اضافه شوند یا از آن حذف شوند) آپدیت خواهد شد. به طور مشابه می توانید برای اشاره به data از خارج از foreach از $parent استفاده کنید. برای مثال:

```
<h1 data-bind="text: blogPostTitle"></h1>
<ul data-bind="foreach: likes">
  <li>
    <b data-bind="text: name"></b> likes the blog post <b data-bind="text:
$parent.blogPostTitle"></b>
  </li>
</ul>
```

برای اطلاعات بیشتر در رابطه با $index و دیگر مشخصه های متنی مانند $parent به سند مقید سازی مشخصه های متنی مراجعه کنید.

مثال 3 : استفاده از "as" برای دادن نام مستعار (alias) به هر یک از آیتم های "foreach"

همانطور که در نکته 1 بیان شد می توانید با استفاده از متغیر متنی $data به هر یک از آرایه های ورودی اشاره کنید. هرچند ه در بعضی از موارد بهتر است که با استفاده از آپشن as اسم آیتم موجود را توصیفی تر کنید.

```
<ul data-bind="foreach: { data: people, as: 'person' }"></ul>
```

حالا در همه ی بخش های داخلی حلقه ی foreach ، binding ها قادر خواهند بود تا برای دسترسی به آیتم آرایه ای فعلی از طریق آرایه ی people که در حال رندر شدن است به person اشاره کند. این حالت به ویژه در مواقعی که از بلوک های تو در توی foreach استفاده کرده باشید و نیاز باشد که به آیتمی اشاره کنید که در سطح بالاتری از سلسله مراتب اعلان شده است مفید است. برای مثال

```
<ul data-bind="foreach: { data: categories, as: 'category' }">
  <li>
    <ul data-bind="foreach: { data: items, as: 'item' }">
      <li>
        <span data-bind="text: category.name"></span>:
        <span data-bind="text: item"></span>
      </li>
    </ul>
  </li>
</ul>

<script>
  var viewModel = {
    categories: ko.observableArray([
      { name: 'Fruit', items: [ 'Apple', 'Orange', 'Banana' ] },
      { name: 'Vegetables', items: [ 'Celery', 'Corn', 'Spinach' ] }
    ])
  };
  ko.applyBindings(viewModel);
</script>
```

توجه: به خاطر داشته باشید که در as از مقدار لفظی رشته ای استفاده کنید (مانند: as: 'category' نه به صورت as: category)، زیرا شما مقدار متغیری که در حال حاضر موجود است را نمیخوانید بلکه در حال دادن اسمی به متغیر جدید هستید.

نکته 4 : استفاده از foreach بدون المان نگهدارنده

در برخی از موارد می توانید بخشی از markup را کپی کنید. اما هیچ المان نگهدارنده ای برای اینکه foreach binding را در آن قرار دهید، ندارید. برای مثال فرض کنید بخواهید که کد زیر را ایجاد کنید:

```
<ul>
  <li class="header">Header item</li>
```

```
  <!-- The following are generated dynamically from an array -->
  <li>Item A</li>
  <li>Item B</li>
  <li>Item C</li>
</ul>
```

در این مثال هیچ جایی وجود ندارد که بتوانید یک foreach binding معمولی را در آن قرار دهید. این binding را نه می توانید در <ul> قرار دهید (زیرا در اینصورت آیتم header را کپی میکنید) و نه می توانید نگهدارنده ی دیگری را داخل <ul> قرار دهید (زیرا تنها استفاده از المان های <li> داخل <ul> مجاز است).

برای حل این مشکل می توانید از سینتکس جریان کنترل بدون نگهدارنده استفاده کنید. که این سینتکس بر اساس تگ های توضیحی انجام می شود. برای مثال :

```
<ul>
  <li class="header">Header item</li>
  <!-- ko foreach: myItems -->
    <li>Item <span data-bind="text: $data"></span></li>
  <!-- /ko -->
</ul>

<script type="text/javascript">
  ko.applyBindings({
    myItems: [ 'A', 'B', 'C' ]
  });
</script>
```

کامنت های <!-- ko --> و <!-- /ko --> به عنوان نشانگرهای آغازین و پایانی عمل میکنند. به این صورت که المانی مجازی را تعریف می کنند که در داخل خود شامل markup هستند. KnockoutJS سینتکس این المان مجازی را درک کرده و درست مانند وقتی که شما یک المان نگهدارنده ی واقعی داشته باشید، مقید می شود.

نکته 5 : چگونه تغییرات آرایه شناسایی و مدیریت می شوند

زمانی که محتویات آرایه ی مدل خود را اصلاح کنید(با اضافه کردن، حرکت دادن یا پاک کردن ورودی های آن) در اینصورت foreach binding برای اینکه بفهمد چه چیزی تغییر کرده است از الگوریتم بهینه ی متمایز کننده ای استفاده می کند. به گونه ای که این binding برای ایجاد انطباق DOM را آپدیت می کند. این یعنی این binding می تواند ترکیب های دلخواه تغییرات همزمان را مدیریت کند.

- زمانی که آرایه های ورودی را اضافه می کنید، در اینصورت foreach کپی های جدیدی از قالب شما را نمایش داده و آن ها را در DOM موجود وارد می کند.
- زمانی که آرایه های ورودی را حذف می کنید، در اینصورت foreach به سادگی المان های متناظر DOM را حذف می کند.
- زمانی که آرایه های ورودی را مجددا مرتب می کنید (reorder) (نمونه اشیاء قبلی را نگه می دارید)، در اینصورت foreach معمولا فقط المان های متناظر DOM را به مکان جدید خود انتقال می دهد.

توجه داشته باشید که تضمینی در شناسایی عمل reordering وجود ندارد. برای اینکه مطمئن شوید که این الگوریتم سریع کار خود را تمام می کند، بهتر است که جا به جایی های ساده ی تعداد کمی از آرایه های ورودی را شناسایی کنید. اگر این الگوریتم تعداد زیادی از reordering های همزمان را در کنار پاک کردن ها و اضافه کردن های نامرتبط شناسایی کند، در اینصورت می توانید برای بالا بردن سرعت آن، در reordering خود به جای اینکه تنها از جابه جایی استفاده کنید، به صورت همزمان از حذف کردن و اضافه کردن بهره ببرید. در این صورت است که المان های متناظر DOM از بین می روند و مجددا ایجاد می شوند. تعداد کمی از برنامه نویسان با این مشکل مواجه می شوند. اگر شما با این مشکل روبرو شوید در این صورت تجربه کاربر نهایی که پشت سر خواهید گذاشت معمولا دقیقا به همین صورت است.

نکته 6 : ورودی های ازبین رفته به صورت پیش فرض مخفی هستند

ممکن است گاهی اوقات بخواهید که بدون اینکه سابقه ی موجودیت آرایه ی ورودی را از دست بدهید، این آرایه را به عنوان "پاک شده" نشان کنید. به این کار پاک کردن غیرمخرب گفته می شود. برای دریافت اطلاعات بیشتر در رابطه با چگونگی انجام این کار به تابع destroy در observableArray مراجعه کنید.

به صورت پیش فرض foreach binding از روی همه ی آرایه های ورودی که به عنوان "نابود شده" نشان شده اند، عبور می کند (یا به عبارتی آن ها را مخفی می کند). گر بخواهید ورودی های از بین رفته را نمایش دهید می توانید از آپشن includeDestroyed استفاده کنید. برای مثال:

```
<div data-bind='foreach: { data: myArray, includeDestroyed: true }'>
    ...
</div>
```

نکته 7 : پسا پردازش یا متحرک سازی المان های DOM تولید شده

در صورتی که بخواهید در المان های تولید شده ی DOM منطق های شخصی بیشتری را اجرا کنید، می توانید مانند زیر از هریک از توابع callback ، afterRender/afterAdd/beforeRemove/beforeMove/afterMove استفاده کنید.

نکته: این توابع تنها برای فعالسازی انیمیشن های مرتبط با تغییرات موجود در یک لیست استفاده می شوند. اگر هدف شما این است که بعد از اضافه شدن المان های DOM جدید رفتارهای دیگری را به این المان ها متصل کنید (برای مثال event handler ها، یا برای فعالسازی کنترل های سوم شخص رابط کاربری) در این صورت اگر

این رفتار جدید را در عوض به عنوان یک custom binding پیاده سازی کنید کارتان بسیار ساده تر خواهد شد. زیرا در این صورت می توانید مستقل از foreach binding این رفتار را در همه جا به کار ببرید.

در اینجا مثال ساده ای نمایش داده شده است که برای اعمال اثر "yellow fade" به آیتم هایی که جدیدا اضافه شده اند از afterAdd استفاده شده است. برای فعالسازی انیمیشن رنگ های پس زمینه به پلاگین jQuery Color نیاز دارید.

```
<ul data-bind="foreach: { data: myItems, afterAdd: yellowFadeIn }">
  <li data-bind="text: $data"></li>
</ul>

<button data-bind="click: addItem">Add</button>

<script type="text/javascript">
  ko.applyBindings({
    myItems: ko.observableArray([ 'A', 'B', 'C' ]),
    yellowFadeIn: function(element, index, data) {
      $(element).filter("li")
            .animate({ backgroundColor: 'yellow' }, 200)
            .animate({ backgroundColor: 'white' }, 800);
    },
    addItem: function() { this.myItems.push('New item'); }
  });
</script>
```

جزئیات کامل:

- afterRender : هر زمان که بلوک foreach کپی می شود و در سند وارد می شود، afterRender فراخوانی می شود. این کار هم زمانی که foreach در ابتدا initialize می شود و هم زمانی که ورودی های جدید به آرایه ی مربوطه اضافه می شود، اتفاق می افتد. KnockoutJS پارامترهای زیر را در اختیار تابع callback تان قرار می دهد:
  1. آرایه ای از المان های DOM وارد شده
  2. یک دیتا آیتم در برابر دیتا آیتم هایی که در حال مقید شدن هستند.
- afterAdd : مانند afterRender عمل می کند با این تفاوت که تنها وقتی احضار می شود که ورودی های جدید به آرایه ی شما اضافه شده باشند (نه وقتی که foreach برای بار اول در محتواهای اولیه ی آرایه ی شما کار تکرار را برای اولین بار انجام می دهد). یکی از کاربردهای معمول afterAdd فراخوانی متدی مانند $(domNode).fadeIn() جی کوئری است. به گونه ای که هر زمان که آیتم ها اضافه می

شوند، شما گذارهای متحرک را شاه خواهید بود. KnockoutJS پارامترهای زیر را در اختیار تابع callback تان قرار می دهد:

1. گره ای از DOM که در حال اضافه شدن به سند است.
2. شاخص المان آرایه ی اضافه شده
3. المان آرایه ی اضافه شده

- beforeRemove : زمانی که یک آیتم آرایه پاک می شود، beforeRemove احضار می شود. با این تفاوت که این کار قبل از اینکه گره های متناظر DOM پاک شوند، اتفاق می افتد. فرض کنید که تابع callback، beforeRemove را تعیین کرده باشید، در این صورت پاک کردن گره های DOM بر عهده ی شماست. کاربرد مشهود beforeRemove فراخوانی چیزی مانند $(domNode).fadeOut() مربوط به جی کوئری است. که به کمک ان بتوانید حذف کردن گره های متناظر DOM را متحرک کنید. در این حالت KnockoutJS نمی تواند بفهمد که چه موقعی مجاز است تا به صورت فیزیکی گره های DOM را پاک کند (چه کسی می داند که انیمیشن شما چه مدت طول خواهد کشید). بنابراین پاک کردن این گره ها بر عهده ی شماست. KnockoutJS پارامترهای زیر را در اختیار تابع callback تان قرار می دهد:
  1. یک گره DOM که شما باید آن را پاک کنید.
  2. شاخص المان آرایه ی پاک شده
  3. المان آرایه ی پاک شده
- beforeMove : زمانی احضار میشود که آیتم آرایه ای داخل آرایه تغییر مکان داده باشد. اما این کار قبل از جابه جا شدن گره های متناظر DOM اتفاق می افتد. توجه داشته باشید که beforeMove بر روی تمام المان های آرایه ای اعمال می شود که شاخص آن ها تغییر کرده است. به همین دلیل اگر در ابتدای آرایه ای آیتم جدیدی را وارد کنید، در اینصورت تابع callback (درصورت مشخص بودن) برای تمام المان های دیگر fire می شود. زیرا مکان شاخص آن ها یک درجه افزایش پیدا کرده است. برای ذخیره سازی مختصات صفحه ی اصلی مربوط به المان های تحت تاثیر قرار گرفته، به گونه ای که امکان انیمیشنی کردن جا به جایی آن ها داخل تابع afterMove callback وجود داشته باشد، می توانید از beforeMove استفاده کنید. KnockoutJS پارامترهای زیر را در اختیار تابع callback تان قرار می دهد:
  1. یک گره DOM که ممکن است در آینده جا به جا شود.
  2. شاخص المان آرایه ی جا به جا شده
  3. المان آرایه ی جابه جا شده
- afterMove : بعد از اینکه آیتم آرایه ای داخل آرایه تغییر مکان می دهد، و بعد از آپدیت شدن DOM برای انجام تطابق توسط foreach، afterMove احضار می شود. توجه داشته باشید که afterMove بر روی تمام المان های آرایه ای اعمال می شود که شاخص آن ها تغییر کرده باشد. به همین دلیل اگر شما آیتم جدیدی را در ابتدای یک آرایه وارد کنید، تابع callback (در صورت مشخص بودن) برای تمام المان های دیگر fire می شود. زیرا جایگاه شاخص آن ها یک درجه افزایش پیدا کرده است. KnockoutJS پارامترهای زیر را در اختیار تابع callback تان قرار می دهد:
  1. یک گره DOM که ممکن است جابه جا شده باشد.

2. شاخص المان آرایه ی جا به جا شده
3. المان آرایه ی جا به جا شده

برای مثال های afterAdd و beforeRemove به گذارهای انیمیشنی مراجعه کنید.

وابستگی ها

غیر از کتابخانه ی اصلی KnockoutJS وابستگی دیگری ندارد.