بسم الله الرحمن الرحیم

آموزشگاه تحلیل داده

تخصصی ترین مرکز برنامه نویسی و دیتابیس در ایران

ASP.NET MVC با Knockout.js

مدرس : مهندس افشین رفوآ

ASP.NET MVC با Knockout.js

مقدمه

این مقاله به چهار بخش مهم تقسیم می شود:

اساس Model، View و ViewModel (MVVM)

فقط jQuery در MVC

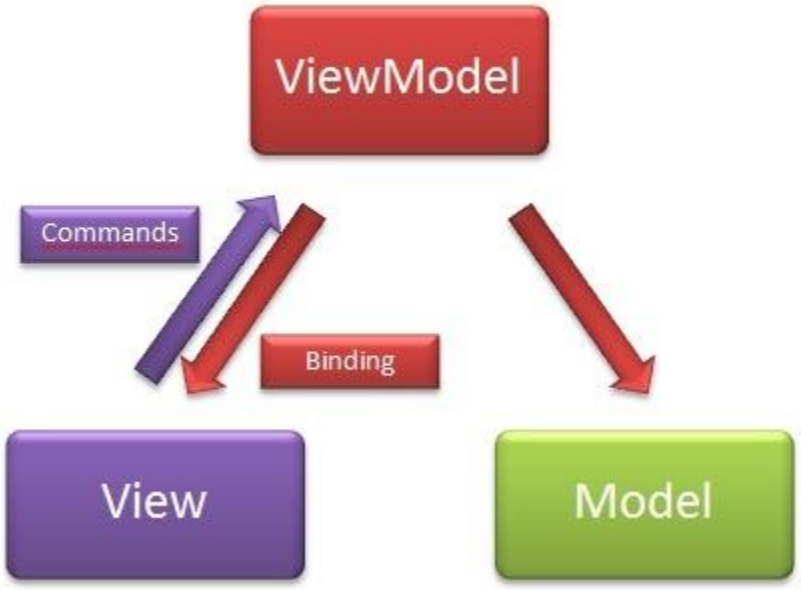
معرفی مختصر Knockout.js

استفاده ساده از Knockout در MVC

اساس MVVM

الگوی طراحی (Design pattern) MVVM در Silverlight/WPF رفته رفته برای توسعه دهندگان ضروری می شود. مبانی معماری MVVM برپایه Martin Fowler's Presentetion Model است که ساختار power MVC و MVP را انعطاف پذیر می کند.

MVVM از سه جزء اصلی تشکیل شده است: Model، View و ViewModel. زمانی که View کاملاً از model بی اطلاع است، ViewModel به Model ارجاع می دهد. در اینصورت توسعه دهنده به هیچ وجه با business logic interface مواجه نمی شود.



نمودار فوق این موقعیت را به خوبی شرح می دهد. Model ممکن است در سرتاسر پروژه تغییر کند. اما View نسبت به این شرایط بی توجه است. Model جدا از View است و ViewModel یک واسطه برای مدیریت binding و دستورات می باشد.

فقط jQuery در MVC

اجازه دهید کمی راجع به jQuery صحبت کنیم. jQuery دارای یک مکانیزم قدرتمند برای binding است که از نام و شناسه ی تگ های HTML و کلاس های CSS، استفاده می کند. جهت ارسال مقادیراز شی منبع (source object) به عناصر HTML، باید یک خط کد برای هر نگاشت از مقدار منبع به عنصر هدف نوشته شود. انجام این کار با استفاده از KO بسیار ساده تر است. چون می توانید بدون ترس از وجود تناقضات، پیچیدگی را تا هر مقیاسی افزایش دهید. Bind کردن jQuery با مقادیر و تگ ها بسیار ساده است.

jQuery در شرایط زیر خوب کار نمی کند:

هر تغییری در سورس اشیاء، عناصر HTML را تحت تاثیر قرار نمی دهد. (می توانید مقادیر را ارسال کنید و دوباره فراخوانی کنید.)

هر تغییری در عناصر HTML، سورس اشیاء را تحت تاثیر قرار نمی دهد.

کد:

```
<h2>
  Using Jquery
  Without Knockout
</h2>
<span>StudentNumber:</span><span id="StudentNumber"></span>
<br />
<span>Surname:</span><input id="StudentSurName" />
<span>Name:</span><input id="StudentName" />
<script type="text/javascript">
  $(document).ready(function () {
    var student = {
      Number: "A123456",
      Surname: "Karatoprak",
      Name: "Yusuf"
    };
    $("#StudentNumber").text(student.Number);
    $("#StudentSurName").val(student.Surname);
    $("#StudentName").val(student.Name);
  });
</script>
```

با استفاده از تگ های HTML و کلاس های CSS قادر خواهید بود مقادیر موردنظرتان را به تگ های HTML متصل (bind) کنید. jQuery یک روش سطح پائین برای مدیریت عناصر و event handler های یک صفحه وب است. در jQuery هیچ مفهومی برای یک data model اساسی ندارد. اگر مانند مثال فوق، مقادیر هر آبجکتی را bind کنید، پس از تغییر Model، نمی توانید تغییرات را در UI مشاهده کنید. باید صفحات وب را تازه سازی (refresh) کنید تا تغییرات view مشاهده شوند. از طرفی، اگر مقادیر عناصر HTML را تغییر دهید Model شما اجرا (fire) نمی شود.

معرفی مختصر Knockout.js

knockout.js چیست؟

KO جایگزین jQuery یا دیگر کتابخانه های js (Prototype، MooTools) نیست. KO با تمرکز بر روی MVVM، Model را، با فراخوانی های AJAX، برای View مدیریت می کند. KO ارتباط خودکار بین ViewModel و View را، که با فراخوانی های رابط کاربری به وجود می آیند، مدیریت می کند.

Model: Business Logic

View: HTML/CSS. اگر اشیاء ViewModel را تغییر دهیم، View به صورت خودکار تحت تاثیر قرار می گیرد.

ViewModel: واسط بین ارتباط Model و View است. چون Model هیچ اطلاعی از View ندارد.

Observable و Bind کردن: تمرکز KO بر روی مفاهیم js داده محور (Data-Driven js) است. به عبارتی ایجاد هر تغییری در View باعث اجرای Model و ایجاد تغییرات در Model باعث بروزرسانی View می‌شود.

KO همیشه برای ارتباط دوطرفه هوشیار است.

کد:

```
<h2>With Knockout</h2>
<span>Student Number:</span><span data-bind="text: Number"></span>
<br />
```

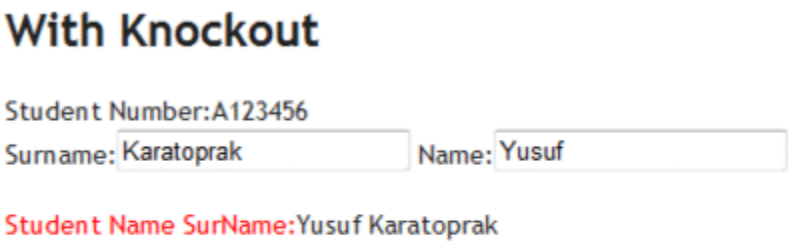
```
<span>Surname:</span><input data-bind="value: Surname" />
<span>Name:</span><input data-bind="value: Name" />
<script type="text/javascript">
  var student = {
    Number: "A123456",
    Surname: "Karatoprak",
    Name: "Yusuf"
  }
  // Activates knockout.js
  ko.applyBindings(student);
</script>
```

در bind کردن تگ HTML، نمی توانیم ساختار Data-Bind را مشاهده کنیم. KO، Data-Binding را از طریق تگ data-bind انجام می دهد. اما ترجیح آن بدین قرار است: KO برای تعیین مقادیر اشیاء Model، تغییرات View را کنترل می کند. خصوصیت Observable بسیار حائز اهمیت است. MVVM نیاز دارد تا هر تغییری در UI را مشاهده (Observe) کند. KO هر تغییری در View را در نظر می گیرد. درواقع زمانی که یک جعبه متن (text box) را تغییر می دهید، داده های ViewModel مربوطه بروزرسانی می شوند.

ViewModelتان را طوری تغییر دهید که خواص Name و Surname بوسیله ko.Observable قابل مشاهده (observable) باشند:

```
<script type="text/javascript">
  var student = {
    Number: ko.observable("A123456"),
    Surname: ko.observable("Karatoprak"),
    Name: ko.observable("Yusuf")
  }
  // Activates knockout.js
  ko.applyBindings(student);
</script>
```

حالا برنامه را مجدداً اجرا کنید و محتویات جعبه های متن را تغییر دهید. مشاهد ه می کنید که این بار، نه تنها داده های ViewModel هنگام ویرایش بروزرسانی می شوند، بلکه کل UI وابسته هم به موازات آن تغییر می کند.



استفاده ساده Knockout در MVC

برنامه های واقعی باید بوسیله یک پایگاه داده تغذیه شوند. بنابراین Model داده های ذخیره شده برنامه شما خواهد بود که با تکنولوژی server-side قابل پیاده سازی است. View علاقه مند به درخواست های UI است. ViewModel شامل یکسری اشیاء است تا هر درخواستی از View را کنترل و مدیریت کند. برنامه نویس باید بتواند چگونگی تولید یک ViewModel را نه تنها برای WPF بلکه برای صفحات HTML داده محور نیز، تصور کرده و حدس بزند. داده محور (data-Driven) بدین معناست که در پیاده سازی ViewModel از JavaScript استفاده شده است.

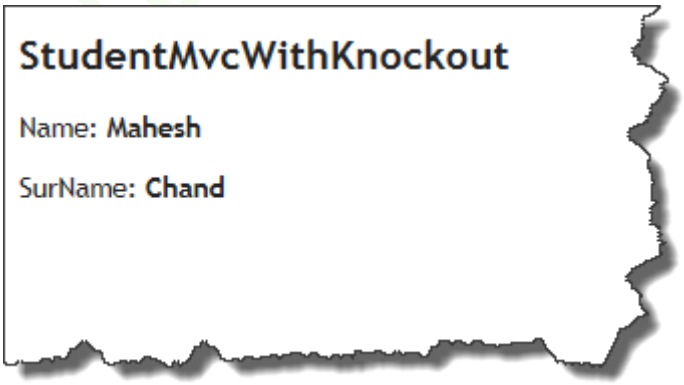
گاهی، نیازی نیست ViewModel را با JavaScript پیاده سازی کنیم. می توانیم server-side Model را به View متصل (bind) کنیم. با استفاده از "@Html.Raw(Json.Encode(Model));" می توانیم server-side Model را به View متصل (bind) کنیم.

```
Model:public class Student
{
public string Number { get; set; }
public string Name { get; set; }
public string Surname { get; set; }
}
Controller:

 [HttpGet]
    publicActionResult StudentMvcWithKnockout()
    {
      Student student =new Student();
```

```
            student.Number = "B123456";
            student.Name = "Mahesh";
            student.Surname = "Chand";
            return View(student);
        }
    View:
    @using System.Web.Script.Serialization;
    @model MvcAppWithJquery.Models.Student
    @{
      ViewBag.Title ="StudentMvcWithKnockout";
      Layout = "~/Views/Shared/_Layout.cshtml";
    }
     <h2>StudentMvcWithKnockout</h2>
       <scriptsrc="../../Scripts/knockout-2.1.0.js"type="text/javascript"></script>
        <scriptsrc="../../Scripts/knockout.mapping-latest.js"type="text/javascript"></script>
        <p>Name:<strongdata-bind="text: Name"></strong></p>
    <p>SurName:<strongdata-bind="text: Surname"></strong></p>
    <scripttype="text/javascript">
       $(function()
      {
        var model =@Html.Raw(Json.Encode(Model));
            ko.applyBindings(model);
      });
       </script>
```

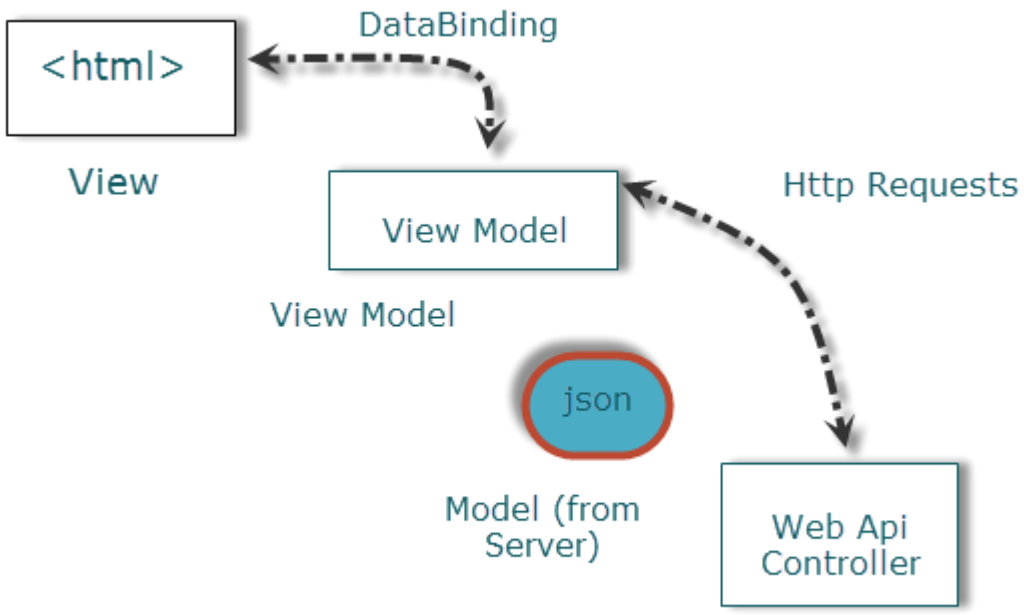
نتیجه:



با فراخوانی ko.mapping در view، می توانیم به داده های JSON دسترسی داشته باشیم. اما ما باید بوسیله "return Json(StudentList,JsonRequestBehavior.AllowGet);" داده های JSON را از Controller ارسال کند. بنابراین به یکسری مجموعه ها (collections) در ViewModel نیاز خواهیم داشت. به سادگی (ko.applyBindings(viewModel را فراخوانی می کنیم.

```
$(document).ready(function () { ko.applyBindings(viewModel); });
```

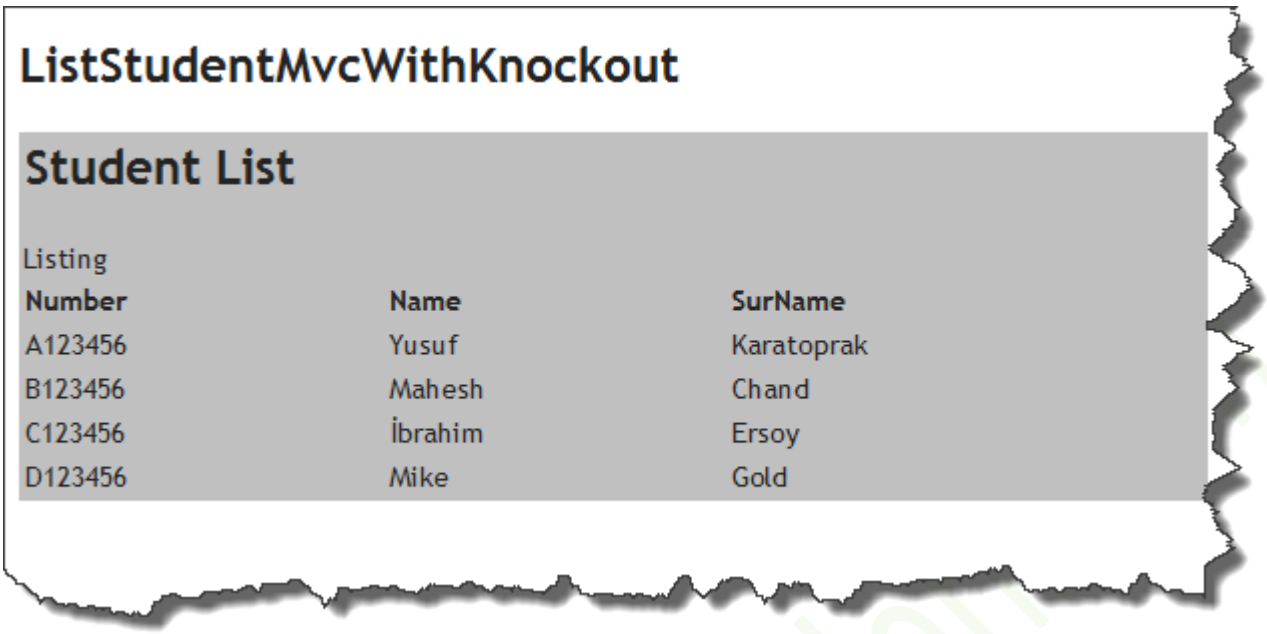
متدهای $.ajax و $.getJSON داده های JSON را دریافت می کنند. شما می توانید هر دو متد را در کد ببینید.



کد:

```
Controller
public JsonResult GetStudents()
    {
       List<Student> StudentList = newList<Student>(){new Student(){ Number="A123456", Name="Yusuf",
Surname="Karatoprak"},
                                        new Student(){ Number="B123456", Name="Mahesh", Surname="Chand"},
                                         new Student(){ Number="C123456", Name="Ä°brahim", Surname="Ersoy"},
                                           new Student(){ Number="D123456", Name="Mike", Surname="Gold"}};
return Json(StudentList,JsonRequestBehavior.AllowGet);
}
View
<tbodydata-bind="foreach: Students">
<trstyle="border-bottom: 1px solid #000000;">
          <td>
            <spandata-bind="text: Number"></span>
          </td>
          <td>
            <spandata-bind="text: Name"></span>
          </td>
           <td>
            <spandata-bind="text: Surname"></span>
          </td>
        </tr>
      </tbody>
  </table>
</div>
</form>
<scripttype="text/javascript">
  var AppViewModel =function () {
    var self =this;
    self.Students = ko.mapping.fromJS([]);
$.getJSON('/Student/GetStudents/',function (data) {
       ko.mapping.fromJS(data, {}, self.Students);
    });
  }
  $(document).ready(function () {
    var viewModel =new AppViewModel();
      ko.applyBindings(viewModel);
});
</script>
```

نتیجه:



بازبینی کد

Self.Students=ko.mapping.fromJS([]) بسیار مهم است، چون فهم اینکه از "چه چیزی نگاشت می کنیم؟" لازم به نظر می رسد. نگاشت بوسیله JS کنترل می شود. با فراخوانی '/Student/GetStudents' می توانیم ViewModel را تغذیه کنیم.

برای دریافت داده های JSON باید از $.getJSON و $.ajax استفاده کنیم.

با استفاده از ko.mapping.fromJS(data,{},self.Students); مان ViewModel، به نام self.Students را با داده هایی که به فرمت JSON هستند، پر می کنیم.

خلاصه

با بهره گیری از KO، پیاده سازی ساده تر می شود و قابلیت نگهداری بهبود می یابد. تغییرات Model توسط ViewModel و بخش های بروزرسانی شده UI قابل مشاهده هستند. KO براحتی UI را به Data Model متصل می کند. در KO می توانید روال های data را شارژ کنید تا سایر رخدادهای js (نظیر Click، Mouseover و Grid) توسط jQuery توسعه یابند. KO، کتابخانه JavaScript خالص است که با هر

تکنولوژی server و client-side ای کار می کند. با بهره گیری از KO، میتوان از MVVM در تکنولوژی MVC استفاده کرد. KO یک مکمل و یک روش سطح بالا برای ارتباط data model با UI است.