

پیش از اینکه آموزش AngularJS را شروع کنیم بهتر است با مفهوم برنامه‌های تک صفحه ای وب و یا Single Page Web Applications آشنا شویم؛ چرا که AngularJS برای توسعه هر چه ساده‌تر و قوی‌تر این گونه برنامه‌ها متولد شده است.

Single Page Application

برای درک چگونگی کارکرد این برنامه ها، مثالی را میزنیم که هر روزه با آن سرو کار دارید، یکی از نمونه‌های کامل و قدرتمند برنامه‌های Single Page Application و یا به اختصار SPA، سرویس پست الکترونیکی Google و یا همان Gmail است. اجازه بدهید تا ویژگی‌های SPA را با بررسی Gmail انجام دهیم، تا به درک روشنی از آن برسید:

Reload نشدن صفحات

در Gmail هیچ گاه صفحه Reload و یا اصطلاحا بارگیری مجدد نمی‌شود. برای مثال، وقتی شما لیست ایمیل‌های خود را مشاهده میکنید و سپس بر روی یکی از آن‌ها کلیک میکنید، بدون اینکه به صفحه ای دیگر هدایت شوید؛ ایمیل مورد نظر را میبینید. در حقیقت تمامی اطلاعات در همان صفحه نمایش داده می‌شوند و بر عکس وب سایت‌های معمول است که از صفحه ای به صفحه‌ی دیگر هدایت میشوند، در یک صفحه تمام کارهای مورد نیاز خود را انجام می‌دهید و احتیاجی به بارگیری مجدد صفحات نیست. با توجه به این صحبت‌ها برای توسعه دهنده‌های وب آشکار است که تکنیک AJAX، نقشی اساسی در این گونه برنامه‌ها دارد، چون کله‌ی عملیات ارتباط با سرور در پشت زمینه انجام می‌شوند.

تغییر URL در نوار آدرس مرورگر

وقتی شما بر روی یک ایمیل کلیک می‌کنید و آن ایمیل را بدون Reload شدن مجدد صفحه مشاهده می‌کنید، آدرس صفحه در مرورگر نیز تغییر می‌کند. خب مزیت این ویژگی چیست؟ مزیت این ویژگی در این است که هر ایمیل شما دارای یک آدرس منحصر به فرد است و به شما امکان Bookmark کردن آن لینک، باز کردن آن در یک Tab جدید و یا حتی ارسال آن به دوستان خود را دارید. حتی اگر این مطلب را جدا از Gmail در نظر بگیریم، به موتورهای جست و جو کمک می‌کند، تا هر صفحه را جداگانه Index کنند؛ جدا از اینکه وبسایت ما SPA است. همچنین این کار یک مزیت مهم دیگر نیز دارد؛ و آن کار کردن کلیدهای back و forward مرورگر، برای بازگشت به صفحات پیمایش شده قبلی است.

شاید قبل از بیان این ویژگی با خود گفته باشید که پیاده سازی Reload نشدن صفحات با AJAX آن چنان کار پیچیده ای نیست. بله درست است، اما آیا شما قبل از این راه حلی برای تغییر URL اندیشیده بودید؟ مطمئنا شما هم صفحات وب زیادی را دیده اید که همه‌ی صفحات آن دارای یک URL در نوار آدرس مرورگر هستند و هیچگاه تغییر نمی‌کنند و با باز کردن یک لینک در یک Tab جدید، باز همان صفحه‌ی تکراری را مشاهده می‌کنند! و یا بدتر از همه که دکمه‌ی back مرورگر غیر عادی عمل می‌کند. بله، اینها تنها تعدادی از صدها مشکلات رایج سیستم‌های نوشته شده ای است که سعی کردند همه‌ی کارها در یک صفحه انجام شود.

Cache شدن اطلاعات دریافتی

شاید خیلی‌ها ویژگی‌های فوق را برای یک SPA کافی بدانند، اما تعدادی هم مانند نگارنده وجود یک کمبود را حس می‌کنند و آن کش شدن اطلاعات دریافتی در مرورگر است. Gmail این امکان را به خوبی پیاده سازی کرده است. لیست ایمیل‌های دریافتی در بار اول از سرور دریافت می‌شود، سپس شما بر روی یک ایمیل کلیک و آن را مشاهده می‌کنید. حال به لیست ایمیل‌های دریافتی بازگردید، آیا رفت و برگشتی به سرور انجام می‌شود؟ مسلما خیر. حتی اگر دوباره بر روی آن ایمیل مشاهده شده، کلیک کنید، بدون رفت و برگشتی به سرور آن ایمیل را مشاهده می‌کنید.

کش شدن اطلاعات سبب می‌شود که بار سرور خیلی کاهش یابد و رفت و آمدهای بیهوده صورت نگیرد. کش شدن داده‌ها یک مزیت دیگر نیز دارد و آن تبدیل برنامه‌های معمول وب stateless به برنامه‌های شبه دسکتاپ state full است. تکنیک AJAX در پیاده سازی امکانات فوق نقشی اساسی را بازی می‌کند. کمی به عقب برمیگردیم یعنی زمانی که AJAX برای اولین بار مطرح شد و هدف اصلی به وجود آمدن آن پیاده سازی برنامه‌های وب به شکل دسکتاپ بود و این کار از طریق انجام تمامی ارتباطات سرور با XMLHttpRequest امکان پذیر می‌شد. شاید آن زمان با توجه به محدودیت تکنولوژی‌ها موجود این کار به صورت

تمام و کمال امکان پذیر نبود، اما امروزه به بهترین شکل ممکن قابل پیاده سازی است.

شاید اکنون این سوال پیش بیاید که چرا باید وبسایت خود را به شکل SPA طراحی کنیم؟

برای پاسخ دادن به این سوال باید گفت که سیستم‌های وب امروزی به دو دسته‌ی زیر تقسیم می‌شوند:

- Web Documents و یا همان وب سایت‌های معمول

- Web Applications و یا همان Single Page Web Applications

اگر هدف شما طراحی یک وب سایت معمول است که هدف آن، نمایش یک سری اطلاعات است و به قولی دارای محتواست، مطمئناً پیاده سازی این سیستم به صورت SPA کاری بیهوده به نظر می‌آید؛ ولی اگر هدفتان نوشتن سیستم‌هایی مثل Gmail، Google Maps، Azure، Facebook و ... است، پیاده سازی آن‌ها به صورت وب سایت‌های معمولی، غیر معقول به نظر می‌آید. حتی بخش‌های مدیریتی یک وبسایت هم می‌تواند به خوبی توسط SPA پیاده سازی شود، چرا که واقعا برای مدیریت اطلاعات یک وب سایت احتیاجی نیست، که از این صفحه به آن صفحه جا به جا شد.

معرفی کتابخانه‌ی AngularJS

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.

قبل از اینکه به بررسی ویژگی‌های Angular بپردازم، بهتر است ابتدا مطلبی درباره‌ی به کارگیری Angular از Brad Green که کارمند گوگل است، بیان کنم.

در سال 2009 تیمی در گوگل مشغول انجام پروژه ای به نام Google Feedback بودند. آن‌ها سعی داشتند تا در طی چند ماه، به سرعت کدهای خوب و تست پذیر بنویسند. پس از 6 ماه کدنویسی، نتیجه‌ی کار 17000 خط کد شد. در آن موقع یکی از اعضای تیم به نام Misko Hevery، ادعا کرد که می‌تواند کل این پروژه را در دو هفته به کمک کتابخانه‌ی متن بازی که در اوقات فراغت توسعه داده است، بازنویسی کند. Misko نتوانست در دو هفته این کار را انجام دهد. اما پس از سه هفته همه‌ی اعضای تیم را شگفت زده کرد. نتیجه‌ی کار تنها 1500 خط بود! همین باعث شد که ما بفهمیم که، Misko بر روی چیزی کاری میکند که ارزش دنبال کردن دارد.

پس از آن قضیه Misko و Brad بر روی Angular کار کردند و اکنون هم Angular توسط تیمی در گوگل و هزاران توسعه دهنده‌ی متن باز حرفه ای در سرتاسر جهان، در حال توسعه است.

فکر کنم همین داستان ذکر شده، قدرت فوق العاده زیاد این فریم ورک را برای همگان آشکار سازد.

ویژگی‌های AngularJS:

- قالب‌های سمت کاربر (Client Side Templates): انگولار دارای یک template engine قدرتمند برای تعریف قالب است.

- پیروی از الگوی طراحی MVC: انگولار، الگوی طراحی MVC را برای توسعه پیشنهاد می‌دهد و امکانات زیادی برای توسعه هر چه راحت‌تر با این الگو فراهم کرده است.

- Data Binding: امکان تعریف انقیاد داده دوطرفه (Two-Way Data Binding) در این فریم ورک به راحتی هرچه تمام، امکان پذیر است.

- Dependency Injection: این فریم ورک برای دریافت وابستگی‌های تعریف شده، دارای یک سیستم تزریق وابستگی توکار است.

- تعریف Service‌های سفارشی: در این فریم ورک امکان تعریف سرویس‌های دلخواه به صورت ماژول وجود دارد. این ماژول‌های مجزا را به کمک سیستم تزریق وابستگی توکار Angular، به راحتی در هر جای برنامه می‌توان تزریق کرد.

- تعریف Directive‌های سفارشی: یکی از جذاب‌ترین و قدرتمندترین امکانات این فریم ورک، تعریف Directive‌های سفارشی

است. Directive ها، امکان توسعه HTML را فراهم کرده اند. توسعه‌ی HTML اکنون در قالب Web Components ها فراهم شده است، اما هنوز هم خیلی از مرورگرهای جدید نیز از آن پشتیبانی نمی‌کنند.

- **فرمت کردن اطلاعات با استفاده از فیلترهای سفارشی:** با استفاده از فیلترها می‌توانید چگونگی الحاق شدن اطلاعات را برای نمایش به کاربر تعیین کنید ؛ انگولار همراه با فیلترهای گوناگون مختلفی عرضه میشود که میتوان برایه مثال به فیلتر ، currency ، date ،uppercase کردن رشته‌ها و اشاره کرد همچنین شما محدود به فیلترهای تعریف شده در انگولار نیستید و آزادید که فیلترهای سفارشی خودتان را نیز تعریف کنید.

- **سیستم Routing:** دارا بودن سیستم Routing قدرتمند، توسعه SPA ها را بسیار ساده کرده است.

- **سیستم اعتبار سنجی:** Angular دارای سیستم اعتبار سنجی توکار قدرتمند برای بررسی داده‌های ورودی است.

- **سرویس تو کار برای ارتباط با سرور:** Angular دارای سرویس پیش فرض ارتباط با سرور به صورت AJAX است.

- **تست پذیری:** Angular دارای بستری آماده برای تست کردن برنامه‌های نوشته شده است و از Unit Tests و Integrated End-to-End Test هم پشتیبانی می‌کند.

- **جامعه‌ی متن باز بسیار قوی**

این‌ها فقط یک مرور کلی بر توانایی‌های این فریم ورک بود و در ادامه هر کدام از این ویژگی را به صورت دقیق بررسی خواهیم کرد.

در مقاله‌ی بعدی، به چگونگی نصب AngularJS خواهیم پرداخت. سپس، اولین کد خود را با استفاده از آن خواهیم نوشت و مطالب Client Side Templates و MVC را دقیق‌تر بررسی خواهیم کرد.

نظرات خوانندگان

نویسنده:

دادخواه

تاریخ:

۱۹:۲۷ ۱۳۹۲/۰۶/۰۶

سلام

آیا استفاده از این فریمورک باعث این می‌شود که دیگر نیازی به JQuery نباشد؟ و یا باز کمبود هایی دارد که به استفاده از JQuery نیاز باشد؟
تشکر

نویسنده:

سالار

تاریخ:

۲۳:۱۹ ۱۳۹۲/۰۶/۰۶

به شدت دنبال آموزش angularjs بودم. سپاس و چند سوال.
- آیا با استفاده از angularjs برای یک SPA دیگر نیازی به asp.net mvc خواهد بود؟
- کامپایلر و ویژوال استودیو به خوبی برای razor کار میکند و بسیار intellisense قوی دارد و امکانات Strongly typed نیز دارد.
آیا این امکانات برای angular نیز موجود است تا خطایابی راحت‌تر شود؟
- من یک اپلیکیشن دارم که کاربر هر بار یکی از آیتم‌های منو را انتخاب میکند و برای آن یک ویو بصورت partialview درون یک تب لود میشود. هر ویو، فایل جاوا اسکریپت مخصوص به خود دارد. کاربر می‌تواند چندین ویوی مجزا را درون تب‌ها باز کند و بین آنها جابه‌جا شود. اما مدیریت دستی آنها بسیار سخت شده است. آیا انگولار امکان نمایش چند ویو را بطور همزمان و جابه‌جا شدن بین آنها را میدهد. (بدون بروز تداخل بین فایل‌های جاوا اسکریپت مربوط به هر ویو).
با تشکر.

نویسنده:

مهدی سعیدی فر

تاریخ:

۸:۰۰ ۱۳۹۲/۰۶/۰۷

سلام؛

کتابخانه‌ی JQuery عموماً برای دو کار استفاده می‌شود:

1- دستکاری یا پردازش DOM

2- ارتباط با سرور به صورت AJAX

انگولار سرویسی برای ارتباط AJAX دارد که شما را کاملاً از JQuery بی‌نیاز می‌کند.
برای پردازش DOM و الصاق رویداد به عناصر صفحه، انگولار دارای API‌های خیلی خوبی است که می‌تواند تا حدی جایگزین JQuery باشد. اما فراموش نکنیم که JQuery علاوه بر پردازش DOM دارای کلی API متنوع است که انگولار به پای آن نمی‌رسد. دلیل هم مشخص است؛ چرا که هر دو کتابخانه برای کاری متفاوت نوشته شده‌اند و مقایسه آن‌ها اصلاً کار درستی نیست. همچنین به این نکته هم توجه داشته باشید که برای JQuery هزاران افزونه‌ی مختلف نوشته‌اند که اجرای آن‌ها، وابسته به JQuery است. پس هیچ کتابخانه‌ای در این زمینه موازی کاری نمی‌کند، چون توانایی رقابت ندارد. (البته تعدادی را هم با کاملاً انگولار بازنویسی کرده‌اند). انگولار در مورد دستکاری DOM هم یک مورد را شما گوشزد می‌کند: اگر می‌خواهید DOM را دستکاری کنید، این کار را باید از طریق Directive‌ها انجام دهید. منظور این است کدهای مورد نظرتان را با هر کتابخانه‌ای که می‌خواهید بنویسید، ولی آن‌ها را درون یک Directive سفارشی بنویسید و روی هوا کد ننویسید تا قابل تست کردن باشد.
در ادامه کاملاً متوجه منظورم خواهید شد.

نویسنده:

مهدی سعیدی فر

تاریخ:

۸:۳۱ ۱۳۹۲/۰۶/۰۷

- AngularJS یک فریم ورک سمت کلاینتی به زبان جاوا اسکریپت است و ASP.NET MVC یک فریم ورک سمت سرور و این دو نمی‌توانند جای یکدیگر را پر کنند و ربطی به هم ندارند.

- در اینجا View‌های شما یک HTML ساده است و مفاهیم Strongly Typed View هیچ معنی خاصی ندارد. Model در انگولار یک

کلاس جاوا اسکریپتی ساده (POJO) است. view هم یک فایل HTML که قالب کار را مشخص می‌کند و Controller هم باز یک کلاس ساده‌ی جاوا اسکریپت است.

- اتفاقاً یک بار همچنین چیزی را از من خواستند (بدون انگولار) و من انجامش دادم. البته این طراحی اشتباه است. آیا شما ده بخش را در چند TAB مختلف باز می‌کنید، آدرس منحصر به فردی هم در نوار آدرس مرورگر برای آن‌ها در نظر گرفته اید؟ این کار در آن زمان که همه‌ی برنامه با یک URL کار می‌کرد خیلی عالی بود، ولی اکنون که شما می‌توانید برای View یک آدرس منحصر به فرد داشته باشید و آن‌ها را Tab خود مرورگر مدیریت کنید، احتیاجی به آن کارها نیست. به هر حال، در انگولار این جمله بی معنی است که: بین فایل‌های جاوا اسکریپت تداخل ایجاد شود! در انگولار هر View دارای Controller متناظر خود است. اعمال تجاری مشترک هم در سرویس‌ها تعریف می‌شوند و در کنترلرها تزییق می‌شوند. پس کد جاوا اسکریپتی روی هوا نوشته نمی‌شود (حتی یک خط)

صبر کنید تا کمی پیش برویم. این فقط مقدمه بود...

نویسنده: سالار
تاریخ: ۱۳۹۲/۰۶/۰۷ ۹:۵۲

سپاس فراوان.
آیا در یک وب اپلیکیشن و نه وب سایت عمومی، واقعاً نیازی به داشتن یک url مجزا برای هر ویو داریم. مگر هدف angular ایجاد برنامه‌هایی شبیه برنامه‌های دسکتاپ نیست. در برنامه‌های دسکتاپ که هر ویو آدرس مجزایی ندارد. با تشکر.

نویسنده: دادخواه
تاریخ: ۱۳۹۲/۰۶/۰۷ ۱۰:۲۱

سلام
تشکر از جواب خوبتان
من متوجه شدم که دانسته‌های شما نسبت به فریمورک‌های جاوا اسکریپت بسیار خوب است. پس اگر لطف کنید درباره چند تا از فریم ورک‌های دیگر بخصوص Backbone اطلاعاتی بدهید که هر کتابخانه در اصل برای چه کاری ساخته شده است. منظورم این است که به مقایسه بین فریم ورک‌های معروف که بیشتر برای چه کاری تهیه شده اند. اگر چنین مقایسه‌ای به صورت فارسی هست معرفی کنید.
تشکر

نویسنده: مهدی سعیدی فر
تاریخ: ۱۳۹۲/۰۶/۰۷ ۱۰:۲۶

اغلب کارمندان یک سازمان یک عادت عمومی دارند، بر روی لینکشان کلیک راست می‌کنند و گزینه‌ی open in a new tab را می‌زنند و یا بدتر از کلیدهای back و forward مرورگر برای عقب جلو کردن صفحات پیمایش شده استفاده می‌کنند. حالا هر چقدر که توی گوششان بخوانید، که فلان لینک را باید روش فقط کلیک کنی، باز هم به خرجشان نمی‌رود. خیلی از کارمندان برای مثال لینک درج صورت حساب جدید را bookmark کرده اند تا به سرعت به آن دسترسی داشته باشند. همچنین autocomplete مرورگر هم در یافتن صفحات پیمایش شده به آن‌ها خیلی کمک می‌کند. یادمه همین مشکل را توی یک برنامه‌ی نوشته شده با سیلورلایت دیدم و مشتری قبول نمی‌کرد که نمی‌تواند در یک tab جدید صفحه باز کند و ...
برنامه‌های شبیه دسکتاپ یعنی اینکه رفرش شدن صفحات را برای تغییر view از بین ببرد. هوز هم ماهیت وب است و باید همیشه این را در نظر داشت، تا بهترین تجربه‌ی رابط کاربری را فراهم کرد. نباید گفت که برنامه‌ی من فلان جور طراحی شده؛ همینی هست که هست، باید با رابط کاربری عموم وب اینترفیس‌ها هماهنگ باشد.

نویسنده: _mehdi
تاریخ: ۱۳۹۲/۰۶/۲۱ ۱۲:۳۸

سلام

چطوری میشه به فایل‌های راهنما و آموزش‌های موجود در فایل دریافتی از angular js دسترسی داشت . (پسوند فایل‌ها ngdoc)

نویسنده: محسن خان

تاریخ: ۱۳:۵۶ ۱۳۹۲/۰۶/۲۱

These files are parsed by our docs parser (nodejs script), source can be found here:
<https://github.com/angular/angular.js/tree/master/docs/src>
 It is combined together with docs parsed from the source and result is html, served at
docs.angularjs.org

[ماخذ](#)

نویسنده: _mehdi

تاریخ: ۲۰:۳۵ ۱۳۹۲/۰۶/۲۱

سلام

ممنون بابت پاسختون ولی من متوجه نشدم که باید چطوری این فایل‌ها را به فایل‌های html تبدیل کرد . من باید nodejs را نصب کنم و با این فایل‌ها که در بالا ذکر کردید (البته کدومشون) و دادن مسیر پوشه راهنما فایل‌ها را به html تبدیل کنم لطفاً دقیق توضیح بدید اگه براتون امکان داره.

نویسنده: آریو

تاریخ: ۸:۳۸ ۱۳۹۲/۰۸/۱۵

سلام. با تشکر از مطالب بسیار خوبتون یه سوال داشتم.

من خیلی علاقه دارم روی الگوی MVVM کار کنم و تا اینجا به دو انتخاب رسیدم یکی KnockoutJS و دیگری AngularJS . سوالی که دارم اینه که با توجه به اینکه من روی ASP.NET MVC کار میکنم کدوم کتابخانه برای این فریمورک مناسبتره. درسته که این کتابخانه‌ها جاوااسکریپت هستند و تکنولوژی سمت سرور مطرح نیست. اما با توجه به اینکه میخوام روی MVVM وقت بزارم به نظرتون با در نظر گرفتن امکانات ، منابع آموزشی ، میزان استفاده در توسعه دهندگان ایرانی و در نهایت قدرت عملکرد روی کدوم کار کنم؟

آیا اینها دقیقا یک کار اما با امکانات متفاوت انجام میدن یا کاربردهای متفاوتی دارند؟
 باتشکر

نویسنده: محسن خان

تاریخ: ۸:۵۲ ۱۳۹۲/۰۸/۱۵

[مقایسه 8 قسمتی AngularJS vs Knockout](#)

نویسنده: گل رز

تاریخ: ۱۰:۱۹ ۱۳۹۲/۰۸/۱۵

منابع آموزشی برای هر کدام وجود دارد . اگه قرار به یادگیری باشه از کجا و از چه منبعی باید شروع کنیم ؟ منبع خاصی مد نظرتون هست ؟

نویسنده: وحید نصیری

تاریخ: ۱۰:۲۴ ۱۳۹۲/۰۸/۱۵

- در سایت جاری: [AngularJS Knockout](#)

- در سایت‌های دیگر: [AngularJS Knockout SPA](#)

نویسنده: مهدی
تاریخ: ۱۴:۷ ۱۳۹۳/۰۸/۰۴

آیا با استفاده از این فریمورک دیگر نیازی به استفاده از knockout نداریم.

نویسنده: شاهین کیاست
تاریخ: ۱۶:۴۳ ۱۳۹۳/۰۸/۰۴

knockoutjs کتاب خانه ای برای انجام DataBinding در وب هست، یعنی برای مقید سازی اشیاء جاوا اسکریپت به کنترل های html.

AngularJS یک Toolset برای درست کردن framework و برنامه های تک صفحه ای می باشد، angular قابلیت مقید سازی داده ها را هم فراهم می کند.
لطفا برای اطلاعات بیشتر [این مطلب](#) را مطالعه کنید.

نویسنده: فخاری
تاریخ: ۰:۲۷ ۱۳۹۳/۱۰/۱۹

با توجه به این مطلب:

" سیستم های وب امروزی به دو دسته ی زیر تقسیم می شوند:

- Web Documents و یا همان وب سایت های معمول

- Web Applications و یا همان Single Page Web Applications "

نرم افزارهای تحت وب را با چه قالبی درست می کنند؟

ASP.NET MVC

یا

(Single Page application (SPA

یعنی آیا میشه که ما نرم افزار تحت وب را با ASP.NET MVC درست کنیم و از Angularjs استفاده کنیم یا اینکه نه بهتره حتما همون اول کار قالب پروژه را از نوع SPA انتخاب کنیم ؟

عنوان:	AngularJS #2
نویسنده:	مهدی سعیدی فر
تاریخ:	۱۶:۵۰ ۱۳۹۲/۰۶/۰۹
آدرس:	www.dotnettips.info
گروه‌ها:	AngularJS

بهتر است قبل از این که به ادامه‌ی آموزش بپردازم، دو نکته را متذکر شوم:

- 1) روند آموزشی این فریمورک از کل به جز است؛ به این معنا که ابتدا تمامی قابلیت‌های اصلی فریمورک را به صورت کلی و بدون وارد شدن به جزئیات بیان می‌کنم و پس از آن، جزئیات را در قالب مثال‌هایی واقعی بیان خواهم کرد.
- 2) IDE مورد استفاده بنده Visual Studio 2012 است. همچنین از ابتدا پروژه را با ASP.NET MVC شروع می‌کنم. شاید بگویید که می‌شود Angular را بدون درگیر شدن با مباحث ASP.NET MVC بیان کرد؛ اما پاسخ من این است که این مثال‌ها باید قابل پیاده‌سازی در نرم‌افزارهای واقعی باشند و یکی از بسترهای مورد علاقه‌ی من ASP.NET MVC است. اگرچه باز هم تاکید می‌کنم که کلیه‌ی مباحث ذکرشده، برای کلیه‌ی زبان‌های سمت سرور دیگر هم قابل استفاده است و هدف من در اینجا بیان یک سری چالش‌ها در ASP.NET MVC است.

نحوه‌ی دریافت AngularJS

- 1) NuGet Package Manager
- 2) دریافت از وبسایت angularjs.org

دریافت از طریق NuGet Package Manager

روش ارجح افزودن کتابخانه‌های جانبی در یک پروژه‌ی واقعی، استفاده از NuGet Package Manager است. دلیل آن هم بارها بیان شده است از جمله: باخبر شدن از آخرین به‌روزرسانی کتابخانه‌ها، دریافت وابستگی‌های کتابخانه‌ی مورد نظر و نبودن محدودیت تحریم برای دریافت فایل‌ها است.

روش کار هم بسیار ساده است، کافی است که بر روی پروژه کلیک راست کرده و گزینه‌ی Manage NuGet Packages را انتخاب کنید و با جست جو `angularjs` نسبت به نصب آن اقدام نمایید.

اگر هم با ابزارهای گرافیکی رابطه‌ی خوبی ندارید، می‌توانید از Package Manager Console فراهم‌شده توسط NuGet استفاده کنید. کافی است در کنسول پاورشل آن عبارت زیر را تایپ کنید:

```
Install-Package angularjs
```

پس از نصب `angularjs`، شاهد تغییراتی در پوشه‌ی `Scripts` پروژه‌ی خود خواهید بود. تعداد زیادی فایل جاوا اسکریپت که با عبارت `angular` شروع شده‌اند، به این پوشه اضافه شده است. در حال حاضر ما تنها به فایل `angular.js` نیاز داریم و احتیاجی به فایل‌های دیگر نیست.

همچنین یک پوشه به نام `i18n` نیز اضافه شده است که برای مباحث `Globalization` و `Internationalization` به کار گرفته می‌شود.

دریافت از سایت angularjs.org

برای دریافت Angular از وب سایت رسمی‌اش، به angularjs.org مراجعه کنید؛ اما گویا به دلیل تحریم‌ها این سایت برای IP ایران مسدود شده است (البته افرادی نیز بدون مشکل به آن دسترسی دارند). دکمه‌ی `Download` را فشار داده و در نهایت کلیک دریافت را بزنید. اگر نسخه‌ی کامل آن را دریافت کنید، لیستی از مستندات AngularJS را نیز در فایل دریافتی، خواهید داشت. در هر صورت این روش برای استفاده از `angular` در یک پروژه‌ی واقعی توصیه نمی‌شود.

پس به عنوان یک `best practice`، همیشه کتابخانه‌های جانبی را با NuGet دریافت و نصب کنید. رفع موانع تحریم‌ها، یکی از مزایای مهم آن است.

پس از دریافت `angular`، نوشتن برنامه‌ی معروف `Hello, World` به وسیله‌ی آن، می‌تواند بهترین شروع باشد؛ اما اگر اجازه بدهید، نوشتن این برنامه را در قالب توضیح قالب‌های سمت کلاینت انجام دهیم.

قالب‌های سمت کلاینت (Client Side Templates)

در برنامه‌های وب چند صفحه‌ای و یا اکثر وب سایت‌های معمول، داده‌ها و کدهای HTML، در سمت سرور اصطلاحاً سرهم و مونتاژ

شده و خروجی نهایی که HTML خام است به مرورگر کاربر ارسال می‌شود. با یک مثال بیشتر توضیح می‌دهم: در ASP.NET MVC معمولاً از لحظه‌ای که کاربر صفحه‌ای را درخواست می‌کند تا زمانی که پاسخ خود را در قالب HTML می‌بیند، این فرآیند طی می‌شود: ابتدا درخواست به Controller هدایت می‌شود و سپس اطلاعات مورد نیاز از پایگاه داده خوانده شده و در قالب یک Model به View که یک فایل HTML ساده است، منتقل می‌شود. سپس به کمک موتور نمایشی Razor، داده‌ها در جای مناسب خود قرار می‌گیرند و در نهایت، خروجی که HTML خام است به مرورگر کلاینت درخواست‌کننده ارسال می‌شود تا در مرورگر خود نتیجه را مشاهده نماید. روال کار نیز در اکثر SPAهای معمول و یا اصطلاحاً برنامه‌های AJAX، با کمی تغییر به همین شکل است.

اما در Angular داستان به شکل دیگری اتفاق می‌افتد؛ Angular قالب HTML و داده‌ها را به صورت جداگانه از سرور دریافت می‌کند و در مرورگر کاربر آن‌ها را سرهم و مونتاژ می‌کند. بدیهی است که در اینجا قالب، یک فایل HTML ساده و داده‌ها می‌تواند به فرم JSON باشد. در نتیجه کار سرور دیگر فراهم کردن قالب و داده‌ها برای کلاینت است و بقیه‌ی ماجرا در سمت کلاینت رخ می‌دهد. خیلی خوب، مزیت این کار نسبت به روش‌های معمول چیست؟ اگر اجازه بدهید این را با یک مثال شرح دهیم:

در بسیاری از سایت‌ها، ویژگی‌ای به نام اسکرول نامحدود وجود دارد. در همین سایت نیز دکمه‌ای با عنوان بیشتر در انتهای لیستی از مطالب، برای مشاهده‌ی ادامه‌ی لیست قرار گرفته است. سعی کنید پس از فشردن دکمه‌ی بیشتر، داده‌های دریافتی از سرور را مشاهده کنید. پس از انجام این کار مشاهده خواهید کرد که پاسخ سرور HTML خام است. اگر تعداد 10 پست از سرور درخواست شود، 10 بار محتوای HTML تکراری نیز دریافت خواهد شد؛ در صورتی که ساختار HTML یک پست هم کفایت می‌کرد و تنها داده‌ها در آن 10 پست متفاوتند؛ چرا که قالب کار مشخص است و فقط به ازای هر پست باید آن داده‌ها در جای مناسب خود قرار داد.

دیدگاه‌های یک پست هم به خوبی با Angular قابل پیاده سازی است. قالب HTML یک دیدگاه را برای angular تعریف کرده و داده‌های مناسب که احتمالاً JSON خام است از سرور دریافت شود. نتیجه‌ی این کار هم صرفه جوی در پهنای باند مصرفی و افزایش فوق العاده‌ی سرعت است، همچنین در صورت نیاز می‌توان داده‌ها و قالب‌ها را کش کرد تا مراجعه به سرور به حداقل برسد.

چگونگی انجام این کار در AngularJS به صورت خلاصه به این صورت است که در angular یک directive به نام ng-repeat تعریف شده است که مانند یک حلقه‌ی foreach برای HTML عمل می‌کند. شما در داخل حلقه، قالب را مشخص می‌کنید و به ازای تعداد داده‌ها، آن حلقه تکرار می‌شود و بر روی داده‌ها پیمایش صورت می‌گیرد. البته این مثال‌ها فقط دو نمونه از کاربرد این ویژگی در دنیای واقعی بود و مطمئن باشید که در مقالات آینده مثال‌های زیادی از این موضوع را پیاده‌سازی خواهیم کرد.

بهتر است که دیگر خیلی وارد جزئیات نشویم و اولین برنامه‌ی خود را به کمک angularjs بنویسیم. این برنامه، همان برنامه‌ی معروف Hello, World است؛ اما در این برنامه به جای نوشتن یک Hello, World ساده در صفحه، آن را با ساختار angularjs پیاده‌سازی می‌کنیم.

در داخل ویژوال استادیو یک فایل HTML ساده ایجاد کنید و کدهای زیر را داخل آن بنویسید.

```
<!DOCTYPE html>
<html ng-app>
<head>
  <title>Sample 1</title>
</head>
<body>
  <div ng-controller="GreetingController">
    <p>{{greeting.text}}, World!</p>
  </div>

  <script src="../../Scripts/angular.js"></script>
  <script>
    function GreetingController($scope) {
      $scope.greeting = {
        text: "Hello"
      };
    }
  </script>
</body>
</html>
```

سپس فایل فوق را در مرورگر اجرا کنید. بله؛ عبارت Hello, World را مشاهده خواهید کرد. یک بار دیگر خاصیت text را در \$scope.greeting به hi تغییر بدهید و باز هم نتیجه را مشاهده کنید.

این مثال در نگاه اول خیلی ساده است، اما دنیایی از مفاهیم angular را در بر دارد. شما خواص جدیدی را برای عناصر HTML

مشاهده می‌کنید: `ng-app`, `ng-controller`، آکلوها و عبارت درون آن و متغیر `$scope` به عنوان پارامتر.

حال بیایید ویژگی‌ها و مفاهیم جالب کدهای نوشته شده را بررسی کنیم؛ چرا که فرصت برای بررسی `ng-app` و بقیه‌ی موارد نا آشنا زیاد است:

- هیچ `id` و یا `class` برای عناصر `html` در نظر گرفته نشده تا با استفاده از آنها، رویدادی را برای عناصر مورد نظر مشخص کنیم.

- وقتی در `GreetingController` مقدار `greeting.text` را مشخص کرده ایم، باز هم هیچ رویدادی را صدا نزده و یا مشخص نکرده ایم.

- `GreetingController` یک کلاس ساده‌ی جاوا اسکریپت (POJO) است و از هیچ چیزی که توسط `angular` فراهم شده باشد، ارث بری نکرده است.

- اگر به متد سازنده‌ی کلاس `GreetingController` دقت کنید، متغیر `$scope` به عنوان پارامتر تعریف شده است. نکته‌ی جالب این است که ما هیچ گاه به صورت دستی سازنده‌ی کلاس `GreetingController` را صدا نزده ایم و حتی درون سازنده هم `$scope` را ایجاد نکرده ایم؛ پس چگونه توانسته ایم خاصیتی را به آن نسبت داده و برنامه به خوبی کار کند. بهتر است برای پاسخ به این سوال خودتان دست به کار شوید؛ ابتدا نام متغیر `$scope` را به نام دلخواه دیگری تغییر دهید و سپس برنامه را اجرا کنید. بله برنامه دیگر کار نمی‌کند. دلیل آن چیست؟ همان طور که گفتیم `Angular` دارای یک سیستم تزریق وابستگی توکار است و در اینجا نیز `$scope` به عنوان وابستگی در سازنده‌ی این کلاس مشخص شده است تا نمونه‌ی مناسب آن توسط `angular` به کلاس `GreetingController` ما تزریق شود؛ اما چرا به نام آن یعنی `$scope` حساس است؟ به این دلیل که زبان جاوا اسکریپت یک زبان پویا است و نوع در آن مطرح نیست؛ `angular` مجبور است که از نام پارامترها برای تزریق وابستگی استفاده می‌کند. در مقالات آینده چگونگی عملکرد سیستم تزریق وابستگی `angular` را به تشریح بیان می‌کنم.

- همچنین همان طور که در مورد قبلی نیز به آن اشاره کردم، ما هیچ گاه خود دستی سازنده‌ی `GreetingController` را صدا نزدیم و جایی نیز نحوه‌ی صدا زدن آن را مشخص نکرده ایم.

تا همین جا فکر کنم کاملاً برای شما مشخص شده است که ساختار فریمورک `Angular` با تمامی کتابخانه‌های مشابه متفاوت است و با ساختاری کاملاً اصولی و حساب شده طرف هستیم. همچنین در مقالات آینده توجه شما را به قابلیت‌هایی بسیار قدرتمندتر جلب خواهیم کرد.

MVC، MVP، MVVM و یا MVW

در بخش اول این مقاله، الگوی طراحی پیشنهادی فریمورک `Angular` را `MVC` بیان کرده‌ام؛ اما همان طور که گفته بودم `AngularJS` از انقیاد داده دوطرفه (Two Way Data Binding) نیز به خوبی پشتیبانی می‌کند و به همین دلیل عده‌ای آن را یک `MVVM Framework` تلقی می‌کنند. حتی داستان به همین جا ختم نمی‌شود و عده‌ای آن را به چشم `MVP Framework` نیز نگاه می‌کنند. در ابتدا سایت رسمی `AngularJS` الگوی طراحی مورد استفاده را `MVC` بیان می‌نمود ولی در این چند وقت اخیر عنوانش را به `MVW Framework` تغییر داده است.

`MVW` مخفف عبارت `Model View Whatever` هست و کاملاً مفهومش مشخص است. `Model` و `View` بخش‌های مشترک تمام الگوها بودند و تنها بخش سوم مورد اختلاف توسعه دهندگان بود؛ در نتیجه انتخاب آن را بر عهده‌ی استفاده کننده قرار داده اند و تمام امکانات لازم برای پیاده‌سازی این الگوهای طراحی را فراهم کرده اند. در طی این مقالات صرف نظر از تمام الگوهای طراحی فوق، من بیشتر بر روی `MVC` تمرکز خواهم کرد.

الگوی طراحی `MVC` در سال 1970 به عنوان بخشی از زبان برنامه نویسی `Smalltalk` معرفی شد و از همان ابتدا به سرعت محبوبیت زیادی در بین محیط‌های توسعه‌ی دسکتاپی از قبیل `C++` و `Java` که رابط کاربری گرافیکی به نوعی در آن‌ها دخیل است، پیدا کرد.

تفکر `MVC` این را بیان می‌کند که باید جداسازی واضح و روشنی بین مدیریت داده‌ها (`Model`)، منطق برنامه (`Controller`) و نمایش داده‌ها به کاربر (`View`) وجود داشته باشد و در اصل هدفش جداسازی اجزای رابط کاربری به بخش‌هایی مجزا است. شاید این سوال برای شما پیش بیاید که چرا باید چنین الگویی را در برنامه‌ها پیاده کرد؟

احتمالاً تا کنون از بین برنامه‌هایی که نوشته اید، رابط کاربری بیشتر از آن‌ها را نیز خودتان مجبور شده اید طراحی کنید؛ به این دلیل که برنامه‌ی شما بدون رابط کاربری قابل اجرا شدن نبوده است. اجرای برنامه‌ی شما منوط به وجود تعدادی دکمه و `textbox`

و ... بوده است و به قولی منطق برنامه به رابط گرافیکی گره خورده بوده است. پس می‌توان گفت که پیاده‌سازی الگوی طراحی وقتی ضرورت پیدا می‌کند که رابط گرافیکی، قسمتی از برنامه‌ی شما را تشکیل دهد.

آیا با وجود زبان‌های طراحی ساده‌ای مثل HTML و XAML و ... احتیاجی است که برنامه‌نویس وقت خود را صرف طراحی رابط کاربری کند؟ مسلماً خیر، چون دیگر با این امکانات یک طراح هم از پس این کار به خوبی و یا حتی بهتر بر می‌آید. دیگر وظیفه‌ی برنامه‌نویس نوشتن کدهای مربوط به منطق برنامه است. کدهایی که بدون UI هم قابل تست شدن باشد و به راحتی بتوان برای آن‌ها آزمون‌های واحد نوشت. برنامه‌نویس باید این را در نظر بگیرد که UI وجود ندارد و حتی ممکن است هیچ گاه هم ایجاد نشود و این کدها تبدیل به یک کتابخانه شود و مورد استفاده قرار بگیرد تا در یک برنامه با رابط کاربری گرافیکی.

در MVC، روال عمومی کار به این شکل است که View داده‌ها را از Model دریافت می‌کند و به کاربر نمایش می‌دهد. وقتی که کاربر با کلیک کردن و تایپ کردن با برنامه ارتباط برقرار می‌نماید، Controller به این درخواست‌ها پاسخ می‌دهد و داده‌های موجود در Model را به روز رسانی می‌کند. در نهایت هم Model تغییرات خود را به View منعکس می‌کند تا View آن‌ها را که پیش از آن نمایش می‌داده است، تغییر دهد و View را از تغییرات رخ داده آگاه نماید.

اما در برنامه‌های Angular قضیه از چه قرار است؟ در Angular، قالب HTML یا اگر بخواهم دقیق‌تر بگویم (Document Object Model (DOM معادل View است؛ کلاس‌های جاوا اسکریپتی نقش Controller را دارند؛ و خواص اشیای جاوا اسکریپتی و یا حتی خود اشیای نقش Model را بر عهده دارند.

ساختار بخشیدن به برنامه با استفاده از MVC یک مزیت مهم دیگر نیز دارد: ساختار کار کاملاً مشخص است و هر کسی نمی‌تواند به صورت سلیقه‌ای آن را پیاده‌سازی کند. با یک مثال این موضوع را تشریح می‌کنم: اگر کسی پروژه‌ی بنده را که با ASP.NET MVC نوشتم، بررسی کند، اصلاً احساس غریبی نمی‌کند و به راحتی می‌تواند آن را توسعه دهد. دلیل این موضوع این است که ASP.NET MVC یک ساختار مشخص را به توسعه‌دهندگان اجبار کرده است و هر کسی این ساختار را رعایت کند و با آن آشنا باشد، به راحتی می‌تواند با آن کار کند. توسعه‌دهنده می‌داند که من Model را کجا تعریف کرده‌ام، Controller مربوط به هر View کجاست و در کدام قسمت با پایگاه داده ارتباط برقرار کرده‌ام؛ اما در مورد کدهای JavaScript و سمت کلاینت چه طور؟ توسعه‌دهنده‌ای که می‌خواهد کار من را ادامه بدهد دچار وحشت می‌شود! الگوی مشخصی وجود ندارد؛ معلوم نیست که کجا DOM را دستکاری کرده‌ام، در کدام قسمت با سرور ارتباط برقرار شده و ... به قول معروف با یک اسپاگتی کد تمام عیار طرف می‌شود. AngularJS این مشکل را حل نموده و ساختار خاصی را سعی کرده به شما دیکته کند و تا حد ممکن دست شما را نیز باز گذاشته است. جدا از همه‌ی اینها، برنامه‌های مبتنی بر Angular به راحتی نگه داری و تست می‌شوند و بدون هیچ دغدغه‌ای آن‌ها را می‌توان توسعه داد.

در حاشیه

شاید در هنگام دریافت فایل angularjs و افزودن آن به پروژه‌ی خود شروع به اعتراض کرده‌اید که نسخه‌ی فشرده شده‌ی آن 87 کیلو بایت حجم دارد در صورتی که این حجم در کتابخانه‌های مشابه ممکن است حتی به 10 کیلوبایت هم نرسد. اگر دقت کرده باشید من در بیان AngularJS از واژه‌ی کتاب‌خانه استفاده نکردم و فقط از واژه‌ی فریمورک استفاده کردم. بله نمی‌شود angular را با کتاب‌خانه‌هایی مقایسه کرد که مهمترین ویژگی خود را Data Binding می‌دانند. AngularJS یک بستر کاری قدرتمند است که تمام راه‌حل‌های موجود را در خود جمع کرده است. تیم توسعه‌دهنده‌ی آن هم هیچ ادعایی ندارد و می‌گویند که ما هیچ چیزی را خودمان اختراع نکرده‌ایم، بلکه راه‌حل‌های عالی را برگزیدیم، تفکرهای خوب را ارتقا بخشیده و در فریمورک خود استفاده کردیم و حتی از ایده‌های خوب دیگر کتاب‌خانه‌ها هم استفاده کرده‌ایم. بنابر این نباید به حجم آن در مقابل توانایی‌هایی که دارد اعتراض کرد.

همچنین به نظر می‌آید که AngularJS یک فریمورک پیچیده است. ولی من همیشه بین پیچیده و پیچیده شده تفاوت قائل می‌شوم. به نظر شخصی خودم Angular به دلیل مشکلات خاص و پیچیده‌ای که حل می‌کند پیچیده است و پیچیده شده نیست. اگر آن را پیچیده شده حس می‌کنید، تنها دلیلش، نحوه‌ی آموزش دادن بنده است، تمام سعی خود را می‌کنم که مفاهیم را تا حد ممکن ساده بیان کنم و امیدوارم در آینده که با مثال‌های بیشتری روبرو می‌شوید، این مفاهیم به کارتان بیاید.

در مقاله‌ی بعدی به مفاهیم انقیاد داده، تزریق وابستگی، هدایت گر‌ها (Directives) و سرویس‌ها در AngularJS می‌پردازم.

[دریافت مثال این قسمت](#)

نظرات خوانندگان

نویسنده: دادخواه
تاریخ: ۲۳:۵۲ ۱۳۹۲/۰۶/۰۹

سلام؛ بحث امنیت مخصوصا در طراحی چی میشه؟ در این روش تمام تمپلیت ها و طراحی ها در سمت کاربر میشه. ولی در روش ASP.net مثلا کدهای Razor برای کاربر فرستاده نمیشه. فقط کد HTML ساخته شده فرستاده میشه و سخت تر میشه فهمید طراح دقیقا چه کرده.
تشکر

نویسنده: چارلی
تاریخ: ۷:۵۷ ۱۳۹۲/۰۶/۱۰

مشکل امنیت برای چی؟ شما از سمت سرور اطلاعات مورد نیاز دریافت میکنید بدون نیاز به اینکه چجوری تولید شدند مثل Razor و با این فریمورک باهش کار میکنید قرار نیست منطق برنامه بیارید سمت کلاینت که اگه این روش دیدید مطمئن باشید یه جای کار مشکل داره!

نویسنده: دادخواه
تاریخ: ۱۰:۴۹ ۱۳۹۲/۰۶/۱۰

مگر در این روش در سمت کلاینت تمپلیت ها را تعریف نمی کنید؟
تمپلیت ها و کنترلرها و توابع نیز در فایل JS هست که ان هم به سمت کاربر ارسال میشه. خب حالا کاربر صفحه را ذخیره می کنه فایل های JS را هم داره. فقط تنها چیزی را که نداره اطلاعات بانک هست که انرا هم از طریق Ajax ی که صادر میشه حدودا میشه فهمید قضیه چیه. فکر کنم به راحتی میشه یه کپی از روی سایت تهیه کرد.

نویسنده: احمد
تاریخ: ۱۲:۳۱ ۱۳۹۲/۰۶/۱۰

- 1- منطق برنامه در Razor نوشته نمیشود ، بلکه در کلاسها و توابع پیاده سازی شده است.
- 2- سایت بدون data به درد کسی نمیخورد. (مثلا جی میل که به کدهای سمت کلاینت(html,js,...) دسترسی وجود دارد)
- 3- بهتر است بار تولید UI سمت کلاینت باشد تا سرور. در سمت سرور باید فقط data رد و بدل شود.

نویسنده: سعید پیروز
تاریخ: ۱۲:۵۴ ۱۳۹۲/۰۶/۱۰

سلام؛ اگه می شه در مورد فعال کردن intellisense برای angular در vs2012 هم یک توضیحی بدید.

نویسنده: مهدی سعیدی فر
تاریخ: ۱۳:۴ ۱۳۹۲/۰۶/۱۰

[اینجا](#)

[پلاگینی برای resharper](#)

visual studio 2013 به صورت پیش فرض از angular پشتیبانی می کند. در ضمن به آن صورت هم فکر نکنم احتیاجی به intellisense باشد. من به شخصه بدون intellisense به راحتی ازش استفاده می کنم.

نویسنده: محسن خان

تاریخ: ۲۲:۱۶ ۱۳۹۲/۰۶/۱۰

در مورد ترکیب Client Side Templates با MVC: یکی از خوبی‌های بازگشت دادن یک partial view کامل در MVC (که بله، یک HTML کامل رو بر می‌گردونه [در حالت Ajax ایی مثلا](#)) نسبت به این روش، امکان استفاده از متدهای کمکی سمت سرور برای رندر کردن View هست. مثلا فرض کنید یک لیست فایل‌ها قراره نمایش داده بشه. در View یا Partial View میشه بدون تعریف یک کلاس اضافه‌تر برای بازگشت دادن اطلاعات به صورت JSON که بخواد در AngularJS سمت کلاینت استفاده بشه، اطلاعات رو خیلی ساده برای نمایش، با razor و سی‌شارپ فرمت کرد. مثلا تاریخ رو شمسی کرد. اندازه رو به کیلوبایت یا مگابایت نمایش داد (در حد فراخوانی یک متد الحاقی). یک if و else گذاشت که اگر کاربر لاگین بود این قسمت از partial view رو که درون حلقه داره تولید میشه، مشاهده نکنه یا برعکس. یک قسمت از حلقه هم یک فرم کوچک درست کرد برای ارسال دیتا به سرور اون هم فرمی که آدرسش رو از T4MVC به صورت strongly typed می‌گیره و یا فیلدهاش از Html Helperهای MVC استفاده می‌کنند که این‌ها هم سمت سرور رندر می‌شن. الان چون تمام کار با جاوا اسکریپت باید انجام بشه، یعنی تمام این مراحل رو باید به صورت JSON بازگشت داد که AngularJS بخواد اون‌ها رو سمت کلاینت، سر هم کنه. به علاوه امکان کامپایل کردن Viewهای razor و یافتن خطاهای احتمالی رو هم از دست می‌دیم چون همه چیز قراره سمت کلاینت رندر بشه.

نویسنده: مهدی سعیدی فر
تاریخ: ۸:۵۱ ۱۳۹۲/۰۶/۱۱

احساس می‌کنم، کمی از صحبت‌های من اشتباه برداشت شده است. قالب باید HTML باشد، اما مهم نیست که این قالب توسط چه کسی تولید شده است. برای مثال من در پروژه‌ی خودم یک کنترلر تعریف کرده‌ام که در آن همهی actionها فقط partialview بر می‌گردانند. حال قبل از اینکه این فایل‌های cshtml تبدیل به html شوند و به کلاینت برگردانده شوند، من با razor عملیات دلخواه خود را انجام می‌دهم.

برای اینکه تاریخ‌ها را شمسی کرده و از این قبیل چیدمان داده‌ها، قبل از اینکه تبدیل به json شده و به کلاینت پاس داده شوند، باید در یک حلقه داده‌های مورد نظر را به فرمت مورد نظر درآورده و در نهایت تبدیل به json کرده و به کلاینت بگردانند.

برای حل مشکل T4MVC می‌توان در همان ابتدای کار تمام آدرس‌های مورد نظر را در یک شی جاوا اسکریپت global تعریف کرد و در سراسر برنامه از آن استفاده کرد.

شاید دلایل شما برای این مثال کوچک منطقی به نظر آید، اما هدف angular حل مشکلات برنامه‌های بزرگ و حرفه‌ای است. برای مثال نوشتن یک filemanager با استفاده از angular فوق العاده لذت بخش است و به راحتی می‌توان یک فایل منیجر حرفه‌ای را با آن نوشت. برای آنهم برنامه داریم، اما اگر وقت شود...

نویسنده: سالار
تاریخ: ۹:۵۹ ۱۳۹۲/۰۶/۱۱

با سلام. اگر partialview ما نیاز به اعتبارسنجی داشته باشد، ما در سمت سرور می‌توانستیم یک سری attribute برای اعتبارسنجی هر خاصیت درون ویومدل‌های خود ایجاد کنیم، و خود razor این attributeها را درون html نهایی رندر می‌کرد. آیا این امکان در سمت کلاینت برای انگولار نیز وجود دارد؟

نویسنده: مهدی سعیدی فر
تاریخ: ۱۰:۵۰ ۱۳۹۲/۰۶/۱۱

خود angularjs دارای یک سری api قدرتمند برای اعتبارسنجی هستند. البته انتظار helperهای شسته رفته asp.net mvc را نداشته باشید که بر اساس model بتواند اعتبارسنجی سمت کلاینت کند. باید برایش خودتان دستی کد بنویسید. البته فکر نکنم تهیه‌ی helper برای angular که بر اساس model، کدهای متناظر اعتبارسنجی را تولید کند کار مشکلی باشد.

نویسنده: وحید م
تاریخ: ۲۳:۰۶ ۱۳۹۲/۰۹/۲۹

با سلام سوالی داشتیم در بالا فرمودید: "من در پروژه خودم یک کنترلر تعریف کردم که در آن همه‌ی actionها فقط partial view برگرداند" یعنی چی؟
 "حال قبل از اینکه این فایل‌های cshtml تبدیل به html شوند و به کلاینت برگردانده شوند" این جمله هم نامفهوم بود. ممنون میشم اگر توضیح بیشتری بدید.

نویسنده: محسن خان
 تاریخ: ۱۳۹۲/۰۹/۳۰ ۱۷:۰۰

مطلب [بارگذاری PartialView با استفاده از jQuery در زمان اجرا](#) را مطالعه کنید.

نویسنده: ناصر طاهری
 تاریخ: ۱۳۹۲/۰۹/۳۰ ۱۲:۰۷

راه دیگه برای بارگذاری صفحات تعریف [مسیر یاب](#) است فرض کنید مسیر زیر را تعریف کرده ایم :

```
var PostApp = angular.module('PostApp', []).config(['$routeProvider',
function ($routeProvider) {
    $routeProvider.
        when('/list', {
            templateUrl: '/Administrator/Post/Index',
            controller: 'PostController'
        });
}]);
```

در قسمت templateUrl مسیر یک اکشن Index در کنترلری به نام Post است که یک partial view بر میگردداند به شکل زیر :

```
public virtual ActionResult Index(int? id)
{
    var model = new PostViewModels
    {
        //.....
    };
    return PartialView(viewName: "_Index", model: model);
}
```

در این اکشن ما مدل را به partial view ارسال میکنیم و ویو توسط razor رندر میشود و نتیجه که یک فایل html است بازگشت داده میشود و ما میتوانیم داخل این html از امکانات Angular استفاده کنیم یعنی:
 "قبل از اینکه این فایل‌های cshtml تبدیل به html شوند و به کلاینت برگردانده شوند، من با razor عملیات دلخواه خود را انجام میدهم."

```
@using ViewModels.Administrator.Post
// استفاده از امکانات Razor
@(Html.EnumDropDownListMenu<PostPermiton, AppViewPostResource>("permiton-", "{{item.id}}"))
// استفاده از امکانات Angular
<div ng-controller="PostController">
    <ul>
        <li ng-repeat="item in ListOfItems">
            {{item.Title}}
        </li>
    </ul>
</div>
```

نویسنده: محسن خان
 تاریخ: ۱۳۹۲/۰۹/۳۰ ۱۳:۰۸

ListOfItems در مدل MVC قابل استفاده است در AngularJS؟

ما داخل صفحه‌ی Partial View میتونیم از امکانات Angular برای زمان بازگشت به سمت کلاینت استفاده کنیم. ListItem مربوط به زمانی میشود که صفحه‌ی رندر شده و در اختیار کلاینت قرار گرفته است. و آماده استفاده از داده‌های در اختیار قرار داده شده توسط متغیری آرایه ای به نام ListItem در کنترلر موجود در Angular است. یعنی صفحه رندر شده میشود به چیزی شبیه به این :

```
// استفاده از امکانات Razor
لیست مطالب

// استفاده از امکانات Angular
<div ng-controller="PostController">
  <ul>
    <li ng-repeat="item in ListItem">
      {{item.Title}}
    </li>
  </ul>
</div>
```

و حالا که یک صفحه‌ی HTML خام شده است میتوانید از آن استفاده کنید. و این هم کنترلری که این صفحه را مدیریت میکند برای مثال :

```
PostApp.controller('PostController', function ($scope, $http, postServices) {
  //...
  $scope.ListOfItems =
    postServices.GetPosts(post)
      .success(function (data) {
        $scope.ListOfItems = data;
      });
  //...
})
```

سلام
بنده طبق فرمایشات شما به روش زیر عمل کردم ولی بهم کار نمیده و پارشال مورد نظر را به من نشان نمیده. میشه بگید مشکل از کجاست؟

```
public class PostController : Controller
{
    ApplicationDbContext db;
    // GET: Post
    public ActionResult List()
    {
        using (db=new ApplicationDbContext())
        {
            var query = db.Posts.ToList();
            return PartialView("List",query);
        }
    }
}
```

و کدهای ویو:

```
<div ng-app="postmodule">
  <div ng-controller="PostController">
    <ul>
      <li ng-repeat="item in ListItem">
        {{item.Title}}
      </li>
    </ul>
  </div>
</div>
```

```

        <hr />
        {{item.Text}}
    </li>
</ul>
</div>
</div>

```

و کدهایی که در فایل Layout.cshtml نوشته ام:

```

<div class="container body-content" >

    <script src="~/Scripts/angular.js"></script>
    <script src="~/Scripts/angular-route.js"></script>
    <script>
        var PostApp = angular.module('postmodule', []).config(['$routeProvider',
function ($routeProvider) {
    $routeProvider.
        when('/list', {
            templateUrl: '/Post/List',
            controller: 'PostController'
        });
    });
    PostApp.controller('PostController', function ($scope, $http, postServices) {
        //...
        $scope.ListOfItems =
            postServices.GetPosts()
                .success(function (data) {
                    $scope.ListOfItems = data;
                });
        //...
    });
    PostApp.service('postServices', function ($http) {
        this.GetPosts = function () {
            return $http.get('/Post/List');
        };
    });
</script>

    <a href="#list">list post</a>
    <div ng-view=""></div>

    @RenderBody()
    <hr />
    <footer>
        <p>&copy; @DateTime.Now.Year - My ASP.NET Application</p>
    </footer>
</div>

```

نویسنده: عزیزخانی
تاریخ: ۱۹:۱۴ ۱۳۹۳/۱۰/۲۳

باید ngRoute رو اضافه کنید

```
var PostApp = angular.module('postmodule', ['ngRoute']).config( ...
```

view و list هم جداگانه برگشت داده شوند

در این مقاله مفاهیم انقیاد داده (Data Binding)، تزریق وابستگی (Dependency Injection)، هدایت گرها (Directives) و سرویس‌ها را بررسی خواهیم کرد و از مقاله‌ی آینده، به بررسی ویژگی‌ها و امکانات AngularJS در قالب مثال خواهیم پرداخت.

انقیاد داده (Data Binding)

سناریو هایی وجود دارد که در آن‌ها باید اطلاعات قسمتی از صفحه به صورت نامتقارن (Asynchronous) با داده‌های دریافتی جدید به روز رسانی شود. روش معمول برای انجام چنین کاری؛ دریافت داده‌ها از سرور است که عموماً به فرم HTML میباشند و جایگزینی آن با بخشی از صفحه که قرار است به روز رسانی شود، اما حالتی را در نظر بگیرید که با داده‌هایی از جنس JSON طرف هستید و اطلاعات صفحه را با این داده‌ها باید به روز رسانی کنید. معمولاً برای حل چنین مشکلی مجبور به نوشتن مقدار زیادی کد هستید تا بتوانید به خوبی اطلاعات View را به روز رسانی کنید. حتماً با خودتان فکر کرده‌اید که قطعاً راهی وجود دارد تا بدون نوشتن کدی، قسمتی از View را به Model متناظر خود نگاشت کرده و این دو به صورت بلادرنگ از تغییرات یکدیگر آگاه شوند. این عمل عموماً به مفهوم انقیاد داده شناخته می‌شود و Angular هم به خوبی از انقیاد داده دوطرفه پشتیبانی می‌کند. برای مشاهده این ویژگی در Angular، مثال مقاله‌ی قبل را به کدهای زیر تغییر دهید تا پیغام به صورت پویا توسط کاربر وارد شود:

```
<!DOCTYPE html>
<html ng-app>
<head>
  <title>Sample2</title>
</head>
<body>
  <div>
    <input type="text" ng-model="greeting.text" />
    <p>{{greeting.text}}, World!</p>
  </div>
  <script src="../../Scripts/angular.js"></script>
</body>
</html>
```

بدون نیاز به حتی یک خط کد نویسی! با مشخص کردن input به عنوان Model از طریق ng-model، خاصیت greeting.text که در داخل {{ }} مشخص شده را به متن داخل textbox مقید (bind) کردیم. نتیجه می‌گیریم که جفت آکلود {{ }} برای اعمال Data Binding استفاده می‌شود. حال یک دکمه نیز بر روی فرم قرار می‌دهیم که با کلیک کردن بر روی آن، متن داخل textbox را نمایش دهد.

```
<!DOCTYPE html>
<html ng-app>
<head>
  <title>Sample2</title>
</head>
<body>
  <div ng-controller="GreetingController">
    <input type="text" ng-model="greeting.text" />
    <p>{{greeting.text}}, World!</p>
    <button ng-click="showData()">Show</button>
  </div>
  <script src="../../Scripts/angular.js"></script>
  <script>
    var GreetingController = function ($scope, $window) {
      $scope.greeting = {
        text: "Hello"
      };

      $scope.showData = function () {
        $window.alert($scope.greeting.text);
      };
    };
  </script>
</body>
</html>
```

به کمک ng-click، تابع showData به هنگام کلیک شدن، فراخوانی می‌شود. \$window نیز به عنوان پارامتر کلاس GreetingController مشخص شده است. \$window نیز یکی از سرویس‌های پیش فرض تعریف شده توسط Angular است و ما در اینجا در سازنده‌ی کلاس آن را به عنوان وابستگی درخواست کرده ایم تا توسط سیستم تزریق وابستگی توکار، نمونه‌ی مناسب آن در اختیار ما بگذارد. \$window نیز تقریباً معادل شی window است و یکی از دلایل استفاده از آن ساده‌تر شدن نوشتن آزمون‌های واحد است.

حال متنی را داخل textbox نوشته و دکمه‌ی show را فشار دهید. متن نوشته شده را به صورت یک popup مشاهده خواهید کرد. همچنین شی \$scope نیز نمونه‌ی مناسب آن توسط سیستم تزریق وابستگی Angular، در اختیار Controller قرار می‌گیرد و نمونه‌ی در اختیار قرار گرفته، برای ارتباط با View Model و سیستم انقیاد داده استفاده می‌شود. معمولاً انقیاد داده در الگوی طراحی MVC سازگار است، این امکان در Angular گنجانده شده است. **تزریق وابستگی (Dependency Injection)** الگوی طراحی MVC سازگار است، این امکان در Angular گنجانده شده است. تزریق وابستگی (Dependency Injection) الگوی طراحی MVC سازگار است، این امکان در Angular گنجانده شده است. تا به این جای کار قطعاً بارها و بارها اسم آن را خوانده اید. در مثال فوق، پارامتری با نام \$scope را برای سازنده‌ی کنترلر خود در نظر گرفتیم و ما بدون انجام هیچ کاری نمونه‌ی مناسب آن را که برای انجام اعمال انقیاد داده با viewmodel استفاده می‌شود را دریافت کردیم. به عنوان مثال، \$window را نیز در سازنده‌ی کلاس کنترلر خود به عنوان یک وابستگی تعریف کردیم و تزریق نمونه‌ی مناسب آن توسط سیستم تزریق وابستگی توکار Angular صورت می‌گرفت. اگر با IOC Containerها در زبانی مثل C# کار کرده باشید، قطعاً با IOC Container فراهم شده توسط Angular هم مشکلی نخواهید داشت.

اما یک مشکل! در زبانی مثل C# که همه‌ی متغیرهای دارای نوع هستند، IOC Container با استفاده از Reflection، نوع پارامترهای درخواستی توسط سازنده‌ی کلاس را بررسی کرده و با توجه به اطلاعاتی که ما از قبل در دسترس آن قرار داده بودیم، نمونه‌ی مناسب آن را در اختیار درخواست کننده می‌گذارد.

اما در زبان جاوا اسکریپت که متغیرها دارای نوع نیستند، این کار به چه شکل انجام می‌گیرد؟ Angular برای این کار از نام پارامترها استفاده می‌کند. برای مثال Angular از نام پارامتر \$scope می‌فهمد که باید چه نمونه‌ای را به کلاس تزریق کند. پس نام پارامترها در سیستم تزریق وابستگی Angular نقش مهمی را ایفا می‌کنند.

اما در زبان جاوا اسکریپت، به طور پیش فرض امکانی برای به دست آوردن نام پارامترهای یک تابع وجود ندارد؛ پس Angular چگونه نام پارامترها را به دست می‌آورد؟ جواب در سورس کد Angular و در تابعی به نام annotate نهفته است که اساس کار این تابع استفاده از چهار عبارت با قاعده (Regular Expression) زیر است.

```
var FN_ARGS = /^function\s*([^\s*(\s*(\[^\s*\)]*\s*)))/m;
var FN_ARG_SPLIT = /,/;
var FN_ARG = /^s*(\s*)(\S+?)\1\s*$/;
var STRIP_COMMENTS = /((\/\/*.*$)|\/\/*[^\s\S]*?\/\/*)/mg;
```

تابع annotate تابعی را به عنوان پارامتر دریافت می‌کند و سپس با فراخواندن متد toString آن، کدهای آن تابع را به شکل یک رشته در می‌آورد. حال کدهای تابع را که اکنون به شکل یک رشته در دسترس است را با استفاده از عبارات با قاعده‌ی فوق پردازش می‌کند تا نام پارامترها را به دست آورد. در ابتدا کامنت‌های موجود در تابع را حذف می‌کند، سپس نام پارامترها را استخراج می‌کند و با استفاده از "،" آن‌ها را جدا می‌کند و در نهایت نام پارامترها را در یک آرایه باز می‌گرداند.

استفاده از تزریق وابستگی، امکان نوشتن کدهایی با قابلیت استفاده مجدد و نوشتن ساده‌تر آزمون‌های واحد را فراهم می‌کند. به خصوص کدهایی که با سرور ارتباط برقرار می‌کنند را می‌توان به یک سرویس انتقال داد و از طریق تزریق وابستگی، از آن در کنترلر استفاده کرد. سپس در آزمون‌های واحد می‌توان قسمت ارتباط با سرور را با یک نمونه فرضی جایگزین کرد تا برای تست، احتیاجی به راه اندازی یک وب سرور واقعی و یا مرورگر نباشد. **Directives**

یکی از مزیت‌های Angular این است که قالب‌ها را می‌توان با HTML نوشت و این را باید مدیون موتور قدرتمند تبدیل گر DOM بدانیم که در آن گنجانده شده است و به شما این امکان را می‌دهد تا گرامر HTML را گسترش دهید.

تا به این جای کار با attributeهای زیادی در قالب HTML روبرو شدید که متعلق به HTML نیست. به طور مثال: جفت آکولادها که برای انقیاد داده به کار برده می‌شود، ng-app که برای مشخص کردن بخشی که باید توسط Angular کامپایل شود، ng-controller که برای مشخص کردن این که کدام بخش از View متعلق به کدام Controller است و ... تمامی Directiveهای پیش فرض Angular هستند.

با استفاده از Directive می‌توانید عناصر و خاصیت‌ها و حتی رویدادهای سفارشی برای HTML بنویسید؛ اما واقعاً چه احتیاجی به

تعریف عنصر سفارشی و توسعه گرامر HTML وجود دارد؟

HTML یک زبان طراحی است که در ابتدا برای تولید اسناد ایستا به وجود آمد و هیچ وقت هدفش تولید وب سایت‌های امروزی که کاملاً پویا هستند نبود. این امر تا جایی پیش رفته است که HTML را از یک زبان طراحی تبدیل به یک زبان برنامه نویسی کرده است و احتیاج به چنین زبانی کاملاً مشهود است. به همین دلیل جامعه‌ی وب مفهومی را به نام [Web Components](#) مطرح کرده است. Web Components به شما امکان تعریف عناصر HTML سفارشی را می‌دهد. برای مثال شما یک تگ سفارشی به نام `datepicker` می‌نویسید که دارای رفتار و ویژگی‌های خاص خود است و به راحتی عناصر HTML را با استفاده از آن توسعه می‌دهید. مطمئناً آینده‌ی وب این گونه است، اما هنوز خیلی از مرورگرها از این ویژگی پشتیبانی نمی‌کنند.

یکی دیگر از معادل‌های Web Component های امروز را می‌توان ویجت‌های jQuery UI دانست. اگر بخواهیم تعریفی از ویجت ارائه دهیم به این گونه است که یک ویجت؛ کدهای CSS، HTML و javascript مرتبط به هم را کپسوله کرده است. مهم‌ترین مزیت ویجت‌ها، قابلیت استفاده‌ی مجدد آن‌هاست، به این دلیل که تمام منطق مورد نیاز را در خود کپسوله کرده است؛ برای مثال ویجت `datepicker` که به راحتی در برنامه‌های مختلف بدون احتیاج به نوشتن کدی قابل استفاده است.

خب، متأسفانه Web Component ها هنوز در دنیای وب امروزی رایج نشده اند و ویجت‌ها هم آنچنان قدرت Web Component ها را ندارند. خب Angular با استفاده از امکان تعریف Directive های سفارشی به صورت cross-browser امکان تعریف عناصر سفارشیه همانند web Component ها را به شما می‌دهد. حتی به عقیده‌ی عده ای Directive ها بسیار قدرتمندتر از Web Components عمل می‌کنند و راحتی کار با آن‌ها بیشتر است.

با استفاده از Directive ها می‌توانید عنصر HTML سفارشی مثل `<datepicker />`، خاصیت سفارشی مثل `ng-controller`، رویداد سفارشی مثل `ng-click` را تعریف کنید و یا حتی حالت و اتفاقات رخ داده در برنامه را زیر نظر بگیرید. و این یکی از دلایلی است که می‌گویند Angular دارای ویژگی `forward-thinking` است.

البته Directive ها یکی از قدرتمندترین امکانات فریم ورک AngularJS است و در آینده به صورت مفصل بر روی آن بحث خواهد شد.

سرویس‌ها در AngularJS

حتماً این جمله را در هنگام نوشتن برنامه‌ها با الگوی طراحی MVC بارها و بارها شنیده اید که در `Controller` ها نباید منطق تجاری و پیچیده ای را پیاده سازی کرد و باید به قسمت‌های دیگری به نام سرویس‌ها منتقل شوند و سپس در سازنده‌ی کلاس کنترلر به عنوان پارامتر تعریف شوند تا توسط Angular نمونه‌ی مناسب آن به کنترلر تزریق شود. `Controller` ها نباید پیاده کننده‌ی هیچ منطق تجاری و یا اصطلاحاً `business` برنامه باشد و باید از لایه‌ی سرویس استفاده کنند و تنها وظیفه‌ی کنترلر باید مشخص کردن انقیاد داده و حالت برنامه باشد.

دلیل استفاده از سرویس‌ها در کنترلر ها، نوشتن ساده‌تر آزمون‌های واحد و استفاده‌ی مجدد از سرویس‌ها در قسمت‌های مختلف پروژه و یا حتی پروژه‌های دیگر است.

معمولاً اعمال مرتبط در ارتباط با سرور را در سرویس‌ها پیاده سازی می‌کنند تا بتوان در موقع نوشتن آزمون‌های واحد یک نمونه‌ی فرضی را خودمان ساخته و آن را به عنوان وابستگی به کنترلر که در حال تست آن هستیم تزریق کنیم، در غیر این صورت احتیاج به راه اندازی یک وب سرور واقعی برای نوشتن آزمون‌های واحد و در نتیجه کند شدن انجام آزمون را در بر دارد. قابلیت استفاده‌ی مجدد سرویس هم به این معناست که منطق پیاده سازی شده در آن نباید ربطی به رابط کاربری و ... داشته باشد. برای مثال یک سرویس به نام `userService` باید دارای متد هایی مثل دریافت لیست کاربران، افزودن کاربر و ... باشد و بدیهی است که از این سرویس‌ها می‌شود در قسمت‌های مختلف برنامه استفاده کرد. همچنین سرویس‌ها در Angular به صورت Singleton در اختیار کنترلرها قرار می‌گیرند و این بدین معناست که یک نمونه از هر سرویس ایجاد شده و به بخش‌های مختلف برنامه تزریق می‌شود.

مفاهیم پایه ای AngularJS به پایان رسید. در مقاله بعدی یک مثال تقریباً کامل را نوشته و با اجزای مختلف Angular بیشتر آشنا می‌شویم. با تشکر از [مهدی محزونی](#) برای بازبینی مطلب

در این قسمت قصد داریم تا یک سیستم ارسال دیدگاه را به کمک Angular پیاده سازی کنیم. هدف از این مثال: آشنایی با چند Directive توکار Angular و همچنین آموختن چگونگی کار با سرویس \$http برای ارتباط با سرور است. کدهای HTML زیر را در نظر بگیرید:

```
<div ng-app="myApp">
  <div ng-controller="CommentCtrl">

    <div ng-repeat="comment in comments">
      <div style="float:right;cursor:pointer;" ng-click="remove(comment.Id,$index);">X</div>
      <a href="#">
        
      </a>
      <div>
        <h4>{{comment.Name}}</h4>
        {{comment.CommentBody}}
      </div>
    </div>

    <div>
      <form action="/Comment/Add" method="post">
        <div>
          <label for="Name">Name</label>
          <input id="Name" type="text" name="Name" ng-model="comment.Name" placeholder="Your
Name" />
        </div>
        <div>
          <label for="Email">Email</label>
          <input id="Email" type="text" name="Email" ng-model="comment.Email"
placeholder="Your Email" />
        </div>
        <div>
          <label for="CommentBody">Comment</label>
          <textarea id="CommentBody" name="CommentBody" ng-model="comment.CommentBody"
placeholder="Your Comment"></textarea>
        </div>
        <button type="button" ng-click="addComment()">Send</button>
      </form>
    </div>
  </div>
</div>
```

خب از ابتدا ساختار را مورد بررسی قرار می‌دهیم و موارد ناآشنای آن را توضیح می‌دهیم:

ng-app : خاصیت ng-app جز خواص پیش فرض HTML نیست و یک خاصیت سفارشی است که توسط Angular به صورت پیش فرض تعریف شده است. این خاصیت به Angular می‌گوید که کدام بخش از DOM باید توسط Angular مدیریت و پردازش شود. در اینجا div ای که با خاصیت ng-app مزین شده است به همراه تمامی عناصر فرزند آن توسط موتور پردازش گر DOM توکار مورد پردازش قرار گرفته و اصطلاحاً کامپایل می‌شود. بله! اینجا از لفظ کامپایل شدن برای بیان این فرآیند استفاده کردم. هیچ کدام از این Directive های سفارشی به خودی خود برای مرورگر قابل تفسیر نیست و اینجاست که Angular وارد عمل شده و این Directive ها را به کدهای HTML و جاوا اسکریپت که برای مرورگر قابل فهم است تبدیل می‌کند. به همین جهت با ng-app مشخص می‌کنیم که کدام بخش از DOM باید توسط Angular تفسیر و مدیریت شود. شاید این سوال برای شما مطرح شده باشد که در مثال قبلی ng-app مقداری نداشت و برای تگ html تعریف شده بود. پاسخ این است که در مثال قبلی چون برنامه‌ی ما دارای یک ماژول بیشتر نبود می‌توانستیم از مقدار دهی ng-app صرف نظر کنیم؛ اما در این مثال ما قصد داریم کمی هم مفهوم ماژول را در Angular بررسی کنیم. در نتیجه در این مثال برنامه‌ی ما از ماژولی به نام myApp تشکیل شده است. دلیل اینکه در این مثال ng-app بر روی یک div تعریف شده است این است که همین قسمت از DOM توسط Angular تفسیر شود برای ما کفایت می‌کند. هنگامی ng-app بر روی html تعریف می‌کنیم که قصد داشته باشیم کل صفحه توسط Angular تفسیر شود. **ng-controller** : در Angular کنترلرها تابع سازنده‌ی کلاس‌های ساده‌ی جاوا اسکریپتی هستند که به کمک آن‌ها بخشی از صفحه را مدیریت می‌کنیم. این که کدام بخش از

صفحه توسط کدام کلاس کنترل و مدیریت شود، توسط ng-controller مشخص می‌شود. در اینجا هم عنصری که با ng-controller مشخص شده به همراه تمامی فرزندانش، توسط کلاس جاوا اسکریپتی به نام CommentCtrl مدیریت می‌شود. در حقیقت ما به کمک ng-controller مشخص می‌کنیم که کدام قسمت از View توسط کدام Controller مدیریت می‌شود. مرسوم است که در Angular نام کنترلرها با Ctrl خاتمه یابد. **ng-repeat** : همه‌ی نظرات دارای یک قالب html یکسان هستند که به ازای داده‌های متفاوت تکرار شده اند. اگر می‌خواستیم نظرات را استفاده از موتور نمایشی Razor نشان دهیم از یک حلقه‌ی foreach استفاده می‌کردیم. خبر خوب این است که ng-repeat هم دقیقاً به مانند حلقه‌ی foreach عمل می‌کند. در اینجا عبارت comment in comments دقیقاً برابر با آن چیزی است که در یک حلقه‌ی foreach می‌نوشتیم. Comments در اینجا یک لیست به مانند آرایه ای از comment هست که در کنترلر مقدار دهی شده است. پس اگر با حلقه‌ی foreach مشکلی نداشته باشید با مفهوم ng-repeat هم مشکلی نخواهید داشت و دقیقاً به همان شکل عمل می‌نماید. **ng-click** : همان طور که گفتیم Directive‌های تعریف شده می‌توانند یک event سفارشی نیز باشند. ng-click هم یک Directive تو کار است که توسط Angular به صورت پیش فرض تعریف شده است. کاملاً مشخص است که یک تابع به نام remove تعریف شده است که به هنگام کلیک شدن، فراخوانی می‌شود. دو پارامتر هم به آن ارسال شده است. اولین پارامتر Id دیدگاه مورد نظر است تا به سرور ارسال شود و از پایگاه داده حذف شود. دومین پارامتر \$index است که یک متغیر ویژه است که توسط Angular در هر بار اجرای حلقه‌ی ng-repeat مقدارش یک واحد افزایش می‌یابد. \$index هم به تابع remove ارسال می‌شود تا بتوان فهمید در سمت کلاینت کدام نظر باید حذف شود. **ng-src** : از این Directive برای مشخص کردن src عکس‌ها استفاده می‌شود. البته در این مثال چندان تفاوتی بین ng-src و src معمولی وجود ندارد. ولی اگر آدرس عکس به صورت Content/{{comment.Name}}.gif می‌بود دیگر وضع فرق می‌کرد. چرا که مرورگر با دیدن آدرس در src سعی به لود کردن آن عکس می‌کند و در این حالت در لود کردن آن عکس با شکست روبرو می‌شود. ng-src سبب می‌شود تا در ابتدا آدرس عکس توسط Angular تفسیر شود و سپس آن عکس توسط مرورگر لود شود. **{{comment.Name}}** : آلوده‌های دوتایی برای انقیاد داده (Data Binding) با view-model استفاده می‌شود. این نوع انقیاد داده در مثال‌های قبلی مورد بررسی قرار گرفته است و نکته‌ی بیشتری در اینجا مطرح نیست. **ng-model** : به کمک ng-model می‌توان بین متن داخل textbox و خاصیت شمی مورد نظر انقیاد داده بر قرار کرد و هر دو طرف از تغییرات یکدیگر آگاه شوند. به این عمل انقیاد داده دوطرفه (Two-Way Data-Binding) می‌گویند. برای مثال textbox مربوط به نام را به comment.Name و textbox مربوط به email را به comment.Email مقید (bind) شده است. هر تغییری که در محتوای هر کدام از طرفین صورت گیرد دیگری نیز از آن تغییر با خبر شده و آن را نمایش می‌دهد.

تا به اینجای کار قالب مربوط به HTML را بررسی کردیم. حال به سراغ کدهای جاوا اسکریپت می‌رویم:

```
var app = angular.module('myApp', []);
app.controller('CommentCtrl', function ($scope, $http) {
    $scope.comment = {};
    $http.get('/Comment/GetAll').success(function (data) {
        $scope.comments = data;
    })
    $scope.addComment = function () {
        $http.post("/Comment/Add", $scope.comment).success(function () {
            $scope.comments.push({ Name: $scope.comment.Name, CommentBody: $scope.comment.CommentBody });
            $scope.comment = {};
        });
    };
    $scope.remove = function (id, index) {
        $http.post("/Comment/Remove", { id: id }).success(function () {
            $scope.comments.splice(index, 1);
        });
    };
});
```

در تعریف ng-app اگر به یاد داشته باشید برای آن مقدار myApp در نظر گرفته شده بود. در اینجا هم ما به کمک متغیر سراسری angular که توسط خود کتابخانه تعریف شده است، ماژولی به نام myApp را تعریف کرده ایم. پارامتر دوم را فعلا توضیح نمی‌دهم، ولی در این حد بدانید که برای تعریف وابستگی‌های این ماژول استفاده می‌شود که من آن را برابر یک آرایه خالی قرار داده ام. در سطر بعد برای ماژول تعریف شده یک controller تعریف کرده ام. شاید دفعه‌ی اول است که تعریف کنترلر به این شکل را مشاهده می‌کنید. اما چرا به این شکل کنترلر تعریف شده و به مانند قبل به شکل تابع سازنده‌ی کلاس تعریف نشده است؟ پاسخ این است که اکثر برنامه نویسان از جمله خودم دل خوشی از متغیر سراسری ندارند. در شکل قبلی تعریف کنترلر، کنترلر به شکل یک متغیر سراسری تعریف می‌شد. اما استفاده از ماژول برای تعریف کنترلر سبب می‌شود تا کنترلرهای ما روی هوا تعریف نشده باشند و هر یک در جای مناسب خود باشند. به این شکل مدیریت کدهای برنامه نیز ساده‌تر بود. مثلا اگر کسی از شما بپرسد که فلان کنترلر کجا تعریف شده است؛ به راحتی می‌گویید که در فلان ماژول برنامه تعریف و مدیریت شده است. در تابعی که به عنوان کنترلر تعریف شده است، دو پارامتر به عنوان وابستگی درخواست شده است. \$scope که برای ارتباط با view-model و انقیاد داده به کار می‌رود و http که برای ارتباط با سرور به کار می‌رود. نمونه‌ی مناسب هر دوی این پارامترها توسط سیستم تزریق وابستگی تو کار angular در اختیار کنترلر قرار می‌گیرد.

قبلا چگونگی استفاده از \$scope برای اعمال انقیاد داده توضیح داده شده است. نکته‌ی جدیدی که مطرح است چگونگی استفاده از سرویس \$http برای ارتباط با سرور است. سرویس http دارای 4 متد get , post , put و delete است. واقعا استفاده از این سرویس کاملا واضح و روشن است. در متد addComment وقتی که دیدگاه مورد نظر اضافه شد، به آرایه‌ی کامنت‌ها یک کامنت جدید می‌افزاییم و چون انقیاد داده دو طرفه است، بالاافاصله دیدگاه جدید نیز در view به نمایش در می‌آید. کار تابع remove هم بسیار ساده است. با استفاده از index ارسالی، دیدگاه مورد نظر را از آرایه‌ی کامنت‌ها حذف می‌کنیم و ادامه‌ی کار توسط انقیاد داده دو طرفه انجام می‌شود. همان طور که مشاهده می‌شود مفاهیم انقیاد داده دو طرفه و تزریق وابستگی خودکار سرویس‌های مورد نیاز، کار با angularjs را بسیار ساده و راحت کرده است. اصولا در بسیاری از موارد احتیاجی به باز اختراع چرخ نیست و کتابخانه‌ی angular آن را برای ما از قبل تدارک دیده است.

کدهای این مثال ضمیمه شده است. این کدها در Visual Studio 2013 و به کمک ASP.NET MVC 5 و Entity Framework 6 نوشته شده است. سعی شده تا مثال نوشته شده به واقعیت نزدیک باشد. اگر دقت کنید مدل کامنت در مثالی که نوشتم به گونه‌ای است که دیدگاه‌های چند سطحی به همراه پاسخ هایش مد نظر بوده است. به عنوان تمرین نمایش درختی این گونه دیدگاه‌ها را به کمک Angular انجام دهید. کافیسٹ Treeview in Angular را جست و جو کنید؛ مطمئنا به نتایج زیادی می‌رسید. گرچه در مثال ضمیمه شده اگر جست و جو کنید من پیاده سازیش را انجام دادم. هدف از جست و جو در اینترنت مشاهده این است که بیشتر مسائل در Angular از پیش توسط دیگران حل شده است و احتیاجی نیست که شما با چالش‌های جدیدی دست و پنجه نرم کنید. پس به عنوان تمرین، دیدگاه‌های چند سطحی به همراه پاسخ که نمونه اش را در همین سایتی که درحال مشاهده آن هستید می‌بینید را به کمک AngularJS پیاده سازی کنید.

در مقاله‌ی بعدی چگونگی انتقال منطق تجاری برنامه از کنترلر به لایه سرویس و چگونگی تعریف سرویس جدید را مورد بررسی قرار می‌دهم. [AngularSample1.rar](#)

نظرات خوانندگان

نویسنده: ناصر طاهری
تاریخ: ۱۸:۴۹ ۱۳۹۲/۰۸/۲۳

ممنون از مطلبتون. یک سوال کوچک :

شما با استفاده از کتابخانه Newtonsoft.Json لیست خودتون رو سریالایز کردید و بعد بازگشت دادید :

```
var comments = _db.Comments.Include(x => x.Children).ToList().Where(x => x.Parent == null).ToList();
var result = JsonConvert.SerializeObject(comments, Formatting.Indented,
    new JsonSerializerSettings
    {
        ReferenceLoopHandling = ReferenceLoopHandling.Ignore,
    });
return Content(result);
```

در حالی که با این روش هم میشه پاسخ داد هر چند در شی‌های تو در تو ابتدا باید فیلدها رو مشخص کنیم :

```
var comments = _db.Comments.Include(x => x.Children).ToList().Where(x => x.Parent == null).ToList();
return Json(comments , JsonRequestBehavior.AllowGet);
```

تفاوت این دو آیا در حجم مقدار دیتای ارسالی تفاوتی ندارند؟

طول محتوا با روش شما برای من 1278 و زمان 584ms و در روش دوم طول محتوا به 760 و زمان 135ms کاهش پیدا کرد. در مثالی دیگر محتوا با همین دو روش بالا. البته اینها ربطی به مطلب شما نداشت. فقط میخواستم بدونم تفاوت این دو غیر از طول محتوا و زمان ، با همدیگه چیه؟
بیصبرانه منتظر مطلب بعدی شما هستم.

نویسنده: مهدی سعیدی فر
تاریخ: ۱۹:۲۷ ۱۳۹۲/۰۸/۲۳

علت استفاده من از Newtonsoft.Json توانایی سریالایز کردن اشیا تودرتو است. فکر کنم با کمک یک خاصیت به نام IgnoreJsonConvert بتوانیم بگیم که چه خواصی را سریالایز نکند. همچنین این کتابخانه خیلی توانایی دیگر هم دارد. یک نمونش که می‌تونه مفید باشه اینه که نام خواص اشیا را به صورت استاندارد camelCase سریالایز کنه. همچنین از نظر سرعت هم نسبت به نمونه‌ی توکار برتری قابل توجهی داره.

نویسنده: زمان
تاریخ: ۱۹:۳۶ ۱۳۹۲/۰۸/۲۳

با سلام. تشکر بابت مقاله و یک سوال.

شما عملیات CRUD رو در کلاس کنترلر تعریف کردید. در برخی مقالات دیدم که اینها توسط کنترلرهای WebApi مدیریت میشوند. آیا تفاوتی از لحاظ کارایی بین این دو روش وجود دارد یا شما ترجیحاً از روش اول استفاده کردید؟

نویسنده: محسن خان
تاریخ: ۲۰:۱ ۱۳۹۲/۰۸/۲۳

[There is more than one way to skin a cat](#)
[On The Coexistence of ASP.NET MVC and WebAPI](#)

نویسنده: مهدی سعیدی فر
تاریخ: ۲۰:۷ ۱۳۹۲/۰۸/۲۳

شخصاً علاقه ای به استفاده از webapi برای دریافت اطلاعات به فرمت json ندارم. webapi محدودیت‌ها و مزیت‌های خاص خودش را دارد که در اینجا کنترلرهای معمولی بدون محدودیتی کار مورد نیاز من را انجام می‌دهند.

نویسنده: آریو
تاریخ: ۱۳۹۲/۱۱/۱۵ ۰:۱۷

با سلام. من طبق آموزش‌های شما که واقعا عالیه پیش رفتم و توی یه کار عملی به یه مشکلی خوردم.

فایل اسکریپت من اینه :

```
var saman = angular.module('SamanApplication', []);
saman.service('loginService', ['$http', function (http) {
  var loginData = [];
  this.login = function () {
    http.get('Saman/LogOn/IsLoggedIn').success(function (data, status, headers, config) {
      loginData = data;
    });
    return loginData;
  };
}]);
saman.controller('loginController', function ($scope, loginService) {
  $scope.response = [];
  $scope.click = function () { $scope.response = loginService.login(); };
});
```

فایل html هم :

```
<body ng-app="SamanApplication">
  <div ng-controller="loginController">
    <button ng-click="click()">Test</button>
    {{data}}
  </div>
</body>
```

اما مشکل اینجاست که بار اول که کلیک میکنم اطلاعات از سرور میاد ولی در {{data}} نمایش داده نمیشه. اما بار دوم که کلیک میکنم نمایش داده میشه !
امکانش هست راهنمایی کنید ؟

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۱۱/۱۵ ۱:۱۲

http.get به صورت async اجرا میشه و نه synchronous. یعنی زمانیکه فراخوانی شد، سطر return بعدی اجرا میشه و صبر نمی‌کنه تا به اینجا برسه. بهتره از promises استفاده کنید (راه حل استانداردش): ⬇ و ⬆

نویسنده: محمد
تاریخ: ۱۳۹۲/۱۱/۲۳ ۱۱:۰۶

سلام؛ من از ngResource دارم برای گرفتن اطلاعات از سرور استفاده میکنم. آیا استفاده از این مازول به جای http\$ مشکلی به وجود نیاره ؟ کلا میشه در مورد ngResource بیشتر توضیح بدین ؟

نویسنده: سعید رضایی
تاریخ: ۱۳۹۲/۱۲/۲۱ ۱۲:۱۰

با عرض سلام من از آموزشتون استفاده کردم وقتی دیتا رو ذخیره می‌کنم مقادیر به صورت null ذخیره میشه تو دیتابیس اینم کدم:

html

```

<div ng-app="myApp" id="ng-app">

<div ng-controller="MenuCtrl" style="width:300px">

    <div style="height:200px;overflow:auto;">
        <div ng-repeat="menu in menu" >
<div style="float:right;cursor:pointer;" ng-click="remove(menu.ID,$index);">X</div>
<a href="#">

</a>
<div>
<h4>{{menu.Title}}</h4>
{{menu.Url}}
</div>
</div>
</div>

    <form action="/Menu/Add" method="post">
<div>
<label for="Title">عنوان</label>
<input id="Title" type="text" name="Title" ng-model="menu.Title" placeholder="عنوان" />
</div>
<div>
<label for="Url">آدرس</label>
<input id="Url" type="text" name="Url" ng-model="menu.Url" placeholder="آدرس" />
</div>
<div>
<label for="ParentID">والد</label>
<input id="ParentID" type="text" name="ParentID" ng-model="menu.ParentID" placeholder="والد" />
</div>

<button type="button" ng-click="addmenu()">ذخیره</button>
</form>
</div>
</div>

```

myapp

```

var app = angular.module('myApp', ['ngAnimate']);
app.controller('MenuCtrl', function ($scope, $http) {

    $scope.menu = {};

    $http.get('/Menu/GetAll').success(function (data) {

        $scope.menu = data;

    })
    $scope.addmenu= function () {

        $http.post("/Menu/Add", $scope.menu).success(function () {

            $scope.menus.push({ Title: $scope.menu.Title, Url: $scope.menu.Url, ParentID:
$scope.menu.ParentID });

            $scope.menu = {};

        });
    };

    $scope.remove = function (ID, index) {

        $http.post("/Menu/Remove", { ID: ID }).success(function () {

            $scope.menu.splice(index, 1);

        });
    };

});

```

```

public class MenuController : Controller
{
    //
    // GET: /Menu/
    MyContext _db = new MyContext();
    public ActionResult GetAll()
    {
        var menu = _db.Menus.ToList();
        var result = JsonConvert.SerializeObject(menu, Formatting.Indented,
            new JsonSerializerSettings
            {
                ReferenceLoopHandling = ReferenceLoopHandling.Ignore,
            });
        return Content(result);
    }
    public ActionResult Add(Menu menu)
    {
        _db.Menus.Add(menu);
        _db.SaveChanges();
        return Json("1");
    }
    public ActionResult Remove(int id)
    {
        var selectedMenu = new Menu { ID = id };
        _db.Menus.Attach(selectedMenu);
        _db.Menus.Remove(selectedMenu);
        _db.SaveChanges();
        return Json("1");
    }
    public ActionResult Index()
    {
        return View();
    }
}

```

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۱۲/۲۱ ۱۳:۹

[دیبایگ کنید](#) چه اطلاعات JSON ایی به سرور ارسال میشه؟

نویسنده: سعید رضایی
تاریخ: ۱۳۹۲/۱۲/۲۱ ۱۳:۱۳

null ارسال می‌شه یعنی هیچکدوم از فیلدها مقدار ندارن

نویسنده: علی رضایی
تاریخ: ۱۳۹۳/۰۱/۰۳ ۱۲:۴۳

سلام. با تشکر فراوان برای این آموزش.

در زمان اجرای این برنامه اگر پس از ورود اطلاعات جدید بخواهیم رکوردی را حذف کنیم (قبل از ریفرش)، حذف انجام نمیشود؛ دلیل آن عدم وجود Id نظری است که جدیداً ثبت شده است؛ راه حل آن به شرح ذیل است:

ابتدا در سمت سرور اکش Add باید به شکل ذیل تغییر یابد:

```
public ActionResult Add(Comment comment)
{
    _db.Comments.Add(comment);
    _db.SaveChanges();
    return Json(comment.Id);
}
```

و سپس در سمت کلاینت متد AddComent به شکل ذیل تغییر یابد:

```
$scope.addComment = function () {
    $http.post("/Comment/Add", $scope.comment).success(function (id) {
        $scope.comments.push({Id:id ,Name: $scope.comment.Name, CommentBody:
        $scope.comment.CommentBody });
        $scope.comment = {};
    });
};
```

نویسنده:

مهدی

تاریخ: ۱۱:۳۷ ۱۳۹۳/۰۳/۰۲

آموزشهای خیلی عالی هستن. با اینکه من ASP کار نمیکنم ولی این مقوله برای همه زبانهای تحت وب هست. دستتون درد نکنه.
ممنون

نویسنده:

علی اسدی

تاریخ: ۱۹:۲۸ ۱۳۹۳/۰۳/۱۴

```
//define
app.service('objUser', function ($http) {
    this.user = [{
        id: null,
        firstName: null,
        lastName: null,
        email: null
    }];
    this.userList = function () {
        var promise = $http.get('api/user')
            .success(function (res) {
                return res;
            });
        return promise;
    };
});
//call
app.controller('UserListCtrl', function ($scope, objUser) {
    $scope.user = objUser.user;
    objUser.userList().then(function (promise) {
        $scope.user = promise.data;
    });
});
```

نویسنده: حمیدرضا

تاریخ: ۱۸:۴۱ ۱۳۹۳/۰۷/۲۶

سلام ، من مطالب شما در مورد angularJs رو خوندم،اما مزیت این رو نسبت به c# mvc نفهمیدم

چرا باید یه لایه اضافه کنیم در حالی که همین کار هارم با کنترلرها در mvc میشه انجام داد

نویسنده: محسن خان

تاریخ: ۲۲:۵۹ ۱۳۹۳/۰۷/۲۶

یکی سمت سرور هست. یکی سمت کاربر. AngularJS برای نظم دادن و مدیریت قسمت سمت کاربر که صرفا درون مرورگر اجرا میشه، طراحی شده.

در [پست](#) قبلی با مفاهیم کنترلر و مدل در AngularJS آشنا شدید. قصد دارم روشی را بررسی کنم که یک منبع داده را بین کنترلرهای تعریف شده در یک ماژول را به اشتراک بگذاریم.

ابتدا یک فایل جاوااسکریپت به نام module1 ایجاد می‌کنیم. در این فایل ابتدا ماژول خود را به Angular معرفی کرده و سپس با استفاده از دستور factory سرویس مورد نظر برای به اشتراک گذاری داده را می‌سازیم:

```
var app = angular.module('myApp', []);
app.factory('BookData', function () {
    var books = [
        { code: 1, name: 'book1', },
        { code: 2, name: 'book2', },
        { code: 3, name: 'book3', },
        { code: 4, name: 'book4', },
        { code: 5, name: 'book5', },
    ];
    return books;
});
```

همان طور که در پست قبلی شرح داده شده برای تعریف ماژول از دستور angular.module استفاده می‌کنیم. در خط بعدی یک سرویس به نام BookData را با استفاده از دستور factory در ماژول مربوطه ساخته می‌شود. تابع مورد نظر بک آرایه از کتاب‌ها را که هر کدام از آن‌ها شامل کد و نام است برگشت می‌دهد. قصد داریم کنترلرهای تعریف شده در ماژول myApp بتوانند به این لیست این کتاب‌ها دسترسی داشته باشند. در این مرحله ابتدا یک کنترلر به نام controller1 به صورت زیر می‌سازیم:

```
app.controller('controller1', function ($scope, BookData) {
    $scope.books = BookData;
});
```

تنها نکته قابل ذکر، تزریق مقادیر \$scope و BookData به تابع سازنده کنترلر مربوطه است. از \$scope برای مقید سازی مقادیر مدل به عناصر dom در view استفاده می‌شود و BookData در این جا دقیقا به مقدار برگشت داده شده از سرویس BookData اشاره می‌کند (نام سرویس مورد نظر دقیقا باید با مقداری که به عنوان آرگومان اول در تابع factory پاس می‌دهید یکی باشد). در نتیجه این مقدار را به متغیر books در \$scope نسبت می‌دهیم. برای کنترلر دوم نیز همین مراحل را تکرار می‌کنیم:

```
app.controller('controller2', function ($scope, BookData) {
    $scope.books = BookData;
});
```

در View مورد نظر نیز یک ارجاع به فایل ساخته شده بالا خواهیم داشت و سپس کدهای مربوط به نمایش را به صورت زیر می‌نویسیم (البته ارجاع به فایل اصلی angular.js فراموش نشود):

```
<script type="text/javascript" src="~/scripts/app/controller1.js"></script>

<div ng-app="myApp">
  <div ng-controller="controller1">
    <p>Data from controller1</p>
    <table>
      <tr ng-repeat="book in books">
        <td>
          {{book.code}}
        </td>
        <td>
          {{book.name}}
        </td>
      </tr>
    </table>
  </div>
</div>
```

```
        </tr>
      </table>
    </div>

    <div ng-controller="controller2">
      <p>Data from controller2</p>
      <table>
        <tr ng-repeat="book in books">
          <td>
            {{book.code}}
          </td>
          <td>
            {{book.name}}
          </td>
        </tr>
      </table>
    </div>
  </div>
```

ابتدا در تگ div اول با استفاده از ng-app محدوده ماژول مورد نظر در صفحه را تعیین کرده سپس با استفاده از تگ‌های div جداگانه هر کدام از نواحی تحت کنترل مربوط به کنترلرهای تعریف شده را مشخص می‌کنیم. با استفاده از ng-repeat به راحتی در بین آرایه کتاب‌ها پیمایش کرده و لیست مورد نظر در صفحه نمایش داده می‌شود. (توضیحات مربوط به ng-repeat و {{}} در [پست قبلی](#) شرح داده شده است). خروجی به صورت زیر خواهد بود. واضح است که اطلاعات نمایش داده شده توسط هر دو کنترلر به دلیل استفاده از منبع داده ای یکسان، به یک شکل خواهد بود.

Data from controller1

1	book1
2	book2
3	book3
4	book4
5	book5

Data from controller2

1	book1
2	book2
3	book3
4	book4
5	book5

نظرات خوانندگان

نویسنده: محسن خان
تاریخ: ۹:۵۷ ۱۳۹۲/۰۹/۱۷

ممنون از شما. با توجه به اینکه در factory ایجاد شده یک نمونه جدید از books ایجاد می‌شود، آیا با تزریق آن در سازنده‌های کنترلرهای مختلف، اگر تغییری نیز در یک کنترلر بر روی این books انجام شود، در بقیه هم منعکس می‌شود؟

نویسنده: ناصر طاهری
تاریخ: ۱۰:۴۳ ۱۳۹۲/۰۹/۱۷

بله. اگر متدی را به کنترلر 2 اضافه کنید و سپس یک تگ لینک در view برای فراخوانی این متد، به صورت زیر :

```
sampleApp.controller('controller2', function ($scope, BookData) {  
    $scope.books = BookData;  
    $scope.change = function(){  
        BookData[0].name = 'change this book';  
    }  
});
```

و view :

```
<div ng-controller="controller2">  
    <p>Data from controller2</p>  
    <table>  
        <tr ng-repeat="book in books">  
            <td>  
                {{book.code}}  
            </td>  
            <td>  
                {{book.name}}  
            </td>  
        </tr>  
    </table>  
    <a href="javascript:void(0)" ng-click="change();" >Click here to change book1</a>  
</div>
```

میبینید که در کنترلر شماره 1 هم تغییر میکند.

اگر مطالعه ای اجمالی درباره مزیت‌ها و قدرت‌های فریم ورک Angular داشته باشید یکی از مواردی که بسیار جلب توجه می‌کند مبحث Directive ها است. به کمک Directive ها در Angular می‌توانید کدهای HTML خود را توسعه دهید. این توسعه علاوه بر تعریف تگ‌های جدید، شامل توسعه کلاس‌ها و همچنین ویژگی‌های تگ‌های HTML نیز خواهد بود. کدهای HTML شما بسیار خوانا تر و از طرفی با قابلیت استفاده مجدد می‌شود. البته این پست فقط شروع به کار در این مقوله است زیرا مبحث Directive ها بسیار گسترده‌تر از آن است که بتوان مطالب آن را در یک مقاله گنجاند. برای شروع یک فایل جاوا اسکریپت ایجاد کرده و در ابتدای آن یک ماژول تعریف کنید:

```
var app = angular.module('myApp', []);
```

در این جا نام ماژول را myApp انتخاب کردم. حال یک Directive به نام angry (نام دیرکتیوها را با حروف کوچک آغاز کنید) به صورت زیر ایجاد می‌کنیم:

```
app.directive('angry', function () {
    return {
        restrict: 'E',
        template: '<div style="color:red"> I am angry!</div>'
    }
})
```

تابع سازنده Directive مورد نظر که یک آبجکت را برگشت می‌دهد شامل خواص زیر می‌باشد:

restrict: که چهار مقدار E و A و C و M را می‌پذیرد که به EACM نیز معروف هستند.

E: زمانی که قصد داشته باشیم یک المان جدید بسازیم از E به معنای element در restrict استفاده می‌کنیم(my->my-directive <directive> </my-directive>);

A: زمانی که قصد داشته باشیم Directive مورد نظر به عنوان Attribute در تگ‌ها استفاده شود از A به معنای Attribute در restrict استفاده می‌شود(<div my-directive="exp"></div>);

C: از C نیز برای تعریف Directive به عنوان مقادیر ویژگی کلاس استفاده می‌کنیم(<div class="my-directive: exp"></div>);

M: حالت M نیز برای استفاده Directive در کامنت‌ها است(<!-- directive: my-directive exp -->);

در ادامه یک Directive دیگر به نام happy می‌سازیم:

```
app.directive('happy', function () {
    return {
        restrict: 'A',
        template: '<div style="color:blue"> I am happy!</div>'
    };
})
```

تفاوت اصلی بین این دو Directive در نوع restrict آن‌ها می‌باشد. برای استفاده از این Directive ها در View می‌توان به صورت زیر عمل نمود:

```
<script type="text/javascript" src="~/scripts/Modules/module1.js"></script>

<div ng-app="myApp">
  <angry></angry>
  <div happy></div>
</div>
```

همان طور که می‌بینید دستور Directive اول که را با restrict از نوع E است می‌توان به عنوان یک تگ جدید استفاده کرد.

دستور happy به عنوان ویژگی تگ div مورد استفاده قرار می‌گیرد (به دلیل اینکه restrict آن از نوع A است) که در نهایت خروجی ساده مثال بالا به صورت زیر خواهد بود:



I am angry!
I am happy!

ادامه دارد...

در [پست قبلی](#) با کلیات مفاهیم دیرکتیوها آشنا شدید. در این پست قصد داریم برخی توابع کنترلرهای تعریف شده در Angular را به وسیله دیرکتیوهای تعریف شده در ماژول فراخوانی نماییم. در ادامه این موضوع را طی یک مثال بررسی خواهیم کرد. ابتدا View مورد نظر را به صورت زیر ایجاد می‌کنیم:

```
<script type="text/javascript" src="~/scripts/Modules/module4.js"></script>
<div ng-app="myApp">
  <div ng-controller="myCtrl">
    <span enter>Load More Books</span>
  </div>
</div>
```

برنامه به این صورت است که با ورود نشانگر ماوس بر روی تگ span (فراخوانی رویداد mouseenter برای تگ هایی که دارای دیرکتیو enter باشند) یک تابع به نام loadMoreBook در کنترلر myCtrl فراخوانی می‌شود. یک فایل جاوااسکریپتی به نام myModule بسازید و ماژول مورد نظر را ایجاد نمایید:

```
var app = angular.module('myApp', []);
```

کنترلر مورد نظر را به همراه تابع loadMoreBook را به صورت زیر ایجاد می‌کنیم (البته در اینجا به جای لود واقعی داده از یک alert استفاده کردیم):

```
app.controller('myCtrl', function ($scope) {
  $scope.loadMoreBook = function () {
    alert('Loading Books...');
  }
});
```

حال نوبت به دیرکتیو مورد بحث می‌رسد که به صورت زیر ایجاد می‌شود:

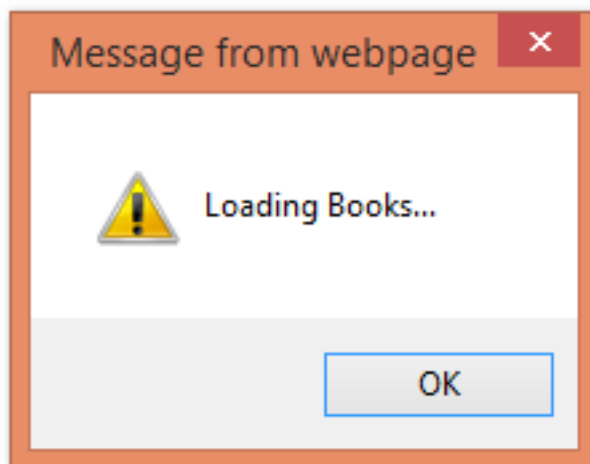
```
app.directive('enter', function () {
  return function (scope, element) {
    element.bind('mouseenter', function () {
      scope.loadMoreBook();
    })
  }
});
```

اولین نکته این است که به در تابع سازنده دیرکتیو به جای برگشت آجک مورد نظر یک تابع برگشت داده می‌شود. برای اینکه بتوان به توابع کنترلر محصور کننده دیرکتیو دسترسی داشت آرگومان اول تابع معادل scope مورد استفاده در کنترلر خواهد بود. آرگومان دوم معادل المانی است که دارای دیرکتیو enter است. در این تابع ابتدا برای رویداد mouseenter رویدادگردان آن پیاده سازی شده است که در آن تابع loadMoreBook کنترلر مورد نظر فراخوانی می‌شود.

خروجی

File Edit View Favorites Tools Help

Load More Books



حال فرض بر این است که در کنترلر بالا تابع دیگری به نام `loadMoreAuthor` برای فراخوانی نویسندگان نیز وجود دارد. به صورت زیر:

```
app.controller('myCtrl', function ($scope) {
  $scope.loadMoreBook = function () {
    alert('Loading Books...');
  }

  $scope.loadMoreAuthor = function () {
    alert('Loading Authors...');
  }
});
```

اما برای انعطاف پذیری بیشتر برنامه، قصد داریم دیرکتیو بالا را به گونه ای تغییر دهیم که نام تابع مورد نظر در کنترلر را به عنوان مقدار یک ویژگی دریافت کند. به صورت زیر:

```
<script type="text/javascript" src="~/scripts/Modules/module4.js"></script>

<div ng-app="myApp">
  <div ng-controller="myCtrl">
    <span enter="loadMoreBook()">Load More Book</span>
    <hr>
    <span enter="loadMoreAuthor()">Load More Author</span>
  </div>
</div>
```

برای به دست آوردن مقدار دیرکتیوی که به عنوان ویژگی در المان تعیین شده، باید از آرگومان سوم در تابع سازنده دیرکتیو به صورت زیر استفاده کرد.

```
app.directive('enter', function () {
  return function (scope, element, attrs) {
    element.bind('mouseenter', function () {
      scope.$apply(attrs.enter);
    })
  }
});
```

در کدهای بالا، برای اینکه بتوان بر اساس نام یک تابع آن را فراخوانی کرد، از سرویس `$apply` که به صورت توکار در `angular` تعبیه شده است استفاده کردم. برای به دست آوردن نام تابع، باید از آرگومان سوم تابع (`attrs`) به همراه نام دیرکتیو استفاده کرد. به دلیل اینکه نام دیرکتیو `enter` است باید پارامتر سرویس `$apply` به صورت `attrs.enter` باشد. خروجی نیز مانند حالت قبل خواهد بود.

همان طور که در پست‌های قبلی ذکر شده بود در angular تزریق وابستگی به صورت پیش فرض وجود دارد. کافیت نام سرویس مورد نظر با نام‌های پیش فرض تعبیه شده در angular یا با نام سرویس‌های ساخته شده توسط خودتان مطابقت داشته باشد. [به عنوان مثال](#) برای تزریق سرویس \$scope در توابع سازنده کنترلر کافیت یک پارامتر به همین نام را به عنوان آرگومان در این توابع در نظر بگیرید. همچنین برای استفاده از سرویس \$http باید یک پارامتر دیگر به همین نام در این توابع در نظر داشته باشید و همچنین برای \$location. در مورد سرویس‌های ساخته شده توسط خودتان نیز باید همین قانون را پیاده کنید. در این پست قصد دارم از injector تعبیه شده در angular برای تغییر رفتار فریم ورک در هنگام شناسایی پارامترهای توابع استفاده کنم. ابتدا مثال زیر را به روش‌های قبلی پیاده سازی می‌کنیم:

```
var app = angular.module('myApp', []);
app.factory('bookService', function () {
    var books = [
        { name: 'A' },
        { name: 'B' },
        { name: 'C' }
    ];
    return books;
});
app.controller('bookCtrl', function ($scope, bookService) {
    $scope.books = bookService;
});
```

view مورد نظر نیز به صورت زیر خواهد بود:

```
<script type="text/javascript" src="~/scripts/Modules/module5.js"></script>
<div ng-app="myApp">
    <div ng-controller="bookCtrl">
        <table>
            <tr ng-repeat="book in books">
                <td>
                    {{book.name}}
                </td>
            </tr>
        </table>
    </div>
</div>
```

نیاز به توضیح نیست که در هنگام تعریف تابع سازنده کنترلر bookCtrl باید نام پارامترهای ورودی تابع در هنگام تزریق وابستگی دقیقاً مانند مثال بالا باشد. (یعنی \$scope برای دسترسی به سرویس scope و bookService برای دسترسی به سرویس ساخته شده توسط factory - ترتیب پارامترها در اینجا اهمیتی ندارد). حال مثال بالا را با استفاده از injector موجود در angular برای تزریق وابستگی‌ها پیاده سازی می‌کنم. ابتدا تابع کنترلر bookCtrl را به صورت زیر ایجاد می‌کنیم:

```
var bookCtrl = function (sc,bs) {
    sc.books = bs;
};
```

از پارامتر sc به جای \$scope و از bs به عنوان bookService در این تابع استفاده شده است. سپس کنترلر موجود را به ماژول مورد نظر نسبت می‌دهیم:

```
app.controller('bookCtrl',bookCtrl);
```

اگر برنامه را به همین صورت اجرا کنید خروجی مورد نظر حاصل نخواهد شد. زیرا آرگومان‌های sc و bs برای angular تعریف نشده است. کافیت وابستگی‌های تابع کنترلر را به صورت زیر برای angular مشخص نماییم:

```
bookCtrl.$inject = ['$scope','bookService'];
```

در نتیجه تعریف کنترلر بالا به صورت کامل زیر خواهد بود:

```
var app = angular.module('myApp', []);
app.factory('bookService', function () {
  var books = [
    { name: 'A' },
    { name: 'B' },
    { name: 'C' }
  ];
  return books;
});
var bookCtrl = function (sc,bs) {
  sc.books = bs;
};
bookCtrl.$inject = ['$scope','bookService'];
app.controller('bookCtrl',bookCtrl);
```

از این پس در هنگام فراخوانی تابع کنترلر bookCtrl سرویس‌های \$scope و bookService به ترتیب به عنوان آرگومان‌های اول و دوم در اختیار کنترلر قرار می‌گیرند. می‌توان به جای فراخوانی مستقیم \$inject، تزریق وابستگی‌ها را در هنگام تعریف توابع سازنده به صورت زیر نیز فراهم ساخت:

```
app.controller('bookCtrl', ['$scope', 'bookService', function (sc, bs) {
  sc.books = bs;
}])
```

نظرات خوانندگان

نویسنده: محسن خان
تاریخ: ۲۳:۰ ۱۳۹۲/۰۹/۲۴

پس با این حساب AngularJS به minification اسکریپت‌ها حساس است. چون در حین minification تمام نام پارامترها با a و b و c جایگزین می‌شوند. در این مورد چه پیشنهادی وجود دارد؟

نویسنده: مهدی سعیدی فر
تاریخ: ۲۳:۱۰ ۱۳۹۲/۰۹/۲۴

درسته. چون سیستم تزریق وابستگی با نام متغیرها کار می‌کند با minification نام متغیرها تغییر می‌کند و در نتیجه برنامه از کار می‌افتد. راه‌های معین کردن صریح وابستگی‌ها در مقاله‌ی فوق ذکر شده. روش اول:

```
bookCtrl.$inject = ['$scope', 'bookService'];
```

روش دوم:

```
app.controller('bookCtrl', ['$scope', 'bookService', function (sc, bs) {
    sc.books = bs;
}])
```

در این روش وابستگی‌های کنترلرها صریحا ذکر شده و با تغییر نام متغیرها انگولار می‌داند که چه وابستگی‌هایی را باید تزریق کند.

نویسنده: وحید
تاریخ: ۱۶:۲۳ ۱۳۹۲/۱۰/۱۴

لطفا توضیحی در مورد \$watch دهید ممنون

نویسنده: مسعود پاکدل
تاریخ: ۲۱:۱ ۱۳۹۲/۱۰/۱۴

به صورت کلی با استفاده از \$watch می‌توان تمامی تغییراتی را که به خواص ViewModel اعمال می‌شوند مشاهده کرد. تعریف کلی آن به صورت زیر است:

```
$watch(watchExpression, listener, objectEquality)
```

«watchExpression: می‌توان نام خاصیت مورد نظر در ViewModel یا یک تابع را که قصد مشاهده تغییرات آن را داریم تعیین کنیم.

«Listener: با تغییر در مقدار watchExpression اگر مقدار قبلی این عبارت با مقدار فعلی آن برابر نباشد این تابع فراخوانی می‌شود.

«objectEquality: به صورت پیش فرض Angular مقادیر مورد نظر برای تغییرات را فقط از نظر Reference Equal بودن چک می‌کند. اگر بخواهیم که Angular به صورت عمقی و درختی مقادیر ابجکت‌ها را بررسی کند مقدار این پارامتر باید true شود.

در فریم ورک Angular هر زمان که عمل مقید سازی خواص ViewModel به عناصر DOM انجام می‌گیرد در واقع یک نمونه از \$watch به لیستی به نام watch list اضافه می‌شود. دقت کنید که صرفا تعریف در محدوده کنترلر کافی نیست بلکه باید خاصیت مورد نظر حتما مقید شود. برای مثال

```
app.controller('MainCtrl', function($scope) {
```

```
$scope.foo = "Foo";  
$scope.world = "World";  
});
```

در View نیز

```
Hello, {{ World }}
```

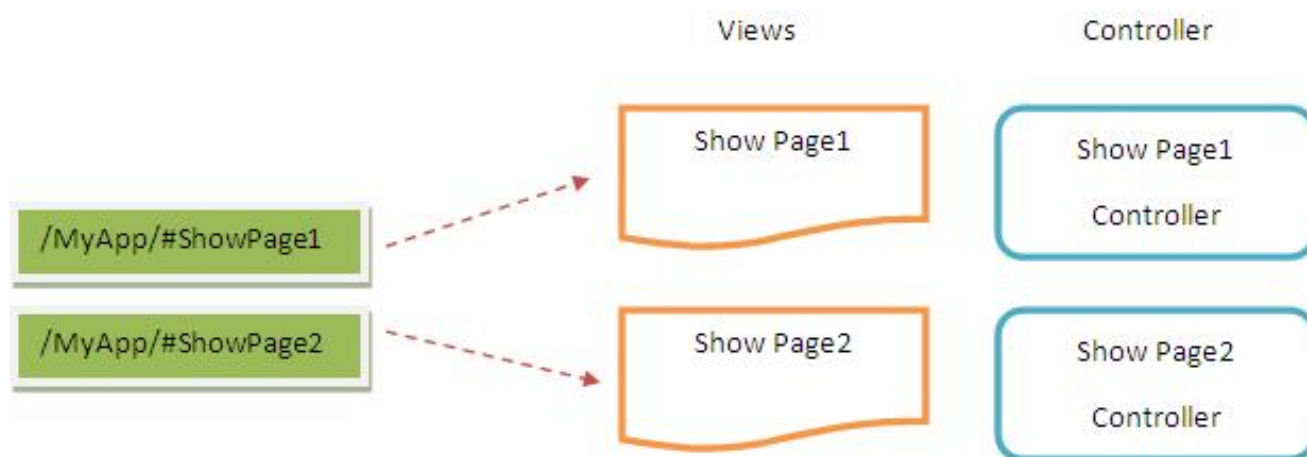
در کنترلر بالا دو خاصیت تعریف شده است، در حالی که در View فقط یک خاصیت مقید شده است. در نتیجه فقط یک \$watch به لیست مورد نظر اضافه شده است.

و به عنوان نکته آخر، در Angular نسخه 1.1.4 تابعی به نام watchCollection اضافه شده است که برای ردیابی تغییرات یک مجموعه مورد استفاده قرار می‌گیرد.

[یک مثال در این مورد](#)

در مطالب قبل کنترلرها و viewها مورد بحث قرار گرفتند. در این پست در نظر داریم یکی از ویژگی‌های دیگر AngularJS به نام مسیریابی (Routing) را مورد بحث قرار دهیم.

یکی از ویژگی‌های برنامه‌های تک صفحه‌ای عدم Reload شدن صفحات است، بر خلاف برنامه‌های وب چند صفحه‌ای که برای نمایش صفحه‌ای دیگر، باید از صفحه‌ای به صفحه‌ای دیگر منتقل شد و عمل Reload هم به طبع نیز اتفاق می‌افتد. در قسمت اول این سری مقالات، مزایای برنامه‌های وب تک صفحه‌ای SPA به صورت کاملتری بیان شده است. در ادامه ما قصد داریم برنامه‌ی وب خود را به صفحات مختلف تقسیم کنیم و سپس با استفاده از امکان مسیریابی موجود در AngularJS آن صفحات را که هر کدام به کنترلری مجزا مقید شده‌اند، در صفحه‌ی اصلی خود بارگذاری کنیم. همچنین استفاده از مسیریابی موجود، میتواند به ما در مدیریت بهتر صفحات کمک فراوانی بکند. به تصویر زیر دقت کنید :



در تصویر بالا دو مسیر با آدرس‌های /ShowPage1 و /ShowPage2 تعریف شده است که هر کدام به یک view مشخص و یک Controller برای مدیریت آن اشاره میکند.

زمانی که ما از تزریق وابستگی‌ها در AngularJS استفاده میکنیم و یک شیء را به کنترلر تزریق میکنیم، Angular توسط Injector سعی در پیدا کردن وابستگی مربوطه و سپس تزریق آن به کنترلر را انجام میدهد. برای استفاده از امکان مسیریابی Route، ما نیز باید از پروایدر مخصوص آن برای تزریق استفاده کنیم. در Angular مسیرهای برنامه توسط پروایدری به نام \$routeProvider شناسایی میشود که خدمات مسیریابی را به ما ارائه میدهد. این سرویس به ما کمک میکند تا بتوانیم اتصال بین کنترلرها، ویوها و آدرس URL جاری مرورگرها را به آسانی برقرار کنیم.

بهتر است کار را شروع کنیم. یک فایل JS ایجاد و سپس محتویات زیر را در آن قرار دهید :

```
var myFirstRoute = angular.module('myFirstRoute', []);
myFirstRoute.config(['$routeProvider',
function($routeProvider) {
$routeProvider.
when('/pageOne', {
templateUrl: 'templates/page_one.html',
controller: 'ShowPage1Controller'
```

```

    }).
    when('/pageTwo', {
      templateUrl: 'templates/page_two.html',
      controller: 'ShowPage2Controller'
    }).
    otherwise({
      redirectTo: '/pageOne'
    });
  }]);

myFirstRoute.controller('ShowPage1Controller', function($scope) {
  $scope.message = 'Content of page-one.html';
});

myFirstRoute.controller('ShowPage2Controller', function($scope) {
  $scope.message = 'Content of page-two.html';
});

```

در کدهای بالا ابتدا یک ماژول تعریف کرده ایم و سپس توسط `config()` تنظیمات مربوط به مسیریابی را انجام داده ایم. با استفاده از متدهای `when` و `otherwise` میتوانیم مسیرها را تعریف کنیم. برای هر مسیر دو پارامتر وجود دارد که اولین پارامتر نام مسیر و دومین پارامتر شامل 2 قسمت میشود که `templateUrl` آن آدرسی که باز خواهد شد و `controller` نیز نام کنترلری که ویو را مدیریت میکند.

توسط `otherwise` میتوانیم مسیر پیشفرض را نیز تعریف کنیم تا در صورتی که مسیری با آدرس‌های بالای آن مطابقت نداشت به این آدرس منتقل شود.

در قطعه کد بالا همچنین دو مسیر با نام‌های `/pageOne` و `/pageTwo` تعریف کرده ایم که هر کدام به ترتیب به `View` های `templates/page_one.html` و `templates/page_two.html` مرتبط شده اند. همچنین دو کنترلر برای مدیریت ویوها نیز تعریف شده است.

زمانی که ما آدرس `http://appname/#pageOne` را در نوار آدرس مرورگر وارد میکنیم، Angular به صورت اتوماتیک آدرس URL را با تنظیماتی که ما در اینجا تعریف کرده ایم مطابقت میدهد و در صورت وجود چنین آدرسی، `view` مربوطه را بارگذاری میکند و در این مثال نیز مطابق با تنظیمات بالا، صفحه‌ی `templates/page_one.html` برای ما بارگذاری و سپس کنترلر `ShowPage1Controller` را فراخوانی میکند، جایی که ما منطق کار را در آن قرار میدهیم.

محتویات فایل `main.html`:

```

<body ng-app="app">
  <div>
    <div>
      <div>
        <ul>
          <li><a href="#pageOne"> Show page one </a></li>
          <li><a href="#pageTwo"> Show page two </a></li>
        </ul>
      </div>
      <div>
        <div ng-view></div>
      </div>
    </div>
  </div>

  <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.0.7/angular.min.js"></script>
  <script src="app.js"></script>
</body>

```

در قطعه کد بالا دو لینک تعریف شده است که ویژگی `href` از علامت هش و نام صفحه تشکیل شده است. یکی از چیزهایی که

شایان ذکر است ، دایرکتیو ng-view است. مکانی برای بارگذاری صفحات در آن.

ما میتوانیم این تگ را به سه شکل زیر نیز استفاده کنیم :

```
<div ng-view></div>
..
<ng-view></ng-view>
..
<div class="ng-view"></div>
```

محتویات صفحه templates/page_one.html :

```
<h2>Page One</h2>
{{ message }}
```

محتویات صفحه templates/page_two.html :

```
<h2>Page Two</h2>
{{ message }}
```

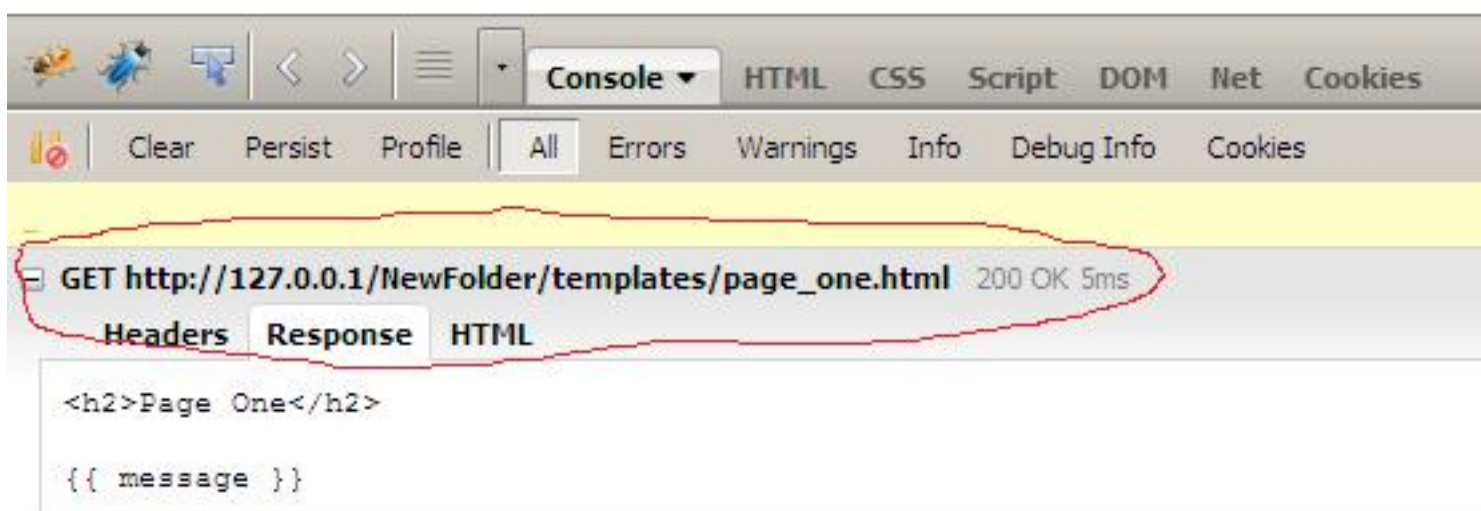
حال اگر پروژه را اجرا کنید و به کنسول مرور گر خود نگاه کنید متوجه میشوید که مسیریاب از مسیر پیشفرض استفاده کرده است و صفحه‌ی page_one.html را به صورت ایجکسی فراخوانی کرده است :

127.0.0.1/NewFolder/#/pageOne

- [Show page one](#)
- [Show page two](#)

Page One

Content of page-one.html



و اگر روی لینک Show Page two کلیک کنید ، صفحه‌ی page_two.html نیز به صورت ایجکسی فراخوانی میشود.

دوباره بر روی لینک Show page one کلیک کنید. بله. هیچ درخواستی به سمت سرور ارسال نشد و صفحه‌ی page_one.html به خوبی نمایش داده شد. یکی از مزیت‌های سیستم مسیریابی قابلیت کش کردن صفحات است تا در صورت فراخوانی مجدد، درخواستی به سمت سرور ارسال نشود و خیلی سریع به شما نمایش داده شود.

مثال این مطلب : [RouteExample.zip](#)

ادامه دارد ...

نظرات خوانندگان

نویسنده: حسین
تاریخ: ۱۳۹۲/۰۹/۲۵ ۱۲:۵۹

سلام این نمونه کد در فایرفاکس به درستی کار میکنه ولی در گوگل کروم یا ie با خطاهای زیر مواجه میشم

```
1)OPTIONS file:///E:/Users/admin/Downloads/Compressed/RouteExample/RouteExample/templates/page_two.html
No 'Access-Control-Allow-Origin' header is present on the requested resource. Origin 'null' is
therefore not allowed access. angular.min.js:99
```

```
2)XMLHttpRequest cannot load
file:///E:/Users/admin/Downloads/Compressed/RouteExample/RouteExample/templates/page_two.html. No
'Access-Control-Allow-Origin' header is present on the requested resource. Origin 'null' is therefore
not allowed access.
```

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۹/۲۵ ۱۳:۳۴

برای اجرا نیاز به سرور داره. ولی کروم برای تست با دستور خط فرمان Chrome.exe --allow-file-access-from-files چنین اجازه‌ای رو به شما میده.

نویسنده: ناصر طاهری
تاریخ: ۱۳۹۲/۰۹/۲۵ ۱۳:۳۴

پروژه را تحت یک سرور مثل Apache یا IIS اجرا کنید. مشکل رفع میشود.

نویسنده: وحید م
تاریخ: ۱۳۹۲/۱۰/۰۵ ۲۳:۵۰

در mvc چطور میشود از حالت روتینگ استفاده کرد بگونه ای که مثلا بجای فایل html بتوان فایل cshtml جهت لود کردن partial view استفاده کرد. و اینکه چگونه مسیر دهی مثل views/home/_partial.cshtml داشته باشیم ممنون از شما

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۱۰/۰۶ ۰:۶

نمیشه مستقیما. چون مسیر کاری MVC از اکشن متد و کنترلرها شروع میشه و نه از View ها. View های قرار گرفته در پوشه Views دارای web.config عدم دسترسی از خارج از سایت هستند. کدهای angularjs هم سمت کاربر هست و نه سمت سرور.

نویسنده: ناصر طاهری
تاریخ: ۱۳۹۲/۱۰/۰۶ ۰:۷

در این [کامنت](#) هم توضیحاتی داده شده.

نویسنده: سعید رضایی
تاریخ: ۱۳۹۳/۰۲/۱۴ ۱۳:۴۶

به جای page_one.html چجوری میشه از cshtml استفاده کرد.
کلا کار درسته که از cshtml فایل ها به templateUrl استفاده کرد یا نه؟
با تشکر

نویسنده: محسن درپرستی
تاریخ: ۱۸:۱ ۱۳۹۳/۰۲/۱۴

اگر بخواید از فایل‌های cshtml استفاده کنید باید از طریق یک اکشن باید اینکار رو بکنید. درست یا غلط بودنش به سناریویی که دارید بستگی دارد.

نویسنده: ناصر طاهری
تاریخ: ۱۸:۵۶ ۱۳۹۳/۰۲/۱۴

یک مثال هم در این کامنت قرار داده شده است. ([+](#))

در [قسمت قبل](#) با نحوه پیاده سازی مسیریابی در AngularJS آشنا شدیم و در این پست میخواهیم نحوه تعریف و ارسال پارامترها به سیستم مسیریاب را فرا بگیریم.

فرض کنید که میخواهیم در لیست سفارشات قسمتی داشته باشیم برای مشاهدهی جزئیات هر سفارش. پس در صفحه نمایش جزئیات کالا نیاز به کد محصول برای واکنشی آن داریم. در Angular زمانی که داریم مسیرها را تعریف میکنیم این امکان را هم داریم که پارامترهایی را هم برای هر مسیر مشخص کنیم. برای این کار فایل app.js مثال قبل را باز کنید و مسیر ذیل را به آن اضافه کنید :

```
when('/showOrderDetails/:orderId', {
  templateUrl: 'templates/show_order.html',
  controller: 'ShowOrderController'
});
```

در بالا ما پارامتری به نام orderId وارد کرده ایم که میتوانیم توسط \$routeParams در کنترلر به آن دست پیدا کنیم :

```
myFirstRoute .controller('ShowOrderController', function($scope, $routeParams) {
  $scope.order_id = $routeParams.orderId;
});
```

فراموش نکنید که باید پارامتر \$routeParams را به کنترلر خود تزریق کنید.

محتوای فایل index.html را نیز به صورت زیر تغییر دهید :

```
<body ng-app="myFirstRoute" style="
  <div>
<div>
<div>
<table dir="rtl">
<thead>
<tr>
<th>#</th><th>کد</th><th>نام محصول</th><th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td><td>1234</td><td>15" Samsung Laptop</td>
<td><a href="#showOrderDetails/1234">جزئیات محصول</a></td>
</tr>
<tr>
<td>2</td><td>5412</td><td>2TB Seagate Hard drive</td>
<td><a href="#showOrderDetails/5412">جزئیات محصول</a></td>
</tr>
<tr>
<td>3</td><td>9874</td><td>D-link router</td>
<td><a href="#showOrderDetails/9874">جزئیات محصول</a></td>
</tr>
</tbody>
</table>

<div ng-view></div>
</div>
</div>
</div>

<script src="js/bootstrap.js"></script>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.0.7/angular.min.js"></script>
<script src="app.js"></script>

</body>
```

نکته‌ی مهم در کد بالا قرار دادن کد کالا بعد از مسیر است، مانند : `showOrderDetails/5412` # و محتویات فایل `templates/show_order.html` :

```
<h2>سفارش شماره<{{order_id}}</h2>
<b>#<{{order_id}}</b>.
محل قرار گیری جزئیات سفارش شماره
```

برنامه را اجرا کنید تا نتیجه را ببینید.

بارگزاری View های محلی توسط تگ <script> :

در بعضی موارد لزومی ندارد که اطلاعات View را از یک فایل دیگر بخوانید و یا حتی اینقدر View شما کوچک است که تمایل دارید آن را به همراه فایل اصلی `index.html` حمل کنید به جای اینکه آن را در یک فایل جدا نگهداری کنید. دایرکتیوی به نام `ng-template` وجود دارد که این امکان را به ما میدهد تا بتوانیم `template` View های کوچکی را در داخل فایل اصلی قرار دهیم. با استفاده از تگ `<script>` به شکل زیر میشود این کار را انجام داد :

```
<script type="text/ng-template" id="add_order.html">
  <h2>ثبت سفارش</h2>
  <{{message}}>
</script>
```

برای درک بهتر مثالی را تهیه میکنیم .

فایل `app.js` مثال قبل را باز کنید و مسیرهای زیر را نیز به آن اضافه کنید :

```
when('/AddNewOrder', {
  templateUrl: 'add_order.html',
  controller: 'AddOrderController'
}).
when('/ShowOrders', {
  templateUrl: 'show_orders.html',
  controller: 'ShowOrdersController'
});
```

سپس دو کنترلر زیر را نیز به آن اضافه کنید :

```
myFirstRoute.controller('AddOrderController', function($scope) {
  $scope.message = 'صفحه نمایش ثبت سفارش جدید';
});

myFirstRoute.controller('ShowOrdersController', function($scope) {
  $scope.message = 'صفحه نمایش لیست سفارشات';
});
```

فایلی به نام `index2.html` برای صفحه اصلی برنامه با محتوای زیر تعریف کنید :

```
<body ng-app="myFirstRoute" style="
  <div>
    <div>
      <div>
        <ul>
          <li><a href="#AddNewOrder">ثبت سفارش جدید</a></li>
          <li><a href="#ShowOrders">نمایش سفارشات</a></li>
        </ul>
      </div>
      <div>
        <div ng-view></div>
      </div>
    </div>
  </div>
  <script type="text/ng-template" id="add_order.html">
    <h2>ثبت سفارش</h2>
    <{{message}}>
```



```

</script>

<script type="text/ng-template" id="show_orders.html">
    <h2> نمایش سفارشات </h2>
    {{message}}
</script>

<script src="js/bootstrap.js"></script>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.0.7/angular.min.js"></script>
<script src="app.js"></script>

</body>

```

همانطور که مشاهده میکنید در کد بالا از 2 تگ اسکریپت برای قرار دادن محتوای View استفاده کرده ایم که خاصیت type آن برابر با text/ng-template و خاصیت id آن نام View template است و دیگر فایل مجزایی برای Viewها ایجاد نکردیم. Angular به صورت خودکار محتوای داخل تگهای Script را به محض فراخوانی آدرسهای موجود در ویژگی id هر تگ به وسیلهی سیستم مسیر یابی، در داخل دایرکتیو ng-view قرار میدهد. پروژه را اجرا کنید تا نتیجه را مشاهده کنید.

افزودن دادههای سفارشی به سیستم مسیریابی :

بیشتر اوقات ممکن است نیاز داشته باشید تا دادههای خاصی را در مسیرهای معینی ارسال کنید. برای مثال ممکن است شما بخواهید از یک کنترلر در مسیرهای مختلف استفاده کنید و برای هر مسیر یک دادهی خاص را نیز ارسال میکنید. به مثال زیر توجه کنید :

```

when('/AddNewOrder', {
    templateUrl: 'templates/add_order.html',
    controller: 'CommonController',
    foodata: 'addorder'
}).
when('/ShowOrders', {
    templateUrl: 'templates/show_orders.html',
    controller: 'CommonController',
    foodata: 'showorders'
});

sampleApp.controller('CommonController', function($scope, $route) {
    //access the foodata property using $route.current
    var foo = $route.current.foodata;

    alert(foo);
});

```

در هر دو مسیر از کنترلر CommonController استفاده کرده ایم با این تفاوت که در مسیر اول یعنی /AddNewOrder یک خاصیت با نام foodata با مقدار addorder تعریف شده است و در مسیر دوم با مقدار showorder. ما میتوانیم با تزریق \$route به کنترلرمان، توسط دستور :

```
$route.current.foodata
```

مقدار موجود در آن را بخوانیم.

نظرات خوانندگان

نویسنده:

علی رضایی

تاریخ:

۲۰:۱۲ ۱۳۹۳/۰۱/۰۵

با سلام و تشکر فراوان جهت اشتراک دانسته هایتان؛

یک سوال:

در صورتی که بخواهیم بخش پایانی مقاله یعنی « افزودن داده‌های سفارشی به سیستم مسیریابی » را پیاده کنیم، همه چیز درست کار میکند، اما اگر از controllerAs استفاده کنیم دیگر route.current\$ در دستر نیست و undefiend میشود. شما به این مشکل برخوردید؟

ممنون

نویسنده:

ناصر طاهری

تاریخ:

۰:۳ ۱۳۹۳/۰۱/۰۶

مشکلی رخ نداد.

1 - ماژول مسیریابی (ngRoute) رو باید تزریق کنید به ماژول اصلی :

```
var myFirstRoute = angular.module('myFirstRoute', ['ngRoute']);
```

2- \$route رو به کنترلر هم تزریق کنید :

```
myFirstRoute.controller('ShowPage1Controller', function ($route) {
  this.message = 'Content of page-one.html';
  //access the foodata property using $route.current
  var foo = $route.current.foodata;
  alert(foo);
});
```

3- این هم یک نمونه از تنظیمات مسیریابی :

```
myFirstRoute.config(['$routeProvider',
  function($routeProvider) {
    $routeProvider.
      when('/pageOne', {
        templateUrl: 'templates/page_one.html',
        controller: 'ShowPage1Controller',
        controllerAs: 'tCtrlOne',
        foodata: 'valueOne'
      }).
      when('/pageTwo', {
        templateUrl: 'templates/page_two.html',
        controller: 'ShowPage2Controller',
        controllerAs: 'tCtrlTwo',
        foodata: 'valueTwo'
      }).
      otherwise({
        redirectTo: '/pageOne'
      });
  }]);
```

البته اینجا فقط برای تصحیح اشتباه است. وگر نه به ControllerAs ارتباطی ندارد. به احتمال زیاد مشکل شما عدم تزریق \$route به کنترلر بوده.

در بخش‌های پیشین ([بخش اول](#) و [بخش دوم](#)) به خوبی با اصول و روش مسیریابی (Routing) در AngularJS آشنا شدیم. در این بخش می‌خواهم به برخی جزئیات درباره مسیریابی بپردازم. اولین موضوع، تغییراتی است که از نسخه 1.2 به بعد در روش استفاده از سرویس مسیریابی در AngularJS بوجود آمده است. از نسخه 1.2 سرویس مسیریابی از هسته اصلی AngularJS خارج شد و برای استفاده از امکانات این سرویس باید فایل angular-route.js و یا angular-route.min.js را به صفحه خود بیفزاییم:

```
<script src="~/Scripts/angular.min.js"></script>
<script src="~/Scripts/angular-route.min.js"></script>
```

سپس باید هنگام تعریف ماژول، ngRoute را به عنوان وابستگی تزریق کنیم:

```
var app = angular.module("mainApp", ['ngRoute']);
```

[روش Controller as در AngularJS](#) که از نسخه 1.2 به بعد امکان استفاده از آن وجود دارد قبلاً معرفی شده است. با پاس کردن خصوصیت controllerAs به متد when می‌توان از view استفاده کرد که در آن از این روش استفاده شده است.

```
.when('/controllerAS', {
  controller: 'testController',
  controllerAs: 'tCtrl',
  template: '<div>{{tCtrl.Title}}</div>'
})
```

باقی ماجرا مانند گذشته است.

موضوع دیگری که پرداختن به آن می‌تواند مفید باشد، بررسی بیشتر متد when است. وقتی در متد config ماژول از \$routeProvider استفاده می‌کنیم، داریم سرویس \$route را تنظیم، مقداردهی اولیه، و نمونه گیری می‌کنیم. درواقع با استفاده از متدهای when و otherwise داریم سرویس \$route را مقداردهی اولیه می‌کنیم ([برای آشنایی با تفاوت factory، service و provider کلیک کنید](#)). خوب! جریان این مقادیری که به عنوان پارامتر به این متدها پاس می‌کنیم چیست؟ متد when به این صورت تعریف شده است:

```
when(string path, object route)
```

پارامتر path در بخش‌های قبل به اندازه کافی معرفی شده است. پارامتر route یک شی است شامل اطلاعاتی که با تطبیق آدرس صفحه با پارامتر path، به \$route.current مقداردهی می‌شود (حالا باید متوجه شده باشید که روال افزودن داده‌های سفارشی به سیستم مسیریابی و دسترسی به آن‌ها که در بخش دوم مطرح شد به چه شکل کار می‌کند). این شی می‌تواند خصوصیات از قبل تعریف شده‌ای داشته باشد که در ادامه آن‌ها را مرور می‌کنیم: **controller**: می‌تواند یک رشته شامل نام کنترلر از قبل تعریف شده، یا یک تابع به عنوان تابع کنترلر باشد. **controllerAs**: رشته‌ای شامل نام مستعار کنترلر. **template**: رشته‌ای شامل قالب html، و یا تابعی که قالب html را باز می‌گرداند. این خصوصیت بر templateUrl اولویت دارد. اگر مقدار این خصوصیت یک تابع باشد، \$routeParams به عنوان پارامتر ورودی به آن پاس می‌شود. **templateUrl**: رشته‌ای شامل مسیر فایل قالب html، و یا تابعی که این رشته را باز می‌گرداند. اگر مقدار این خصوصیت یک تابع باشد، \$routeParams به عنوان پارامتر ورودی به آن پاس می‌شود. **redirectTo**: مقداری برای به روز رسانی \$location، و فراخوانی روال مسیر یابی. این مقدار می‌تواند یک رشته، و یا تابعی که یک رشته را باز می‌گرداند باشد. اگر مقدار این خصوصیت یک تابع باشد، این پارامترها به آن پاس می‌شود: \$routeParams برای دسترسی به پارامترهای آمده در آدرس صفحه جاری.

\$location.path() جاری به صورت یک رشته.

`$location.search()` جاری به صورت یک شی.

caseInsensitiveMatch: یک مقدار منطقی است که مشخص می‌کند بزرگ و کوچک بودن حروف در تطبیق آدرس صفحه با پارامتر route در نظر گرفته بشود یا نه. مقدار پیشفرض این خصوصیت `false` است. یعنی در همه مثالهایی که تا کنون زده شده، اگر بزرگ و کوچک بودن حروف آدرس صفحه با مقدار مشخص شده برای پارامتر route متفاوت باشد، روال مسیریابی انجام نخواهد شد. برای رفع این مشکل کافی است مقدار این خصوصیت را `true` قرار دهیم. برای مثال، مسیر `/controllerAS/` که بالاتر تعریف کرده‌ایم را در نظر بگیرید. اگر `www.mySite.com/#/ControllerAS` را وارد کنیم، هیچ اتفاقی نخواهد افتاد و در واقع این آدرس با route مشخص شده تطبیق پیدا نمی‌کند. اگر بخواهیم کوچک و بزرگ بودن حروف در نظر گرفته نشود، کافیه به این ترتیب عمل کنیم:

```
.when('/controllerAS', {
  controller: 'testController',
  controllerAs: 'tCtrl',
  template: '<div>{{tCtrl.Title}}</div>',
  caseInsensitiveMatch: true
})
```

resolve: نگاشتی از وابستگی‌هایی که می‌خواهیم به کنترلر تزریق شود. [قبلا مفهوم promise توضیح داده شده است](#). اگر هر یک از این وابستگی‌ها یک promise باشد، مسیریاب تا resolve شدن همه آن‌ها یا reject شدن یکی از آن‌ها منتظر می‌ماند. در صورتی که همه promise‌ها resolve شوند، رخداد `$routeChangeSuccess`، و در صورتی که یکی از آن‌ها reject شود رخداد `$routeChangeError` اجرا می‌شود. یکی از کاربردهای resolve زمانیست که بخواهید جلوی تغییر محتویات صفحه، پیش از بارگذاری داده‌ای که از سمت سرور درخواست کرده‌اید را بگیرید.

```
$routeProvider
  .when('/resolveTest',
    {
      resolve: {
        // وابستگی بلافاصله بازمی‌گردد
        person: function () {
          return {
            name: "Hamid Saberi",
            email: "Hamid.Saberi@gmail.com"
          }
        },
        // این وابستگی یک promise بازمی‌گرداند
        // پس تغییر مسیر تا resolve شدن آن به تأخیر می‌افتد
        currentDetails: function ($http) {
          return $http({
            method: 'Get',
            url: '/current_details'
          });
        },
        // می‌توانیم از یک وابستگی در وابستگی دیگر استفاده کنیم
        facebookId: function ($http, currentDetails) {
          $http({
            method: 'GET',
            url: 'http://facebook.com/api/current_user',
            params: {
              email: currentDetails.data.emails[0]
            }
          })
        }
      },
      // بارگذاری فایل‌های اسکریپت مورد نیاز
      fileDeps: function ($q, $rootScope) {
        var deferred = $q.defer();
        var dependencies = [
          'controllers/AboutViewController.js',
          'directives/some-directive.js'
        ];

        // بارگذاری وابستگی‌ها با استفاده از $Script.js
        $script(dependencies, function () {
          // همه وابستگی‌ها بارگذاری شده اند
          $rootScope.$apply(function () {
            deferred.resolve();
          });
        });

        return deferred.promise;
      }
    })
```

```
    }  
  },  
  controller: function ($scope, person, currentDetails, facebookId) {  
    this.Person = person;  
  },  
  controllerAs: 'rtCtrl',  
  template: '<div>{{rtCtrl.Person.name}}</div>',  
  caseInsensitiveMatch: true  
})
```

نظرات خوانندگان

نویسنده: مسعود رمضانی
تاریخ: ۱۳۹۲/۱۲/۲۰ ۱۱:۰

دوست عزیز خیلی مطلب مفیدی بود.

ممنونم (:

در پست‌های قبلی بیان شد که برای پیاده سازی عملیات مقید سازی عناصر View به مدل در کنترلر باید \$scope را به تابع سازنده کنترلر تزریق کرد. برای مثال:

```
var app = angular.module('myApp', []);
app.controller('myController', function ($scope) {
    $scope.name = 'Masoud';
    $scope.family = 'Pakdel';
});
```

View متناظر نیز به صورت می‌باشد:

```
<div ng-app="myApp">
  <div ng-controller="myController">
    <div>
      {{name}} {{family}}
    </div>
  </div>
</div>
```

در Angular 1.2 روشی به نام controller as معرفی شده است که با توجه به نوع پیاده سازی آن نیازی به تزریق \$scope در توابع سازنده نیست. فقط در کنترلر به جای وابستگی مستقیم به \$scope از کلمه کلیدی this و در هنگام عملیات مقید سازی باید از نام مستعار تعیین شده برای کنترلر استفاده نمایید. برای مثال

```
var app = angular.module('myApp', []);
app.controller('myController', function () {
    this.name = 'Masoud';
    this.family = 'Pakdel';
});
```

و استفاده آن در View

```
<div ng-app="myApp">
  <div ng-controller="myController as myCtrl">
    <div>
      {{myCtrl.name}} {{myCtrl.family}}
    </div>
  </div>
</div>
```

در هنگام عملیات [routing](#) نیز می‌توان این عناوین مستعار را برای کنترلر با استفاده از controllerAs مشخص نمود. به صورت زیر:

```
app.config(function($routeProvider){
    $routeProvider.when('/first', {
        templateUrl: 'first.html',
        controller: 'FirstCtrl',
        controllerAs: 'fc' })
    .when('/second', {
        templateUrl: 'second.html',
        controller: 'SecondCtrl',
        controllerAs: 'sc' })
    .otherwise({
        redirectTo: '/first'
    });
});
```

نظرات خوانندگان

نویسنده: مهدی سعیدی فر
تاریخ: ۱۶:۴۵ ۱۳۹۲/۰۹/۲۸

ممنون! می‌خواستم بپرسم به جز syntax استفاده مزیت دیگری نیز دارد؟ چون به شخصه استفاده از \$scope را بیشتر می‌پسندم.

نویسنده: مسعود پاکدل
تاریخ: ۲۱:۲۷ ۱۳۹۲/۰۹/۲۸

خیر. در مجموع نمی‌توان تفاوتی خاص بین این دو روش برشمرد. \$scope روشی کلاسیک است در حالی که controller as در نسخه جدید پشتیبانی می‌شود. مهم‌ترین مزیت روش controller as عدم تزریق \$scope به تابع سازنده کنترلر است؛ اما علاوه بر این به نظر من روش controller as به دلیل استفاده از this که تقریباً تمام برنامه نویسان جاوااسکریپت با آن آشنایی دارند روشی تمیزتر است. البته بد نیست که نگاهی هم به ([^](#)) داشته باشید.

در Angular مکانیزمی وجود دارد که بر اساس آن می‌توان از توابع و خواص تعریف شده در یک کنترلر در سایر کنترلرها نیز استفاده کرد که در واقع از آن به عنوان ارث بری کنترلرها عنوان می‌شود؛ ولی نکته ای که وجود دارد این است که در جاوااسکریپت OOP پشتیبانی نمی‌شود پس چگونه یک آبجکت کنترلر توابع و خصوصیات کنترلر دیگر را به ارث می‌برد؟ با ذکر یک مثال این مورد را بررسی خواهیم کرد.

ابتدا دو کنترلر به صورت زیر ایجاد می‌کنیم:

```
var app = angular.module('myApp', []);

app.controller('parentController', function () {
    this.title = 'Title from parent controller';
});

app.controller('childController', function () {
    this.title = 'Title from child controller';
});
```

در کدهای بالا دو کنترلر به نام parentController و childController ایجاد کردم. از parentController به عنوان کنترلر والد استفاده خواهد شد و قصد داریم که از title آن در کنترلر child استفاده نماییم. View متناظر را به صورت زیر ایجاد خواهیم کرد.

```
<div ng-app="myApp">
  <div ng-controller="parentController as parentCtrl">
    <div ng-controller="childController as childCtrl">
      <input type="text" ng-model="parentCtrl.title" size="100"/>
      <input type="text" ng-model="childCtrl.title" size="100"/>
    </div>
  </div>
</div>
```

اولین نکته این است که تگ div تعریف شده برای کنترلر child در محدوده تگ div کنترلر parent قرار گرفته است. دومین نکته این است که از روش [controller as](#) برای مقید سازی خواص به المان‌های صفحه استفاده شده است. اگر برنامه را اجرا نماییم خروجی زیر را مشاهده خواهید کرد.

File Edit View Favorites Tools Help

Title from parent controller

Title from child controller

هر دو کنترلر دارای یک title مجزا هستند ولی با استفاده از یک اشاره گر توانستیم این دو title را تفکیک نماییم. حال برای دسترسی به title کنترلر parent در سایر کنترلرها کافیه از نام مستعار parentCtrl استفاده نماییم. از آن جا که محدوده کنترلر child در داخل محدوده کنترلر parent است دسترسی به کنترلر parent امکان پذیر می‌باشد.

```
<div ng-app="myApp">
  <div ng-controller="parentController as parentCtrl">
    <div ng-controller="childController as childCtrl">
      <input type="text" ng-model="parentCtrl.title" size="100"/>
      <input type="text" ng-model="parentCtrl.title" size="100"/>
    </div>
  </div>
</div>
```

خروجی:

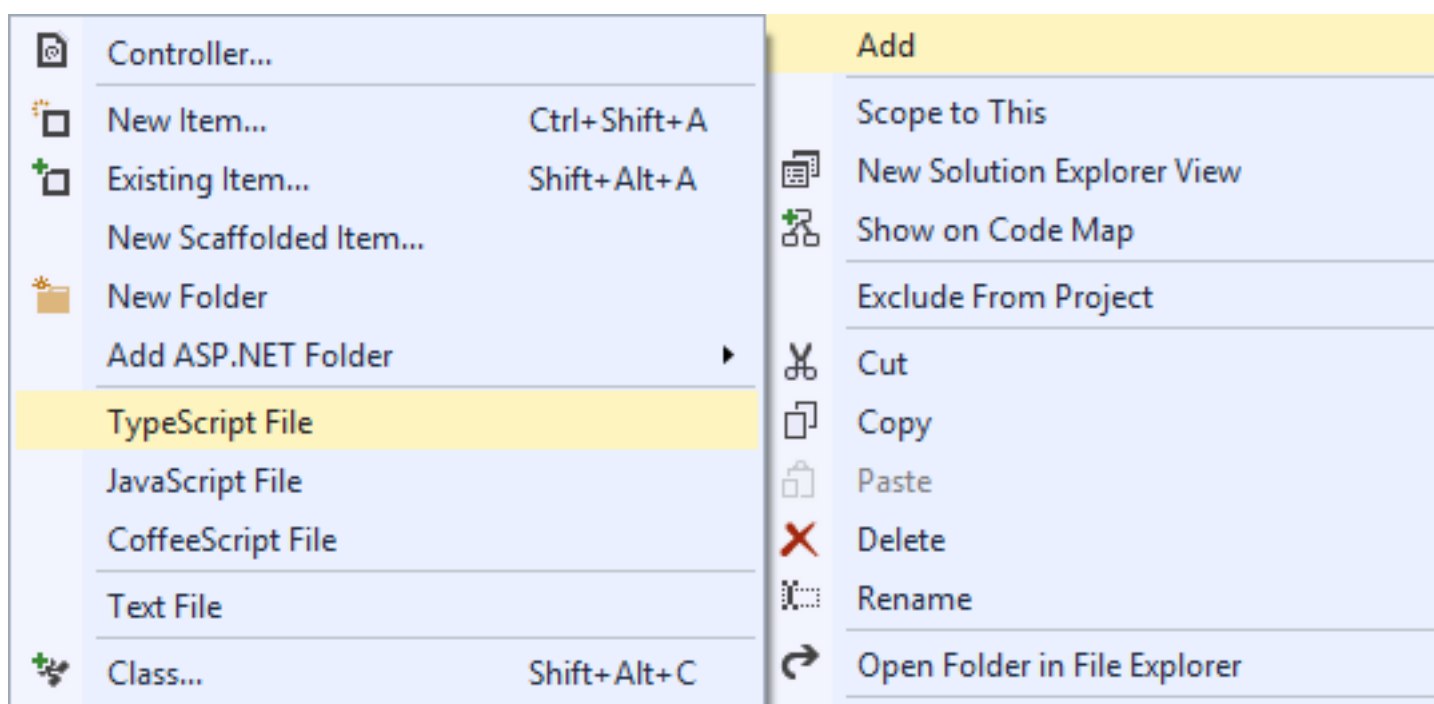
File Edit View Favorites Tools Help

Title from parent controller

Title from parent controller

نکته: دسترسی به کنترلر child در کنترلر parent امکان پذیر نیست.

پیشتر با ویژگی ها و نحوه کد نویسی این زبان آشنا شدید. از طرفی دیگر، نحوه تعریف کنترلرها در Angular نیز آموزش داده شد. در این پست قصد داریم طی یک مثال ساده با استفاده از زبان Typescript یک کنترلر Angular را ایجاد و سپس از آن در یک پروژه Asp.Net MVC استفاده نمایم. از آن جا که به صورت پیش فرض در VS.Net امکانات TypeScript نصب نشده است، برای شروع ابتدا TypeScript را از اینجا دانلود نمایید. بعد از نصب یک پروژه Asp.Net MVC ایجاد نمایید و سپس با استفاده از nuget فایل‌های مربوط به AngularJs را نصب نمایید. در این پست به تفصیل این مورد بررسی شده است (عملیات BundleConfig فایل‌های مورد نیاز به عهده خودتان). در پوشه scripts یک فولدر به نام app ساخته، سپس یک فایل TypeScript به نام ProductController.ts ایجاد کنید. (بعد از نصب TypeScript گزینه TypeScript File مشاهده خواهد شد)



در فایل ProductController.ts کدهای زیر را کپی نمایید:

```
module Product {
    export interface Scope {
        message: string;
    }

    export class Controller {
        constructor($scope: Scope) {
            $scope.message = "Hello from Masoud";
        }
    }
}
```

توضیح کدها بالا :

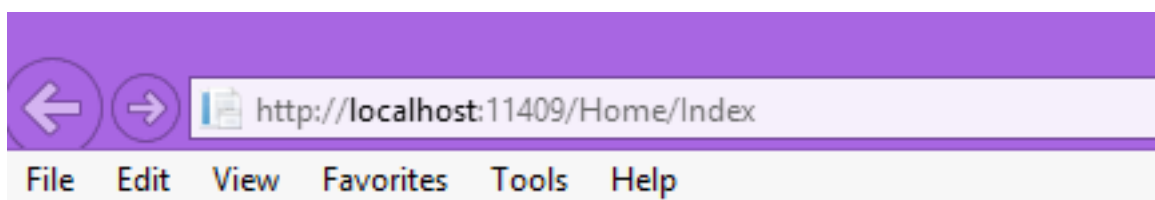
ابتدا یک ماژول به نام Product ایجاد می‌کنیم. سپس یک اینترفیس برای ایجاد Scope که جهت مقید سازی عناصر DOM به آبجکت‌های کنترلر مورد استفاده قرار می‌گیرد، ایجاد می‌کنیم. در داخل این اینترفیس متغیری به نام message از نوع string داریم. قصد داریم این متغیر را به یک عنصر مقید کنیم. حال یک کلاس به نام کنترلر ایجاد می‌کنیم که در تابع سازنده آن تزریق

وابستگی برای `$scope` از نوع اینترفیس `Scope` تعیین شده است. در نتیجه در بدنه سازنده می‌توانیم به متغیر `message` مقدار مورد نظر را نسبت دهیم.

کلمه کلیدی `export` برای تعریف عمومی کلاس استفاده شده است. یک `View` ایجاد و کدهای زیر را در آن کپی کنید:

```
<script type="text/javascript" src="~/scripts/app/ProductController.js"></script>
<div ng-app>
  <div ng-controller="Product.Controller">
    <p>{{message}}</p>
  </div>
</div>
```

اولین نکته در تگ `script` است که فراخوانی فایل `Typescript` باید با پسوند `.js` انجام گیرد. به دلیل اینکه فایل‌های `Typescript` بعد از کامپایل تبدیل به فایل‌های `JavaScript` خواهند شد؛ در نتیجه پسوند آن نیز `.js` است. دومین نکته در فراخوانی کنترلر مورد نظر است که از ترکیب نام ماژول و نام کلاس است. بعد از اجرای پروژه خروجی به صورت زیر خواهد بود:



نظرات خوانندگان

نویسنده: ایلیا اکبری فرد
تاریخ: ۱۵:۲۶ ۱۳۹۲/۱۰/۲۵

با سلام.

در کنترلر ، چگونه watch\$ مربوط به شی scope\$ را بوسیله TypeScript میتوان فراخوانی کرد؟

نویسنده: مسعود پاکدل
تاریخ: ۱۳:۷ ۱۳۹۲/۱۰/۲۷

ابتدا کنترلر خود را به صورت زیر تعریف کنید:

```
class MyController {
    thescope: any;
    static $inject = ['$scope'];

    constructor($scope) {
        this.thescope = $scope;

        this.thescope.$watch('watchtext', function(newValue, oldValue) {
            this.thescope.counter = scope.counter + 1;
            this.thescope.lastvalue = oldValue;
            this.thescope.currentvalue = newValue;});
    }
}
```

حال برای استفاده از کنترلر بالا به صورت زیر عمل نمایید:

```
module myApp.ctrl1 {
    myApp.controller("MyController", function($scope) {
        return new MyController($scope);
    })
}
```

در پست‌های قبلی با [AngularJS](#)، [TypeScript](#) و [Web Api](#) آشنا شدید. در این پست قصد دارم از ترکیب این موارد برای پیاده سازی عملیات واکشی اطلاعات سرویس Web Api در قالب یک پروژه استفاده نمایم. برای شروع ابتدا یک پروژه Asp.Net MVC ایجاد کنید.

در قسمت مدل ابتدا یک کلاس پایه برای مدل ایجاد خواهیم کرد:

```
public abstract class Entity
{
    public Guid Id { get; set; }
}
```

حال کلاسی به نام Book ایجاد می‌کنیم:

```
public class Book : EntityBase
{
    public string Name { get; set; }
    public decimal Author { get; set; }
}
```

در پوشه مدل یک کلاسی به نام BookRepository ایجاد کنید و کدهای زیر را در آن کپی نمایید (به جای پیاده سازی بر روی بانک اطلاعاتی، عملیات بر روی لیست درون حافظه انجام می‌گیرد):

```
public class BookRepository
{
    private readonly ConcurrentDictionary<Guid, Book> result = new ConcurrentDictionary<Guid, Book>();

    public IQueryable<Book> GetAll()
    {
        return result.Values.AsQueryable();
    }

    public Book Add(Book entity)
    {
        if (entity.Id == Guid.Empty) entity.Id = Guid.NewGuid();
        if (result.ContainsKey(entity.Id)) return null;
        if (!result.TryAdd(entity.Id, entity)) return null;
        return entity;
    }
}
```

نوبت به کلاس کنترلر می‌رسد. یک کنترلر Api به نام BooksController ایجاد کنید و سپس کدهای زیر را در آن کپی نمایید:

```
public class BooksController : ApiController
{
    public static BookRepository repository = new BookRepository();

    public BooksController()
    {
        repository.Add(new Book
        {
            Id=Guid.NewGuid(),
            Name="C#",
            Author="Masoud Pakdel"
        });
        repository.Add(new Book
```

```

    {
        Id = Guid.NewGuid(),
        Name = "F#",
        Author = "Masoud Pakdel"
    });

    repository.Add(new Book
    {
        Id = Guid.NewGuid(),
        Name = "TypeScript",
        Author = "Masoud Pakdel"
    });
}

public IEnumerable<Book> Get()
{
    return repository.GetAll().ToArray();
}
}

```

در این کنترلر، اکشنی به نام Get داریم که در آن اطلاعات کتاب‌ها از Repository مربوطه برگشت داده خواهد شد. در سازنده این کنترلر ابتدا سه کتاب به صورت پیش فرض اضافه می‌شود و انتظار داریم که بعد از اجرای برنامه، لیست مورد نظر را مشاهده نماییم.

حال نوبت به عملیات سمت کلاینت می‌رسد. برای استفاده از قابلیت‌های TypeScript و AngularJs در Vs.Net از این [مقاله](#) کمک بگیرید. بعد از آماده سازی در فولدر script، پوشه ای به نام app می‌سازیم و یک فایل TypeScript به نام BookModel در آن ایجاد می‌کنیم:

```

module Model {
    export class Book{
        Id: string;
        Name: string;
        Author: string;
    }
}

```

واضح است که ماژولی به نام Model داریم که در آن کلاسی به نام Book ایجاد شده است. برای انتقال اطلاعات از طریق سرویس \$http در Angular نیاز به سریالایز کردن این کلاس به فرمت Json خواهیم داشت. قصد داریم View مورد نظر را به صورت زیر ایجاد نماییم:

```

<div ng-controller="Books.Controller">
    <table class="table table-striped table-hover" style="width: 500px;">
        <thead>
            <tr>
                <th>Name</th>
                <th>Author</th>
            </tr>
        </thead>
        <tbody>
            <tr ng-repeat="book in books">
                <td>{{book.Name}}</td>
                <td>{{book.Author}}</td>
            </tr>
        </tbody>
    </table>
</div>

```

توضیح کدهای بالا:

ابتدا یک کنترلری که به نام Controller که در ماژولی به نام Book تعریف شده است باید ایجاد شود. اطلاعات تمام کتب ثبت شده باید از سرویس مورد نظر دریافت و با یک ng-repeat در جدول نمایش داده خواهند شد. در پوشه app یک فایل TypeScript دیگر برای تعریف برخی نیازمندی‌ها به نام AngularModule ایجاد می‌کنیم که کد آن به صورت زیر خواهد بود:

```
declare module AngularModule {
  export interface HttpPromise {
    success(callback: Function) : HttpPromise;
  }
  export interface Http {
    get(url: string): HttpPromise;
  }
}
```

در این ماژول دو اینترفیس تعریف شده است. اولی به نام `HttpPromise` است که تابعی به نام `success` دارد. این تابع باید بعد از موفقیت آمیز بودن عملیات فراخوانی شود. ورودی آن از نوع `Function` است. یعنی اجازه تعریف یک تابع را به عنوان ورودی برای این توابع دارید.

در اینترفیس `Http` نیز تابعی به نام `get` تعریف شده است که برای دریافت اطلاعات از سرویس `api`، مورد استفاده قرار خواهد گرفت. از آن جا که تعریف توابع در اینترفیس فاقد بدنه است در نتیجه این جا فقط امضای توابع مشخص خواهد شد. پیاده سازی توابع به عهده کنترلرها خواهد بود:

مرحله بعد مربوط است به تعریف کنترلری به نام `BookController` تا اینترفیس بالا را پیاده سازی نماید. کدهای آن به صورت زیر خواهد بود:

```
/// <reference path='AngularModule.ts' />
/// <reference path='BookModel.ts' />

module Books {
  export interface Scope {
    books: Model.Book[];
  }

  export class Controller {
    private httpService: any;

    constructor($scope: Scope, $http: any) {
      this.httpService = $http;

      this.getAllBooks(function (data) {
        $scope.books = data;
      });
      var controller = this;
    }

    getAllBooks(successCallback: Function): void {
      this.httpService.get('/api/books').success(function (data, status) {
        successCallback(data);
      });
    }
  }
}
```

توضیح کدهای بالا:

برای دسترسی به تعاریف انجام شده در سایر ماژولها باید ارجاعی به فایل تعاریف ماژولهای مورد نظر داشته باشیم. در غیر این صورت هنگام استفاده از این ماژولها با خطای کامپایلری روبرو خواهیم شد. عملیات ارجاع به صورت زیر است:

```
/// <reference path='AngularModule.ts' />
/// <reference path='BookModel.ts' />
```

در [پست قبلی](#) توضیح داده شد که برای مقید سازی عناصر بهتر است یک اینترفیس به نام `Scope` تعریف کنیم تا بتوانیم متغیرهای مورد نظر برای مقید سازی را در آن تعریف نماییم در این جا تعریف آن به صورت زیر است:

```
export interface Scope {
  books: Model.Book[];
}
```


در این جا فقط نیاز به لیستی از کتاب‌ها داریم تا بتوان در جدول مورد نظر در View آنرا پیمایش کرد. تابعی به نام `getAllBooks` در کنترلر مورد نظر نوشته شده است که ورودی آن یک تابع خواهد بود که باید بعد از واکشی اطلاعات از سرویس، فراخوانی شود. اگر به کدهای بالا دقت کنید می‌بینید که در ابتدا سازنده کنترلر، سرویس `$http` موجود در Angular به متغیری به نام `httpService` نسبت داده می‌شود. با فراخوانی تابع `get` و ارسال آدرس سرویس که با توجه به مقدار مسیر یابی پیش فرض کلاس `WebApiConfig` باید با `api` شروع شود به راحتی اطلاعات مورد نظر به دست خواهد آمد. بعد از واکشی در صورت موفقیت آمیز بودن عملیات تابع `success` اجرا می‌شود که نتیجه آن انتساب مقدار به دست آمده به متغیر `books` تعریف شده در `$scope` می‌باشد.

در نهایت خروجی به صورت زیر خواهد بود:

File Edit View Favorites Tools Help					
Name			Author		
C#			Masoud Pakdel		
TypeScript			Masoud Pakdel		
F#			Masoud Pakdel		

[سورس پیاده سازی مثال بالا در Visual Studio 2013](#)

نظرات خوانندگان

نویسنده: sadegh hp

تاریخ: ۱۱:۳۳ ۱۳۹۲/۱۲/۲۳

چجوری میشه با jasmine یک تست برای متدی که http.post\$ رو در یک سرویس انگولار پیاده کرده نوشت؟ تست متدهای async در انگولار چجوریه ؟

نویسنده: مسعود پاکدل

تاریخ: ۱۳:۱ ۱۳۹۲/۱۲/۲۳

angularJs کتابخانه ای برای mock آجکت ها خود تهیه کرده است.(angular-mock). از آن جا که در angular مبحث تزریق وابستگی بسیار زیبا پیاده سازی شده است با استفاده از این کتابخانه می توانید آجکت های متناظر را mock کنید. برای مثال:

```
describe('myApp', function() {
  var scope;

  beforeEach(angular.mock.module('myApp'));
  beforeEach(angular.mock.inject(function($rootScope) {
    scope = $rootScope.$new();
  }));
  it('...')
});
```

هم چنین برای تست سرویس \$http و شبیه سازی عملیات request و response در انگولار سرویس \$httpBackend تعبیه شده است که یک پیاده سازی Fake از \$http است که در تست ها می توان از آن استفاده کرد. برای مثال:

```
describe('Remote tests', function() {
  var $httpBackend, $rootScope, myService;
  beforeEach(inject(
function(_$httpBackend_, _$rootScope_, _myService_) {
  $httpBackend = _$httpBackend_;
  $rootScope = _$rootScope_;
  myService = _myService_;
}));
it('should make a request to the backend', function() {
  $httpBackend.expect('GET', '/v1/api/current_user')
    .respond(200, {userId: 123});
  myService.getCurrentUser();

  $httpBackend.flush();
});
});
```

دستور httpBackend\$.expect برای ایجاد درخواست مورد نظر استفاده می شود که نوع verb را به عنوان آرگومان اول دریافت می کند. respond نیز مقدار بازگشتی مورد انتظار از سرویس مورد نظر را برگرداند. می توانید از دستورات زیر برای سایر حالات استفاده کنید:

```
httpBackend$.expectGet«
httpBackend$.expectPut«
httpBackend$.expectPost«
httpBackend$.expectDelete«
httpBackend$.expectJson«
httpBackend$.expectHead«
httpBackend$.expectPatch«
```

Flush کردن سرویس \$httpBackend در پایان تست نیز برای همین مبحث async اجرا شدن سرویس های http\$backend است.

نویسنده: صادق اچ پی
تاریخ: ۱۳۹۲/۱۲/۲۵ ۹:۴۸

ممنون از پاسخ شما.

اما سوال بعد اینکه چرا اصلا باید بیرون از سرویس http رو ساخت؟ فرض کنید که ما دسترسی به محتوی متود درون سرویس نداریم و فقط می‌خواهیم اون رو صدا کنیم و ببینیم که متود درون سرویس درست کار میکنه یا نه! بدون اینکه بدونیم چجوری داخل متود پیاده سازی شده که در این مورد یک http.post یا get هست.

نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۱۲/۲۵ ۱۰:۴۳

\$httpBackend یک پیاده سازی fake از \$http است، در نتیجه می‌توانید در هنگام تست، این سرویس را به کنترلرهای خود تزریق کنید. اما قبل از DI باید برای این سرویس مشخص شود که برای مثال در هنگام مواجه شدن با یک درخواست از نوع Get و آدرس X چه خروجی برگشت داده شود. درست شبیه به رفتار mocking framework ها. فرض کنید شما کنترلری به شکل زیر دارید:

```
(function (module) {
    var myController = function ($scope, $http) {
        $http.get("/api/myData")
            .then(function (result) {
                $scope.data= result.data;
            });
    };
    module.controller("MyController",
        ["$scope", "$http", myController]);
})(angular.module("myApp"));
```

همان طور که می‌بینید در این کنترلر از \$http استفاده شده است. حال برای تست آن می‌توان نوشت:

```
describe("myApp", function () {
    beforeEach(module('myApp'));
    describe("MyController", function () {
        var scope, httpBackend;
        beforeEach(inject(function ($rootScope, $controller, $httpBackend, $http) {
            scope = $rootScope.$new();
            httpBackend = $httpBackend;
            httpBackend.when("GET", "/api/myData").respond([{}], {}, {});
            $controller('MyController', {
                $scope: scope,
                $http: $http
            });
        }));
        it("should have 3 row", function () {
            httpBackend.flush();
            expect(scope.data.length).toBe(3);
        });
    });
});
```

httpBackend ساخته شده با استفاده از سرویس \$controller به کنترلر مورد نظر تزریق می‌شود. حال اگر در یک کنترلر 5 بار از سرویس \$http برای فراخوانی 5 resource متفاوت استفاده شده باشد باید برای هر حالت \$httpBackend را طوری تنظیم کرد که بداند برای هر منبع چه خروجی در اختیار کنترلر قرار دهد.

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.

در حین انجام اعمال غیرهمزمان جاوا اسکریپتی مانند فراخوانی‌های jQuery AJAX، برای مدیریت دریافت نتایج، عموماً از یک سری callback استفاده می‌شود. برای مثال:

```
$.get('http://site-url', function(data) {
// این تابع پس از پایان کار عملیات ای‌جکسی در آینده فراخوانی خواهد شد
});
```

تا اینجا مشکلی به نظر نمی‌رسد. اما مورد ذیل چطور؟

```
$.get('http://site-url/0', function(data0) {
// callback #1
$.get('http://site-url/1', function(data1) {
// callback #2
$.post('http://site-url/2', function(data2) {
// callback #3
});
});
});
```

در اینجا نیاز است پس از پایان کار عملیات Ajax ایی اول، عملیات دوم و پس از آن عملیات سومی انجام شود. همانطور که مشاهده می‌کنید، این نوع کدها به سرعت از کنترل خارج می‌شوند؛ خوانایی پایینی داشته و مدیریت استثناءهای رخ داده در آن‌ها نیز در این بین مشکل است. از این جهت که خطاهای هر کدام به سطحی بالاتر منتقل نمی‌شود و باید همانجا محلی و داخل هر callback مدیریت گردد.

روش‌های زیادی برای حل این مساله ارائه شده‌است و در حال حاضر کار کردن با promiseها متداول‌ترین روش حل مدیریت فراخوانی کدهای همزمان جاوا اسکریپتی است. برای نمونه اگر از AngularJS استفاده کنید، سرویس‌های آن برای دریافت اطلاعات از سرور، از یک چنین مفهومی استفاده می‌کنند.

Promise در جاوا اسکریپت چیست؟

شیء Promise، نمایانگر قراردادی است که در آینده می‌تواند مورد قبول واقع شود، یا رد گردد. بررسی این قرارداد، تنها یکبار می‌تواند رخ دهد (پذیرش یا رد آن). هنگامیکه این بررسی صورت گرفت (رد یا پذیرش آن و نه هردو)، یک callback برای اطلاع رسانی فراخوانی می‌گردد. سپس این callback می‌تواند یک Promise دیگر را سبب شود. به این ترتیب می‌توان Promiseها را زنجیر وار به یکدیگر متصل کرد. برای نمونه jQuery به صورت توکار از promises پشتیبانی می‌کند:

```
// returns a promise
$.get('http://site-url/0')
.then(function(data) {
// callback 1
// returns a promise
return $.get('http://site-url/1');
})
.then(function(data) {
// callback 2
// returns a promise
return $.post('http://site-url/2');
})
.then(function(data) {
// callback 3
});
```

متد get در jQuery یک شیء promise را بازگشت می‌دهد. در ادامه می‌توان این نتیجه را توسط متد then، زنجیروار ادامه داد. متدی که به عنوان پارامتر به then ارسال می‌شود، یک callback بوده و پس از پایان کار promise قبلی رخ می‌دهد. آنگه‌مانی که به

این callback ارسال می‌شود، نتیجه‌ی promise قبلی است. در حین اعمال jQuery Ajax، این callback تنها زمانی فراخوانی می‌شود که عملیات قبلی موفقیت آمیز بوده باشد و data ارائه شده، اطلاعاتی است که توسط response دریافتی از سرور، دریافت گردیده‌است.

در این حالت، هر callback حداقل سه کار را می‌تواند انجام دهد:

(الف) یک promise دیگر را بازگشت دهد. نمونه آن را با `return $.get` در کدهای فوق ملاحظه می‌کنید.

(ب) خاتمه عادی. همینجا کار promise با مقدار بازگشت داده شده، پایان می‌یابد.

(ج) صدور یک استثناء. سبب برگشت خوردن و عدم پذیرش promise می‌شود.

استفاده از Promises در سایر کتابخانه‌ها

jQuery پیاده سازی توکاری از promises دارد؛ اما سایر کتابخانه‌ها، مانند AngularJS ایی که مثال زده شده چطور عمل می‌کنند؟ استاندارد به نام [Promises/A+](#) جهت یک دست سازی پیاده سازی‌های promise در جاوا اسکریپت پیشنهاد شده‌است. jQuery نیمی از آن را پیاده سازی کرده‌است؛ اما کتابخانه‌ی دیگری به نام [Q Library](#)، پیاده سازی نسبتاً مفصل‌تری را از این استاندارد ارائه می‌دهد. فریم ورک AngularJS نیز در پشت صحنه از همین کتابخانه برای پیاده سازی promises استفاده می‌کند.

آشنایی با کتابخانه Q

استفاده مقدماتی از Q همانند مثالی است که از jQuery ملاحظه کردید.

```
Q.fcall(callback1)
  .then(callback2);
```

اشیاء promise بازگشت داده شده توسط jQuery نیز توسط کتابخانه Q مورد پذیرش واقع می‌شوند:

```
Q.fcall(function() {
  return $.get('http://my-url');
})
  .then(callback3);
```

علاوه بر این‌ها مفهومی به نام deferred objects نیز در کتابخانه‌ی Q پیاده سازی شده‌است:

```
function waitForClick() {
  var deferred = Q.defer();

  $('#okButton').click(function() {
    deferred.resolve();
  });

  $('#cancelButton').click(function() {
    deferred.reject();
  });

  return deferred.promise;
}

Q.fcall(waitForClick)
  .then(function() {
    // ok button was clicked
  }, function() {
    // cancel button was clicked
  });
```

توسط deferred objects می‌توان بررسی یک promise را به تاخیر انداخت. در مثال فوق، اولین callback فراخوانی شده به نام `waitForClick`، از اشیاء به تاخیر افتاده استفاده می‌کند. ابتدا توسط فراخوانی متد `Q.defer`، یک deferred object ایجاد می‌شود. در این بین اگر کاربر بر روی دکمه‌ی OK کلیک کرد، با فراخوانی `deferred.resolve`، این promise مورد پذیرش واقع خواهد شد و یا اگر کاربر بر روی دکمه‌ی cancel کلیک کند، با فراخوانی متد `deferred.reject`، این promise رد می‌گردد. نهایتاً شیء promise

توسط deferred.promise بازگشت داده خواهد شد.

در ادامه کار، اینبار متد then، دو callback را قبول می‌کند. Callback اول پس از پذیرش قرار داد و Callback دوم پس از رد قرار داد، فراخوانی خواهد گردید.
در رنجیره تعریف شده، اگر معادلی برای reject در نظر گرفته نشده باشد، مانند مثال ذیل:

```
Q.fcall(myFunction1)
  .then(success1)
  .then(success2, failure1);
```

Q به دنبال نزدیک‌ترین متد callback گزارش خطای کار خواهد گشت. در این حالت متد failure1 در صورت شکست اولین promise فراخوانی خواهد شد.

همچنین اگر نتیجه‌ی success1 با شکست مواجه شود نیز failure1 فراخوانی می‌گردد. اما باید در نظر داشت که شکست success2، توسط failure1 مدیریت نمی‌شود.

AngularJS در Promises

در AngularJS امکانات کتابخانه Q توسط پارامتری به نام \$q در اختیار سرویس‌های برنامه قرار می‌گیرد (تزریق می‌شود):

```
var app = angular.module("myApp", []);
app.factory('dataSvc', function($http, $q){
  var basePath="api/books";
  getAllBooks = function(){
    var deferred = $q.defer();
    $http.get(basePath).success(function(data){
      deferred.resolve(data);
    }).error(function(err){
      deferred.reject("service failed!");
    });
    return deferred.promise;
  };

  return{
    getAllBooks:getAllBooks
  };
});

app.controller('HomeController', function($scope, $window, dataSvc){
  function initialize(){
    dataSvc.getAllBooks().then(function(data){
      $scope.books = data;
    }, function(msg){
      $window.alert(msg);
    });
  }

  initialize();
});
```

در اینجا اگر دقت کنید، مباحث و عملکرد آن دقیقاً مانند قبل است. ابتدا یک deferred object با فراخوانی متد q.defer ایجاد شده است. سپس با استفاده از امکانات توکار http آن (بجای استفاده از jQuery Ajax)، کار فراخوانی یک restful service صورت گرفته است (مثلاً فراخوانی یک ASP.NET Web API). در صورت موفقیت کار، متد deferred.resolve و در صورت عدم موفقیت، متد deferred.reject فراخوانی شده‌است. نهایتاً این سرویس، یک deferred.promise را بازگشت می‌دهد.
اکنون در کنترلی که قرار است از این سرویس استفاده کند، متد then کتابخانه Q را ملاحظه می‌کنید که دو Callback متناظر resolve و reject مدیریت promise بازگشت داده شده را به همراه دارد. اگر عملیات Ajaxی موفقیت آمیز باشد، شیء books را مقدار دهی می‌کند و اگر خیر، پیامی را به کاربر نمایش خواهد داد.

پشتیبانی مرورگرهای جدید از استاندارد Promise

در حال حاضر کروم 32 و نگارش‌های شبانه فایرفاکس، Promise را که جزئی از استاندارد JavaScript شده‌است، به صورت توکار

و بدون نیاز به کتابخانه‌های جانبی، پشتیبانی می‌کنند.

```
if (window.Promise) { // Check if the browser supports Promises
  var promise = new Promise(function(resolve, reject) {
    //asynchronous code goes here
  });
}
```

در اینجا با فراخوانی window.Promise مشخص می‌شود که آیا مرورگر جاری از Promises پشتیبانی می‌کند یا خیر. سپس یک شیء promise ایجاد شده و این شیء توسط پارامترهای resolve و reject که هر دو تابع می‌باشند، کار مدیریت کدهای غیرهمزمان را انجام می‌دهد:

```
if (window.Promise) {
  console.log('Promise found');

  var promise = new Promise(function(resolve, reject) {
    // async
    if (result) {
      resolve(data);
    } else {
      reject('error');
    }
  });

  promise.then(function(data) {
    console.log('Promise fulfilled.');
```

```
  }, function(error) {
    console.log('Promise rejected.');
```

```
  });
} else {
  console.log('Promise not available');
```

```
}
```

در مثال فوق ابتدا یک شیء Promise ایجاد شده است. این شیء استاندارد بوده و با کروم 32 قابل آزمایش است. سپس در callback ابتدایی آن می‌توان یک عملیات AJAX ایی را انجام داد. اگر نتیجه‌ی آن موفقیت آمیز بود، تنها کافی است پارامتر اول این callback را فراخوانی کنیم و اگر خیر، پارامتر دوم آن را. برای استفاده از این شیء Promise ایجاد شده، می‌توان از متد then استفاده کرد. این متد نیز در اینجا دو callback پذیرش و رد promise را می‌تواند دریافت کند. برای زنجیر کردن آن کافی است متد then، یک Promise دیگر را بازگشت دهد و از نتیجه‌ی آن در then بعدی استفاده گردد.

در [پست‌های قبلی](#) با مفهوم ng-app آشنا شدید. دایرکتیو ng-app برای استفاده از راه انداز خودکار فریم ورک Angular (معروف به auto-bootstrap) استفاده می‌شود. در حالت پیش فرض، به ازای هر سند Html فقط می‌توان یک ماژول در Angular تعریف کرد. در سند مربوطه اولین المانی که دارای دایرکتیو ng-app باشد به عنوان عنصر ریشه در نظر گرفته می‌شود و تمام عناصر تعریف شده در محدوده این دایرکتیو قابل استفاده برای ماژول مورد نظر خواهد بود. سایر عناصر حتی اگر ng-app یکسان داشته باشند نادیده گرفته می‌شوند.

ابتدا یک مثال زیر را به روش auto-bootstrap بررسی می‌کنیم:

```
<div ng-app="myApp">
  <div ng-controller="myController as ctrl">
    <span>ng-app #1</span> {{ctrl.firstName}} {{ctrl.lastName}}
  </div>
</div>

<div ng-app="myApp">
  <div ng-controller="myController as ctrl">
    <span>ng-app #2</span> {{ctrl.firstName}} {{ctrl.lastName}}
  </div>
</div>

@section scripts
{
  <script type="text/javascript" src="~/scripts/Modules/module8.js"></script>
}
```

در کنترلر مورد نظر نیز تعاریف به صورت زیر خواهد بود:

```
var app = angular.module('myApp', []);
app.controller('myController', function ()
{
  this.firstName = "Masoud";
  this.lastName = "Pakdel";
});
```

در مثال بالا دو تگ div وجود دارد که به صورت مشترک با استفاده از دایرکتیو ng-app به یک ماژول اشاره می‌کنند. طبق گفته‌ها بالا در روش auto-bootstrap اولین عنصری که دارای دایرکتیو ng-app باشد به عنوان محدوده ماژول مورد استفاده قرار خواهد گرفت در نتیجه سایر المان‌ها (در اینجا منظور تگ div دوم است) نادیده گرفته خواهند شد. پس خروجی به صورت زیر می‌شود:

File Edit View Favorites Tools Help

ng-app #1 Masoud Pakdel
ng-app #2 {{ctrl.firstName}} {{ctrl.lastName}}

اما اگر قصد داشته باشیم که در یک سند html دو نقطه شروع تعریف کنیم در حالی که هر کدام از یک منبع داده استفاده نمایند باید bootstrap برنامه را به صورت دستی تعیین کرد. برای این کار کافست از دستور angular.bootstrap به صورت زیر استفاده نماییم:

پیاده سازی مثال بالا

```
<div id="myAppContainer1">
  <div ng-controller="myController as ctrl">
    <span>ng-app #1</span> {{ctrl.firstName}} {{ctrl.lastName}}
  </div>
</div>

<div id="myAppContainer2">
  <div ng-controller="myController as ctrl">
    <span>ng-app #2</span> {{ctrl.firstName}} {{ctrl.lastName}}
  </div>
</div>

@section scripts
{
  <script type="text/javascript" src="~/scripts/Modules/module8.js"></script>
}
```

اولین تغییر مورد نظر این است که، دایرکتیو ng-app حذف شد و به جای آن id برای تگ div تعیین کردیم. در فایل کنترلر مورد نظر نیز تغییر زیر را اعمال می‌کنیم:

```
var app = angular.module('myApp', []);
app.controller('myController', function ()
{
  this.firstName = "Masoud";
  this.lastName = "Pakdel";
});
angular.bootstrap(document.getElementById("myAppContainer1"), ["myApp"]);
angular.bootstrap(document.getElementById("myAppContainer2"), ["myApp"]);
```

با استفاده از دستور angular.bootstrap می‌توان بر اساس id تعیین شده تگ مورد نظر در سند را به دست آورد و ماژول مورد نظر را به آن نسبت داد.

خروجی مثال بالا:



برخلاف حالت قبل هر دو نقطه شروع به یک منبع داده اشاره می‌کنند و محدودیت حالت قبل برطرف می‌شود.

نظرات خوانندگان

نویسنده: ابوالفضل رجب پور
تاریخ: ۱۹:۴ ۱۳۹۲/۱۰/۱۷

سلام و تشکر
بنظرم جای نمونه برنامه‌های کوچکی که گام به گام توضیح داده بشن خالیه. خیلی خوب و کاربردی مفهوم جا می‌افته. مثلاً نمونه‌ی
معروف todo

نویسنده: مسعود پاکدل
تاریخ: ۲۳:۳ ۱۳۹۲/۱۰/۱۷

مثال بالا نمونه ای از یک برنامه کوچک است. بهتر است هر نکته و سرنخ در طی یک پست ارائه شود به جای ذکر تاریخچه،
هستی و چیستی مسائل در طی یک پست.

می‌خواهیم یک مثال ساده از دریافت اطلاعات از سرور و نمایش آن در یک View را توسط AngularJS، با هم بررسی کنیم. همانطور که می‌دانید برای نمایش تعدادی از اشیاء در انگولار می‌توان به این صورت نیز عمل کرد:

```
<div ng-init="products=[
  {id:1,name:'product1',price:25000,description:'description of product'},
  {id:2,name:'product2',price:5000,description:'description of product'},
  {id:3,name:'product3',price:5000,description:'description of product'},
  {id:4,name:'product4',price:2000,description:'description of product'},
  {id:5,name:'product5',price:255000,description:'description of product'}
]">
<div>
  <div>
    <table>
      <tr>
        <th>Id</th>
        <th>Product Name</th>
        <th>Price</th>
        <th>Description</th>
      </tr>
      <tr ng-repeat="product in products">
        <td>{{product.id}}</td>
        <td>{{product.name}}</td>
        <td>{{product.price}}</td>
        <td>{{product.description}}</td>
      </tr>
    </table>
  </div>
</div>
</div>
```

در کد فوق توسط ویژگی ng-init می‌توانیم داده‌هایمان را Initialize کنیم و در نهایت توسط ویژگی ng-repeat می‌توانیم داده‌هایمان را در صفحه نمایش دهیم. این ویژگی دقیقاً مانند یک حلقه foreach عمل میکند؛ مثلاً معادل آن در Razor سمت سرور، به این صورت است:

```
@foreach (var product in Model.products)
{
  <td>product.id</td>
  <td>product.name</td>
  <td>product.price</td>
  <td>product.description</td>
}
```

خوب؛ حالا می‌خواهیم این اطلاعات را از سمت سرور بخوانیم و به صورت فوق نمایش دهیم. ابتدا مدل مان را به این صورت تعریف می‌کنیم:

```
namespace AngularAndMvc.Models
{
  public class Product
  {
    public int Id { get; set; }
    public string Name { get; set; }
    public float Price { get; set; }
    public string Description { get; set; }
  }
}
```

سپس در داخل کنترلر زیر اطلاعات را به صورت in memory data تعریف می‌کنیم (جهت سهولت دموی کار) و به view مورد نظر پاس می‌دهیم. البته شما می‌توانید این اطلاعات را از دیتابیس بخوانید؛ روال کار فرقی نمی‌کند:

```
namespace AngularAndMvc.Controllers
```

```
{
    public class ProductController : Controller
    {
        public ActionResult Index()
        {
            return View("Index", "", GetSerializedProduct());
        }

        public string GetSerializedProduct()
        {
            var products = new[]
            {
                new Product{Id=1,Name="product1",Price=4500,Description="description of this product"},
                new Product{Id=2,Name="product2",Price=500,Description="description of this product"},
                new Product{Id=3,Name="product3",Price=400,Description="description of this product"},
                new Product{Id=4,Name="product4",Price=5500,Description="description of this product"},
                new Product{Id=5,Name="product5",Price=66500,Description="description of this product"}
            };
            var settings = new JsonSerializerSettings { ContractResolver=new
            CamelCasePropertyNamesContractResolver()};
            return JsonConvert.SerializeObject(products,Formatting.None,settings);
        }
    }
}
```

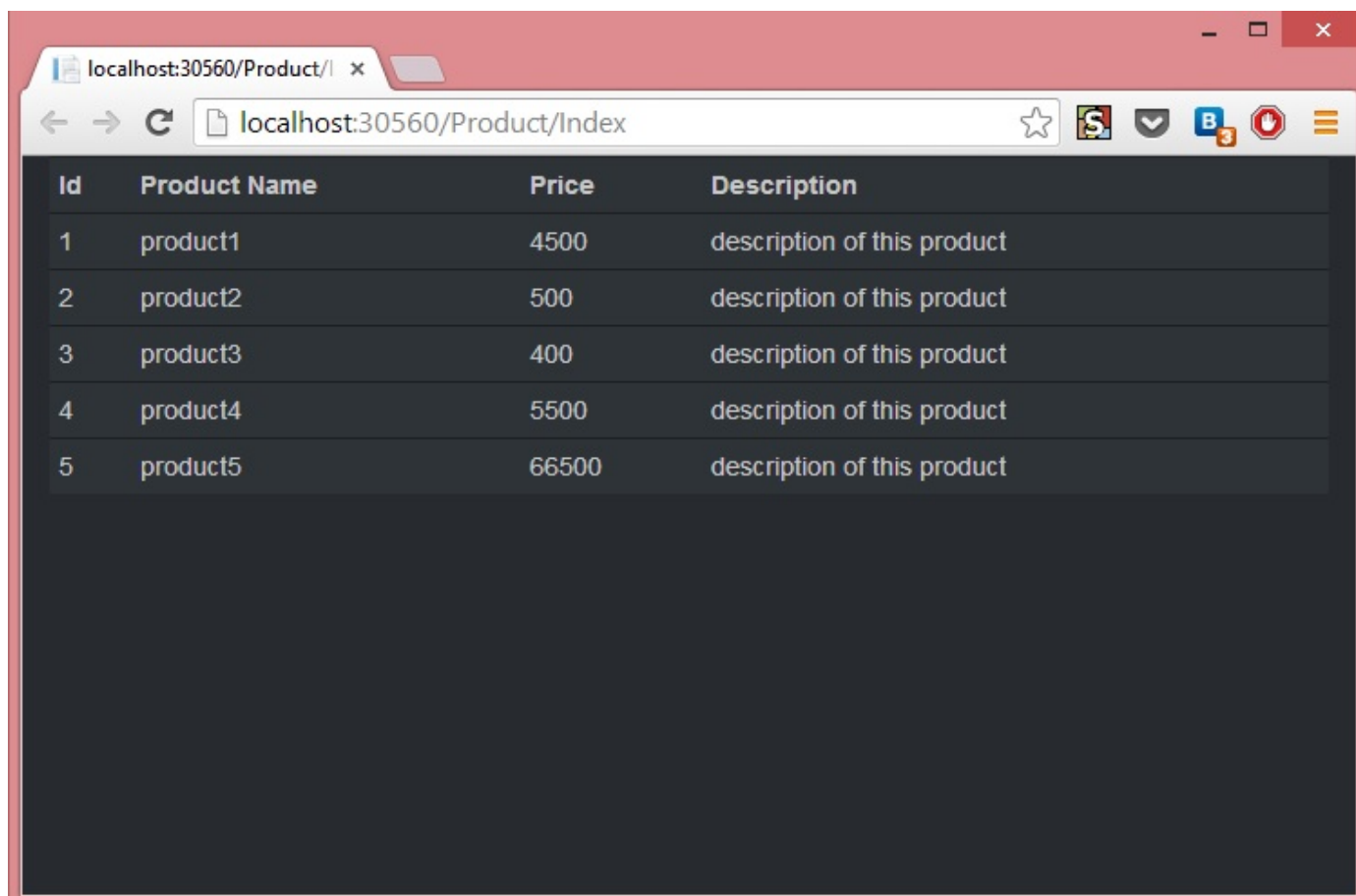
همانطور که در کد بالا مشخص است، اطلاعات را به صورت JSON به View مان پاس داده ایم و برای اینکه ابتدای نام مقادیر بازگشتی به صورت حروف بزرگ نباشند (به صورت خودکار تبدیل به camel case شوند) پارامتر settings را برای متد SerializeObject تعیین کرده ایم:

```
var settings = new JsonSerializerSettings { ContractResolver=new
CamelCasePropertyNamesContractResolver()};
return JsonConvert.SerializeObject(products,Formatting.None,settings);
```

view را نیز به این صورت تغییر می دهیم :

```
@model string
<div ng-init="products = @Model">
    <div>
        <div>
            <table>
                <tr>
                    <th>Id</th>
                    <th>Product Name</th>
                    <th>Price</th>
                    <th>Description</th>
                </tr>
                <tr ng-repeat="product in products">
                    <td>{{product.id}}</td>
                    <td>{{product.name}}</td>
                    <td>{{product.price}}</td>
                    <td>{{product.description}}</td>
                </tr>
            </table>
        </div>
    </div>
</div>
```

تنها تغییری که در کد فوق اعمال شده است، به جای اینکه ویژگی ng-init را به صورت inline مقادیردهی کنیم آن را از کنترلر دریافت کرده ایم. در خروجی هم اطلاعات به این صورت نمایش داده می شوند :



The screenshot shows a web browser window with the address bar displaying 'localhost:30560/Product/Index'. The browser's address bar also shows a tab titled 'localhost:30560/Product/'. The main content area of the browser displays a table with the following data:

Id	Product Name	Price	Description
1	product1	4500	description of this product
2	product2	500	description of this product
3	product3	400	description of this product
4	product4	5500	description of this product
5	product5	66500	description of this product

سورس مثال فوق را هم از [اینجا](#) می توانید دریافت کنید.

نظرات خوانندگان

نویسنده: حمید رضا منصوری
تاریخ: ۱۱:۱۵ ۱۳۹۳/۰۱/۱۵

با تشکر
من از این روش در یک view استفاده کردم (در اینجا هدف نمایش لیست کاربران بود) ولی مشکل من این هست که وقتی این view برای کاربر نمایش داده میشه چون انگولار اون رو کش میکنه با افزودن کاربر جدید این لیست تا درخواست مجدد از سرور بروز نمیشه.

میخواستم بدونم آیا با واکنشی اطلاعات توسط \$http مشکلم حل میشه؟
یا میشه یک view در انگولار کش نشود و به هر با route به اون view مورد نظر از سرور فراخوانی بشه

نویسنده: سیروان عقیقی
تاریخ: ۱۱:۵۲ ۱۳۹۳/۰۱/۱۵

البته میتونید کش رو سمت سرور با اعمال outputcache بر روی اکشن خودتون غیر فعال کنید:

```
[OutputCache(NoStore = true, Duration = 0, VaryByParam = "None")]
public ActionResult Index()
{
    return View("Index", "", GetSerializedProduct());
}
```

نویسنده: محسن درپرستی
تاریخ: ۱۰:۵۳ ۱۳۹۳/۰۱/۱۶

در روش بالا اگر چه اطلاعات از سرور دریافت میشود اما حالتی استاتیک دارد چون لیست محصولات در زمان رندر شدن صفحه تولید و در ng-init قرار میگیرد. و در نتیجه برای بروزکردن حتما باید صفحه مجددا درخواست بشود (رفرش). این روش برای لیست هایی که تغییراتی ندارند یا به ندرت تغییر می کنند مناسب است.
و بله یک راه برای حل این مشکل استفاده از سرویس http می تواند باشد.

با توجه به [پست ها منتشر شده قبلی](#) درباره AngularJS به احتمال قوی شما نیز به این نتیجه رسیده اید که این فریم ورک برای انواع پروژه ها به ویژه پروژه هایی با مقیاس بزرگ بسیار مناسب است. منظور از ساختار پروژه Angular این است که به چه سبکی فایل های پروژه را سازمان دهی کنیم طوری که در هنگام توسعه و تغییرات با مشکل مواجه نشویم. عموماً کدهای مربوط به بخش frontend پروژه دارای ساختار قوی نمی باشند در نتیجه developerها بیشتر سلیقه ای کدهای مربوطه را می نویسند که با گذر زمان این مورد باعث بروز مشکل در امر توسعه نرم افزار می شود (نمونه بارز آن کدهای نوشته شده JQuery در صفحات است). AngularJS نیز همانند سایر کتابخانه ها و فریم ورک های جاوااسکریپتی دیگر از این امر مستثنی نیست و فایل های آن باید طبق روشی مناسب پیاده سازی و مدیریت شوند. انتخاب ساختار و روش سازمان دهی فایل ها وابستگی مستقیم به مقیاس پروژه دارد. ساختار پروژه های کوچک می تواند کاملاً متفاوت با ساختار پروژه های بزرگ باشد. در این پست به بررسی چند روش در این زمینه خواهیم پرداخت.

پروژه های کوچک عموماً دارای ساختاری مشابه تصویر ذیل می باشند:

- css/
- img/
- js/
 - app.js
 - controllers.js
 - directives.js
 - filters.js
 - services.js
- lib/
- partials/

این مورد، روش پیشنهادی در [Angular Seed](#) است و بدین صورت است که تعاریف ماژول ها در فایل app.js انجام می گیرد. تعاریف و پیاده سازی تمام کنترلر ها در فایل controller.js است. و همچنین دایرکتیوها و فیلترها و سرویس ها هر کدام در فایل ها جداگانه تعریف و پیاده سازی می شوند. این روش راه حلی سریع برای پروژه های کوچک با تعداد developer کم است. برای مثال زمانی که یک developer در حال ویرایش فایل controller.js است، از آن جا که فایل مورد نظر checkout خواهد شد در نتیجه سایر developerها امکان تغییر در فایل مورد نظر را نخواهند داشت. سورس فایل ها به مرور زیاد خواهد شد و در نتیجه debug آن سخت می شود.

روش دوم

در این حالت تعاریف کنترلر ها، مدل ها و سرویس ها هرکدام در یک دایرکتوری مجزا قرار خواهد گرفت. برای هر view یک کنترلر و بنا بر نیاز مدل تعریف می کنیم. ساختار آن به صورت زیر می شود:

- controllers/
 - LoginController.js
 - RegistrationController.js
 - ProductDetailController.js
 - SearchResultsController.js
- directives.js
- filters.js
- models/
 - CartModel.js
 - ProductModel.js
 - SearchResultsModel.js
 - UserModel.js
- services/
 - CartService.js
 - UserService.js
 - ProductService.js

دایرکتیوها و فیلترها عموماً در یک فایل قرار داده خواهند شد تا بنابر نیاز در جای مناسب رفرنس داده شوند. این روش ساختار مناسب تری نسبت به روش قبلی دارد اما دارای معایبی هم چون موارد زیر است:

«وابستگی بین فایل ها مشخص نیست در نتیجه بدون استفاده از کتابخانه هایی نظیر requireJs با مشکل مواجه خواهید شد.

«refactoring کدها تا حدودی سخت است.

روش سوم

این ساختار مناسب برای پیاده سازی پروژه ها به صورت ماژولار است و برای پروژه های بزرگ نیز بسیار مناسب است. در این حالت شما فایل های مربوط به هر ماژول را در دایرکتوری خاص آن قرار خواهید داد. به صورت زیر:

- `cart/`
 - `CartModel.js`
 - `CartService.js`
- `common/`
 - `directives.js`
 - `filters.js`
- `product/`
 - `search/`
 - `SearchResultsController.js`
 - `SearchResultsModel.js`
 - `ProductDetailController.js`
 - `ProductModel.js`
 - `ProductService.js`
- `user/`
 - `LoginController.js`
 - `RegistrationController.js`
 - `UserModel.js`
 - `UserService.js`

همان طور که ملاحظه می کنید سرویس ها، کنترلرها و حتی مدل های مربوط به هر بخش در یک مسیر جداگانه قرار می گیرند. علاوه بر آن فایل هایی که قابلیت اشتراکی دارند در مسیری به نام common وجود دارند تا بتوان در جای مناسب برای استفاده از آنها رفرنس داده شود. حتی اگر در پروژه خود فقط یک ماژول دارید باز سعی کنید از این روش برای مدیریت فایل های خود استفاده نمایید. اگر با [ngStart](#) آشنایی داشته باشید به احتمال زیاد با این روش بیگانه نیستید.

بررسی چند نکته درباره کدهای مشترک

در اکثر پروژه های بزرگ، فایل ها و کد هایی وجود خواهد داشت که حالت اشتراکی بین ماژول ها دارند. در این روش این فایل ها در مسیری به نام common یا shared ذخیره می شوند. علاوه بر آن در Angular تکنیک هایی برای به اشتراک گذاشتن این اطلاعات وجود دارد.

«اگر ماژول ها وابستگی شدیدی به فایل ها و سورس های مشترک دارند باید اطمینان حاصل نمایید که این ماژولها فقط به اطلاعات مورد نیاز دسترسی دارند. این اصل interface segregation principle اصول SOLID است.»

«توابعی که کاربرد زیادی دارند و اصطلاحا به عنوان Utility شناخته می شوند باید به `$rootScope` اضافه شوند تا `scope` های وابسته نیز به آنها دسترسی داشته باشند. این مورد به ویژه باعث کاهش تکرار وابستگی های مربوط به هر کنترلر می شود.»

«برای جداسازی وابستگی های بین دو component بهتر از eventها استفاده نمایید. AngularJS این امکان را با استفاده از سرویس های `$on` و `$emit` و `$broadcast` به راحتی میسر کرده است.»

نظرات خوانندگان

نویسنده: ایاک

تاریخ: ۱۳۹۲/۱۱/۲۷ ۹:۵۹

با سلام و تشکر از مقاله خوب شما. برای بارگذاری اسکریپت ها در روش سوم ، از آنجا که ممکن است تعداد دایرکتوری ها زیاد باشد ، شما چه روشی را پیشنهاد می کنید؟

نویسنده: مسعود پاکدل

تاریخ: ۱۳۹۲/۱۱/۲۷ ۱۰:۵۴

زمانی که تعداد فایل ها و دایرکتوری ها در پروژه زیاد می شود (البته این جزء جدانشدنی پروژه های مقیاس بزرگ است) برای جلوگیری از لود یک باره کنترلرها و دایرکتیوها، بهتر از lazy loading برای لود فایل های مورد نیاز استفاده شود. متأسفانه Angular به صورت رسمی از lazy loading پشتیبانی نمی کند اما با کمی تغییر در ساختار و استفاده از کتابخانه های جانبی مثل requireJs یا ScriptJS می توان به این مهم دست یافت. (با عنوان این مطلب که قصد داشتم این مورد را طی یک پست جداگانه بررسی کنم) برای مثال: ابتدا ماژول app خود را به این شکل تنظیم کنید:

```
(function()
{
    var app = angular.module('app', []);

    app.config(function($routeProvider, $controllerProvider, $compileProvider, $filterProvider,
    $provide)
    {
        app.controllerProvider = $controllerProvider;
        app.compileProvider     = $compileProvider;
        app.routeProvider       = $routeProvider;
        app.filterProvider       = $filterProvider;
        app.provider             = $provide;
    });
})();
```

با استفاده از سرویس \$controllerProvider می توان چرخه ساخت کنترلر را به دست گرفت. هم چنین سرویس \$compileProvider برای نمونه سازی دایرکتیوها و \$filterProvider برای فیلترها استفاده می شوند. ساخت کنترلرها و دایرکتیوها نیز به صورت زیر انجام خواهد شد:

```
angular.module('app').controllerProvider.resgister('SomeLazyController', function($scope)
{
    $scope.key = '...';
});
```

و هم چنین یک نمونه از ساخت directive

```
$compileProvider.directive('SomeLazyDirective', function()
{
    return {
        restrict: 'A',
        templateUrl: 'templates/some-lazy-directive.html'
    }
});
```

فقط کافیست در هنگام پیاده سازی routing (که در این [مقاله](#) شرح داده شده است) نوع بارگذاری کنترلرها و دایرکتیو و ... را به صورت lazy انجام دهید :

```
$routeProvider.when('/about', {templateUrl:'views/about.html', resolve:{deps:function($q, $rootScope)
{
    var deferred = $q.defer();
    var dependencies =
    [
        'controllers/AboutViewController.js',
        'directives/some-directive.js'
    ];

    /* نکته اول
    $script(dependencies, function()
    {
        // نکته دوم *
        $rootScope.$apply(function()
        {
            deferred.resolve();
        });
    });

    return deferred.promise;
}}})
```

*نکته اول: تمام وابستگی‌ها توسط scriptJs مدیریت می‌شوند.

*نکته دوم: تمام وابستگی‌ها مربوط به این scope بعد از فراخوانی تابع deferred.resolved بارگذاری خواهند شد.
نقطه شروع برنامه نیز به صورت زیر است:

```
$script(['appModule.js'], function()
{
    angular.bootstrap(document, ['app'])
});
```

[angular.bootstrap](#)

نویسنده: حمید صابری
تاریخ: ۹۵۳ ۱۳۹۲/۱۱/۲۸

ضمن تشکر فراوان از جناب آقای پاکدل عزیز، در این [مقاله](#) به خوبی درباره lazy loading در angularjs بحث شده. نکته مهم اینکه [حتما پروژه‌ای قابل اجرایی که در انتهای مقاله لینک شده](#) را ملاحظه کنید. نکاتی در این پروژه هست از جمله اینکه برای دسترسی به providerها برای lazy loading آنها به این ترتیب به app افزوده شده اند:

```
app.config([
    '$stateProvider',
    '$urlRouterProvider',
    '$locationProvider',
    '$controllerProvider',
    '$compileProvider',
    '$filterProvider',
    '$provide',

    function ($stateProvider, $urlRouterProvider, $locationProvider, $controllerProvider,
    $compileProvider, $filterProvider, $provide) {
        // برای رجیستر کردن غیر همروند اجزای انگیولاری در آینده
        app.lazy =
        {
            controller: $controllerProvider.register,
            directive: $compileProvider.directive,
            filter: $filterProvider.register,
            factory: $provide.factory,
            service: $provide.service
        };
    }
]);
```

(البته این کد از پروژه خودمان است و بعضی وابستگی‌های دیگر هم تزریق شده‌اند).

استفاده از app.lazy باعث سهولت بیشتر در استفاده و خواناتر شدن کد می‌شود. در ادامه به این ترتیب می‌توانید از app.lazy استفاده کنید:

```
angular.module('app').lazy.controller('myController',
    ['$scope', function($scope){
    }]);
```

به این ترتیب کد نوشته شده به دلیل نام گذاری ارجاع \$ controllerProvider با controller به حالت عادی شبیه است، و از طرفی lazy پیش از آن به فهم ماجرا کمک خواهد کرد.

این نقطه شروع یکی از پروژه‌های ماست که به عنوان نمونه بد نیست ملاحظه کنید:

```
<script type="text/javascript">
// --- Scriptjs ---
!function(a, b, c) { function t(a, c) { var e = b.createElement("script"), f = j; e.onload = e.onerror = e[o] = function () { e[m] && !/^c|load/.test(e[m]) || f || (e.onload = e[o] = null, f = 1, c()) }, e.async = 1, e.src = a, d.insertBefore(e, d.firstChild) } function q(a, b) { p(a, function (a) { return !b(a) }) } var d = b.getElementsByTagName("head")[0], e = {}, f = {}, g = {}, h = {}, i = "string", j = !1, k = "push", l = "DOMContentLoaded", m = "readyState", n = "addEventListener", o = "onreadystatechange", p = function (a, b) { for (var c = 0, d = a.length; c < d; ++c) if (!b(a[c])) return j; return 1 }; !b[m] && b[n] && (b[n](l, function r() { b.removeEventListener(l, r, j), b[m] = "complete" }, j), b[m] = "loading"); var s = function (a, b, d) { function o() { if (!--m) { e[l] = 1, j && j(); for (var a in g) p(a.split("|"), n) && !q(g[a], n) && (g[a] = []) } } function n(a) { return a.call ? a() : e[a] } a = a[k] ? a : [a]; var i = b && b.call, j = i ? b : d, l = i ? a.join("") : b, m = a.length; c(function () { q(a, function (a) { h[a] ? (l && (f[l] = 1), o()) : (h[a] = 1, l && (f[l] = 1), t(s.path ? s.path + a + ".js" : a, o)) } }, 0); return s }; s.get = t, s.ready = function (a, b, c) { a = a[k] ? a : [a]; var d = []; !q(a, function (a) { e[a] || d[k](a) }) && p(a, function (a) { return e[a] }) ? b() : !function (a) { g[a] = g[a] || [], g[a][k](b), c && c(d) }(a.join("|")); return s }; var u = a.$script; s.noConflict = function () { a.$script = u; return this }, typeof module != "undefined" && module.exports ? module.exports = s : a.$script = s }(this, document, setTimeout)

$script(['/Scripts/Lib/jquery/jquery-1.10.2.min.js'], function () {
    $script(['/Scripts/Lib/angular/angular.js'], function () {
        $script(['/Scripts/Lib/angular/angular-ui-router.min.js',
            '/Scripts/Lib/angular/angular-resource.min.js',
            '/Scripts/Lib/angular/angular-cache.min.js',
            '/Scripts/Lib/angular/angular-sanitize.min.js',
            '/Scripts/Lib/angular/angular-animate.min.js',
            '/Scripts/Lib/angular/angular-cookie.min.js',
            '/APP/Common/directives.js'
        ], function () {
            $script('/app/app.js', function () {
                angular.bootstrap(document, ['app']);
            });
        });
    });
});
</script>
```

این تگ script در صفحه شروع پروژه آمده است.

کد minify شده scriptjs در ابتدا قرار دارد، پس از آن فایل‌های js مورد نیاز با رعایت وابستگی‌های احتمالی به ترتیب بارگذاری شده‌اند.

این قسمت resolve یکی از بخش‌های مسیریابی است:

```
resolve: {
    fileDeps: ['$q', '$rootScope', function ($q, $rootScope) {
        var deferred = $q.defer();
        var deps = ['/app/HotStories/dataContextService.js',
            '/app/HotStories/hotStController.js'];
        $script(deps, function () {
            $rootScope.$apply(function () {
                deferred.resolve();
            });
        });
        return deferred.promise;
    }]
}
```



```
}
```

این نحوه تعریف سرویسی که فایل آن در وابستگی‌ها آمده و قرار است lazy load شود:

```
angular.module('app').lazy.service('dataContextService',
    ['$rootScope', '$resource', '$angularCacheFactory', '$q', function($rootScope, $resource,
    $cacheFactory, $q){
    ...
    }]);
```

و این هم نحوه تعریف کنترلری که فایل آن در وابستگی‌ها آمده و قرار است lazy load شود:

```
angular.module('app').lazy.controller('hotStController',
    ['$scope', 'ipCookie', 'dataContextService', function($scope, ipCookie, dataContextService){
    ...
    }]);
```

نویسنده:

علی فخرایی

تاریخ:

۱۳:۱۲ ۱۳۹۲/۱۲/۲۷

ممنون از مطلب مفیدتون. اگر ما یک area مثلا به نام administrator برای مدیریت داشته باشیم، آیا باید فایل‌ها را در مسیر ریشه مثلا در پوشه script قرار دهیم؟ یا باید در همان area؟ چون اگر در ریشه قرار دهیم جالب به نظر نمیرسد. ممکنه راهنمایی کنید؟

نویسنده:

مسعود پاکدل

تاریخ:

۱۷:۳۱ ۱۳۹۲/۱۲/۲۷

خیر. می‌توانید فایل‌های مورد نیاز هر ماژول و area را در مسیرهای جداگانه مربوط به area قرار دهید. پوشه Scripts صرفا برای قرار گیری فایل‌های مورد نیاز کتابخانه هاست (نظیر JQuery و angular و q و ...).

نویسنده:

علی فخرایی

تاریخ:

۲۰:۱ ۱۳۹۲/۱۲/۲۷

تشکر. شما در مورد مسیر یابی هم قطعه کدی قرار دادید که میشود وابستگی‌ها و ... را تزریق کرد. منتها اگر ما بیش از 100 مسیر داشته باشیم باید چه کنیم؟ یعنی به ازای هر مسیر باید این قطعه کد تکرار شود :

```
$routeProvider.when('/about', {templateUrl:'views/about.html', resolve:{deps:function($q, $rootScope)
{
    var deferred = $q.defer();
    var dependencies =
    [
        'controllers/AboutViewController.js',
        'directives/some-directive.js'
    ];
    /** نکته اول
    $script(dependencies, function()
    {
        // نکته دوم *
        $rootScope.$apply(function()
        {
            deferred.resolve();
        });
    });
    return deferred.promise;
}}})
```

راه حل پویایی وجود دارد؟

مثلا شما در ساختار سوم بیان کردید که فایل های مربوط به هر قسمت در کنار هم باشند. اعم از کنترلر و دایرکتیوها و فیلترها و ...

آیا میشود برای هر قسمت مثل cart, user, product, یک ماژول app جدا نوشت و در آن طبق مثال شما مسیریابی را تولید کرد؟ یعنی چندین ماژول انگولار app.js برای یک پروژه نوشت؟ استاندارد است؟ بدین صورت دیگر نگران تعداد مسیرهای زیاد نیستیم و مشخص میشود که مسیریابی هر قسمت در کنار آن وجود دارد. امکان پذیر است؟ اگر نیست شما چه راهی برای این کار دارید. ممنون

نویسنده: علی فخرایی
تاریخ: ۱۴:۲۳ ۱۳۹۳/۰۱/۰۲

میشه یک مثال ساده هم در مورد کامنت های دوم و سوم قرار بدید؟

من کلیه مراحل رو پیش رفتم و دو روز کامل درگیرش هستم، اما به نتیجه ای نمیرسم. خطاهای زیر رو در کنسول کروم دریافت میکنم.

```
Uncaught Error: [$injector:modulerr] Failed to instantiate module app due to:
Error: [$injector:nomod] Module 'app' is not available! You either misspelled the module name or forgot
to load it. If registering a module ensure that you specify the de...<omitted>...0) angular.js:78
Uncaught ReferenceError: app is not defined selectAllCheckbox.js:3
Uncaught Error: [$injector:modulerr] Failed to instantiate module app due to:
Error: [$injector:unpr] Unknown provider: $routeProvider
http://errors.angularjs.org/1.2.14/$injector/unpr?p0=%24routeProvider
at http://localhost:8417/Scripts/Angula...<omitted>...0)
```

نویسنده: حمید رضا منصوری
تاریخ: ۱۶:۲۷ ۱۳۹۳/۰۱/۰۳

با تشکر بابت راهنمایتون

لطفا میتونید یه sample ساده از این مطلبتون بزارید. البته اون مثال لینکی که گذاشته بودید رو دیدم ولی نتونستم اجراش کنم و نمونه داخل خودش هم کار نمی کرد.

نویسنده: مسعود پاکدل
تاریخ: ۱۰:۱۶ ۱۳۹۳/۰۱/۰۵

بخش اول سوال: بهتر است که کد مربوط به لود وابستگی ها در یک تابع مجزا نوشته شود و فقط در زمان نیاز این تابع را با پاس دادن وابستگی فراخوانی نمایید (با فرض اینکه نام این فایل dependencyResolver است):

```
(function()
{
  return function(dependencies)
  {
    var definition =
    {
      resolver: ['$q','$rootScope', function($q, $rootScope)
      {
        var deferred = $q.defer();
        $script(dependencies, function()
        {
          $rootScope.$apply(function()
          {
            deferred.resolve();
          });
        });
        return deferred.promise;
      }
    ]
  }
  return definition;
})
```

```
});
```

و برای لود وابستگی نیز تابع `dependencyResolver` را به این صورت فراخوانی نمایید:

```
angular.forEach(config.routes, function(route, path)
{
    $routeProvider.when(path, {templateUrl:route.templateUrl,
    resolve:dependencyResolver(route.dependencies)}});
});
```

در مورد سوال دوم نیز باید عنوان کنم که شما می‌توانید مسیریابی هر ماژول را به صورت جداگانه در تعاریف همان ماژول‌ها انجام دهید که البته روشی مرسوم و معمول است. فقط در هنگام عملیات bootstrapping ماژول اصلی برنامه، سایر ماژول‌ها به عنوان وابستگی آن تعیین می‌شوند. به صورت زیر(عنوان ماژول‌ها را یکتا انتخاب نمایید):

```
var app = angular.module('app', ['anotherModule1' , 'anotherModule2' , 'anotherModule3']);
```

نویسنده: حمید صابری
تاریخ: ۱۷:۰ ۱۳۹۳/۰۱/۰۸

دوست عزیز [اینجا](#) می‌توانید توضیحات بیشتر درباره lazy loading و [یک پیاده سازی ساده](#) از اونو مطالعه کنید.

نویسنده: حمید صابری
تاریخ: ۱۷:۱ ۱۳۹۳/۰۱/۰۸

دوست عزیز [اینجا](#) می‌توانید توضیحات بیشتر درباره lazy loading و [یک پیاده سازی ساده](#) از اونو مطالعه کنید.

نویسنده: ایاک
تاریخ: ۱۲:۴۰ ۱۳۹۳/۰۱/۲۷

با تشکر.

برای این قسمت در صورت امکان توضیح بیشتری می‌دهید؟

```
angular.forEach(config.routes, function(route, path)
{
    $routeProvider.when(path, {templateUrl:route.templateUrl,
    resolve:dependencyResolver(route.dependencies)}});
});
```

این کد باید در کجا نوشته شود و مقدار `config.routes` از کجا دریافت می‌شود؟

بعد از مطالعه پست‌های [^](#) و [^](#) نکته ای به ذهنم رسید که بیان آن از بنده و مطالعه آن توسط شما خالی از لطف نیست. اگر مثال‌های پیاده سازی شده در پست‌های [^](#) و [^](#) را با AngularJS نسخه 1.2 اجرا نمایید به طور حتم با خطا روبرو می‌شوید و نتیجه مورد نظر حاصل نمی‌شود. در این [پست](#) نیز توسط یکی از دوستان اشاره ای به این مطلب شد. دلیل خطا این است که از نسخه 1.2 به بعد در Angular سیستم مسیریابی به این شکل امکان پذیر نیست و بخش مسیریابی به یک فایل دیگر به نام angular-route.js منتقل شده است. در نتیجه اگر به سبک نسخه‌های قبلی Angular از سیستم مسیریابی استفاده نمایید با خطا مواجه خواهید شد و خطای مورد نظر هم مربوط به عدم توانایی در تزریق وابستگی \$routeProvider به ماژول مورد نظر است. حال راه حل چیست؟ کافیهست در هنگام تعریف ماژول، ngRoute را به عنوان وابستگی ماژول تعیین نمایید. و از طرفی فایل اسکریپتی angular-route.js را بعد از angular.js فراخوانی کنید. بررسی مثال:

کدهای زیر مربوط به مثال‌های پست قبلی می‌باشد که شرح کامل آن در این [پست](#) است:

```
var myFirstRoute = angular.module('myFirstRoute', []);

myFirstRoute.config(['$routeProvider',
function($routeProvider) {
    $routeProvider.
        when('/pageOne', {
            templateUrl: 'templates/page_one.html',
            controller: 'ShowPage1Controller'
        }).
        when('/pageTwo', {
            templateUrl: 'templates/page_two.html',
            controller: 'ShowPage2Controller'
        }).
        otherwise({
            redirectTo: '/pageOne'
        });
}]);

myFirstRoute.controller('ShowPage1Controller', function($scope) {
    $scope.message = 'Content of page-one.html';
});

myFirstRoute.controller('ShowPage2Controller', function($scope) {
    $scope.message = 'Content of page-two.html';
});
```

برای هماهنگ کردن مثال بالا با نسخه 1.2 باید به روش زیر عمل نمود:

```
var myFirstRoute = angular.module('myFirstRoute',['ngRoute']);

myFirstRoute.config(['$routeProvider',
function($routeProvider) {
    $routeProvider.
        when('/pageOne', {
            templateUrl: 'templates/page_one.html',
            controller: 'ShowPage1Controller'
        }).
        when('/pageTwo', {
            templateUrl: 'templates/page_two.html',
            controller: 'ShowPage2Controller'
        }).
        otherwise({
            redirectTo: '/pageOne'
        });
}]);
```

تنها تغییر، مشخص کردن ngRoute به عنوان وابستگی ماژول myFirstRoute است (خط اول). نکته دیگر لود فایل angular-route.js قبل از فراخوانی فایل بالا و بعد از فراخوانی فایل angular.js است:

```
<body ng-app="app">
  <div>
    <div>
      <div>
        <ul>
          <li><a href="#pageOne"> Show page one </a></li>
          <li><a href="#pageTwo"> Show page two </a></li>
        </ul>
      </div>
      <div>
        <div ng-view></div>
      </div>
    </div>
  </div>

  <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.0.7/angular.min.js"></script>
  <script src="angular-route.js"></script>
  <script src="app.js"></script>
</body>
```

نظرات خوانندگان

نویسنده: ناصر طاهری
تاریخ: ۱۶:۲۶ ۱۳۹۲/۱۱/۲۹

ممنون از نکته خوبتون.
آیا برای ساماندهی تعداد بالای مسیرها ، راه حلی وجود داره؟

نویسنده: مسعود پاکدل
تاریخ: ۲۳:۱۷ ۱۳۹۲/۱۱/۲۹

بله. من از [Angular Dynamic Routing](#) استفاده می‌کنم.

نویسنده: ایاک
تاریخ: ۲۰:۸ ۱۳۹۲/۱۲/۰۱

با سلام.
نظرتون راجع به پروژه [angular-ui-router](#) برای مسیریابی انگولار که قابلیت بیشتری را نسبت به سیستم مسیریابی پیش فرض آن ارائه می‌دهد چیست؟

نویسنده: محسن کریمی
تاریخ: ۱۳:۲۷ ۱۳۹۲/۱۲/۰۲

تمام مواردی که در routing مربوط به angularjs هستش در ui-router هم وجود دارد و مواردی مثل nested views و multiple named views بهش اضافه شده و عملاً در پروژه‌ها کاربردی‌تر خواهد بود.

[UI-Router](#) ابزاری برای مسیریابی در AngularJS است که این امکان را برایتان فراهم می‌کند تا بخش‌های برنامه رابط کاربریتان را به شکل یک ماشین حالت ساماندهی کنید. برخلاف سرویس \$route که بر اساس مسیریابی URLها ساماندهی شده و کار می‌کند، [UI-Router](#) بر اساس حالت‌ها کار می‌کند، که این حالت‌ها می‌توانند در صورت لزوم مسیریابی هم داشته باشند.

[UI-Router](#) یکی از افزونه‌های مجموعه [Angular-ui](#)، و پاراگراف بالا معرفی آن در [صفحه خانگی](#) است (تقریباً!). این افزونه جزئیات مفصلی دارد و در این مطلب تنها به معرفی آن خواهیم پرداخت (بر اساس مطالب [صفحه خانگی](#)). پیش از ادامه پیشنهاد می‌کنم اگر مطالب زیر را نخوانده‌اید ابتدا آن‌ها را مرور کنید:

[مسیریابی در AngularJS #بخش اول](#)

[مسیریابی در AngularJS #بخش دوم](#)

[مسیریابی در AngularJS #بخش سوم](#)

برای استفاده از [UI-Router](#) باید:

فایل جاوا اسکریپت آن را دانلود کنید ([released](#) یا [minified](#)).

در صفحه اصلی برنامه‌تان پس از include کردن فایل اصلی AngularJS فایل `angular-ui-router.js` (یا `angular-ui-router.min.js`) را include کنید.

'ui.router' را به لیست وابستگی‌های ماژول اصلی اضافه کنید.

نتیجه چیزی شبیه این خواهد بود:

```
<!doctype html>
<html ng-app="myApp">
<head>
  <script src="//ajax.googleapis.com/ajax/libs/angularjs/1.1.5/angular.min.js"></script>
  <script src="js/angular-ui-router.min.js"></script>
  <script>
    var myApp = angular.module('myApp', ['ui.router']);
    // For Component users, it should look like this:
    // var myApp = angular.module('myApp', [require('angular-ui-router')]);
  </script>
  ...
</head>
<body>
  ...
</body>
</html>
```

حالت‌ها و viewهای تو در تو قابلیت اصلی [UI-Router](#) امکان تعریف حالت‌ها و viewهای تو در تو است. در مطلب [مسیریابی در AngularJS #بخش اول](#) دایرکتیو `ng-view` معرفی شده است. هنگام استفاده از سرویس `$route` با این دایرکتیو می‌توان محل مورد نظر برای بارگذاری محتویات مربوط به مسیرها را مشخص کرد. دایرکتیو `ui-view` در [UI-Router](#) همین نقش را دارد. فرض کنید این کد فایل `index.html` باشد:

```
<!-- index.html -->
<body>
  <div ui-view></div>
  <!-- We'll also add some navigation: -->
  <a ui-sref="state1">State 1</a>
  <a ui-sref="state2">State 2</a>
</body>
```

همانطور که ملاحظه می‌کنید در تگ‌های `a` از دایرکتیو `ui-sref` استفاده شده است. این دایرکتیو علاوه بر مدیریت تغییر حالت،

خصوصیت href تگ a را در صورتی که حالت مشخص شده URL داشته باشد تولید می‌کند. البته برای استفاده از UI-Router ملزم به استفاده از دایرکتیو ui-sref نیستید و می‌توانید href را مشخص کنید. ولی با استفاده از ui-sref لازم نیست مسیر یک حالت را به یاد داشته باشید، و یا در صورت تغییر آن، همه hrefها را به روز کنید.

در ادامه برای هر کدام از حالت‌ها یک template اضافه می‌کنیم:

فایل state1.html:

```
<!-- partials/state1.html -->
<h1>State 1</h1>
<hr/>
<a ui-sref="state1.list">Show List</a>
<div ui-view></div>
```

فایل state2.html:

```
<!-- partials/state2.html -->
<h1>State 2</h1>
<hr />
<a ui-sref="state2.list">Show List</a>
<div ui-view></div>
```

دو نکته قابل توجه در این templateها وجود دارد. اول اینکه همانطور که می‌بینید templateها خود شامل تگی با دایرکتیو ui-view هستند. و دوم مقدار دایرکتیو ui-sref است که به صورت state1.list و state2.list آمده است. این جدا سازی با نقطه نشان دهنده سلسله مراتب حالت‌هاست. یعنی حالت‌های state1 و state2 هر کدام حالت فرزندی به نام list دارند. در ادامه وقتی حالت‌ها و مسیریابی را در app.config() تعریف کنیم این مسائل از هاله‌ای از ابهام که در آن هستند خارج می‌شوند! فعلا بیاید با راهنمای UI-Router پیش برویم و فایل‌های template حالت‌های فرزند را تعریف کنیم. templateهایی که قرار است در ui-view پدران‌شان بارگذاری شوند:

```
<!-- partials/state1.list.html -->
<h3>List of State 1 Items</h3>
<ul>
  <li ng-repeat="item in items">{{ item }}</li>
</ul>
```

```
<!-- partials/state2.list.html -->
<h3>List of State 2 Things</h3>
<ul>
  <li ng-repeat="thing in things">{{ thing }}</li>
</ul>
```

خوب! حالا برویم سراغ شعبده بازی! برای اینکه از UI-Router استفاده کنید لازم است \$stateProvider و \$urlRouterProvider را به عنوان وابستگی به app.config() تزریق کنید:

```
myApp.config(['$stateProvider', '$urlRouterProvider',
function($stateProvider, $urlRouterProvider) {
  //
  // For any unmatched url, redirect to /state1
  $urlRouterProvider.otherwise("/state1");
  //
  // Now set up the states
  $stateProvider
    .state('state1', {
      url: "/state1",
      templateUrl: "partials/state1.html"
    })
    .state('state1.list', {
      url: "/list",
      templateUrl: "partials/state1.list.html",
      controller: function($scope) {
        $scope.items = ["A", "List", "Of", "Items"];
      }
    })
})
```



```
.state('state2', {
  url: "/state2",
  templateUrl: "partials/state2.html"
})
.state('state2.list', {
  url: "/list",
  templateUrl: "partials/state2.list.html",
  controller: function($scope) {
    $scope.things = ["A", "Set", "Of", "Things"];
  }
})
}]);
```

در ابتدا با متد `$urlRouterProvider.otherwise()` مسیر پیشفرض مشخص شده است. متد `otherwise` را باید از مقالات مسیریابی در AngularJS به یاد داشته باشید. سپس حالت‌های برنامه با استفاده از متد `state` تعریف شده است. این متد دو پارامتر ورودی دارد؛ اولی نام حالت و دومی یک شی شامل خصوصیات حالت. همانطور که می‌بینید این شی خصوصیات شبیه به همان‌ها که در متد `$routeProvider.when()` وجود داشت دارد. می‌شود گفت این خصوصیات همان‌ها هستند و همان عملکرد را دارند.

خصوصیت `url` مشخص کننده مسیر حالت است. این خصوصیت همان مقدار است که به عنوان پارامتر اول به `$routeProvider.when()` پاس می‌شد. در این پارامتر می‌شود متغیرهای `url` را هم به همان ترتیب تعریف کرد. مثلاً اگر حالت `state1` در آدرسش یک پارامتر `id` داشته باشد می‌شود آن را به این ترتیب تعریف کرد:

```
.state('state1', {
  url: "/state1/:id",
  templateUrl: "partials/state1.html"
})
```

برای خواندن مقدار این متغیر باید از `$stateParams` استفاده کرد:

```
$stateParams.id
```

به خصوصیت `url` دو حالت `state1.list` و `state2.list` دقت کنید. هر دو برابر `/list` است. یعنی هر دو یک مسیر دارند؟ نه! بلکه مسیر `state1.list` برابر `'state1/list/'` و مسیر `state2.list` برابر `'state2/list/'` است. در واقع حالت `state1.list` یعنی `list` فرزند `state1` و به همین ترتیب `state2.list` یعنی `list` فرزند `state2`. و می‌توان گفت UI-Router آدرس `url` حالت فرزند را، آدرسی نسبی، نسبت به `url` حالت پدر می‌داند. این رابطه سلسله مراتبی و پدر و فرزندی را می‌توان با استفاده از خصوصیت `parent` به صورت صریح‌تری مشخص کرد:

```
.state('list', {
  parent: "state1",
  url: "/list",
  templateUrl: "partials/state1.list.html",
  controller: function($scope) {
    $scope.items = ["A", "List", "Of", "Items"];
  }
})
.state('list', {
  parent: "state2",
  url: "/list",
  templateUrl: "partials/state2.list.html",
  controller: function($scope) {
    $scope.items = ["A", "List", "Of", "Items"];
  }
})
```

تا اینجا کار، اگر آدرس `"/state1"` وارد شود، فایل `"partials/state1.html"` در `"ui-view"` فایل `"index.html"` بارگذاری خواهد شد. اگر آدرس `"/state1/list"` وارد شود، ابتدا فایل `"partials/state1.html"` در `"ui-view"` فایل `"index.html"` بارگذاری شده، سپس فایل `"partials/state1.list.html"` در `"ui-view"` آمده در فایل `"partials/state1.html"` بارگذاری می‌شود. این همان امکان حالت‌ها و `view`های تو در تو است که UI-Router فراهم می‌کند. [اینجا](#) می‌توانید خروجی کدهای بالا را مشاهده کنید. اگر مستقیماً `url` یک حالت فرزند وارد شود، یا به عبارت دیگر، اگر بخواهیم مستقیماً برنامه به حالتی که فرزند حالت دیگر است برود، UI-Router برنامه را ابتدا به حالت پدر، و پس از آن به حالت فرزند خواهد برد. حالت فرزند دو چیز را از حالت پدر به ارث

می‌برد:

وابستگی‌های فراهم شده در حالت پدر به وسیله " [resolve](#) "[داده‌های سفارشی](#) مشخص شده در خصوصیت data حالت پدر

استفاده از resolve در UI-Router مشابه [استفاده از آن در \\$route](#) است. ولی افزودن داده‌های سفارشی کمی متفاوت است. برای افزودن داده‌های سفارشی باید از خصوصیت data یک حالت استفاده کرد:

```
.state('state1', {
  url: "/state1",
  templateUrl: "partials/state1.html",
  data:{
    foodata: 'addorder'
  }
})
```

برای دسترسی به این داده‌ها هم می‌توان از `$state.current.data` استفاده کرد:

```
$state.current.data.foodata
```

Viewهای نامگذاری شده و چندگانه

یکی دیگر از قابلیت‌های کاربردی UI-Router امکان داشتن چند `ui-view` در هر `template` است (استفاده همزمان از این قابلیت و حالت‌های تو در تو، امکان مدیریت واسط کاربری را به خوبی فراهم می‌کند). برای توضیح این قابلیت، با [راهنمای UI-Router](#) همراه شویم:

1. دستورالعمل برپایی UI-Router که در بالا آمده را اجرا کنید.

2. یک یا چند `ui-view` به برنامه‌تان اضافه کنید و آن‌ها را نامگذاری کنید:

```
<!-- index.html -->
<body>
  <div ui-view="viewA"></div>
  <div ui-view="viewB"></div>
  <!-- Also a way to navigate -->
  <a ui-sref="route1">Route 1</a>
  <a ui-sref="route2">Route 2</a>
</body>
```

3. حالت‌های برنامه‌تان را در روال `config` ماژول تعریف کنید:

```
myApp.config(function ($stateProvider) {
  $stateProvider
    .state('index', {
      url: "",
      views: {
        "viewA": { template: "index.viewA" },
        "viewB": { template: "index.viewB" }
      }
    })
    .state('route1', {
      url: "/route1",
      views: {
        "viewA": { template: "route1.viewA" },
        "viewB": { template: "route1.viewB" }
      }
    })
    .state('route2', {
      url: "/route2",
      views: {
        "viewA": { template: "route2.viewA" },
        "viewB": { template: "route2.viewB" }
      }
    })
})
```

```
});
})
```

4. خروجی کدهای بالا را [اینجا](#) مشاهده کنید.

چند نکته

[UI-Router](#) جزئیات فراوانی دارد و آنچه آمد تنها پرده برداری از آن بود. دلم می‌خواستم می‌توانستم بیش از این آن را معرفی کنم، اما متأسفانه این روزها وقت آزاد کافی ندارم. در انتها می‌خواهم به چند نکته اشاره کنم: **روش controller as** برای استفاده از روش controller as در UI-Router باید به این ترتیب عمل کنید:

```
.state('list', {
  parent: "state1",
  url: "/list",
  templateUrl: "partials/state1.list.html",
  controller: "state1ListController as listCtrl1"
})
.state('list', {
  parent: "state2",
  url: "/list",
  templateUrl: "partials/state2.list.html",
  controller: "state2ListController as listCtrl2"
})
```

حالت‌های انتزاعی

[حالت انتزاعی](#) حالتی است که url ندارد و در نتیجه برنامه نمی‌تواند در آن حالت قرار گیرد. حالت‌های انتزاعی بسیار به درد خور هستند! مثلاً فرض کنید چند حالت دارید که اشتراکاتی با هم دارند (همه باید در template مشابهی بارگذاری شود، یا وابستگی‌های یکسانی دارند، یا حتی سطح دسترسی یکسان). با تعریف یک حالت انتزاعی و جمع کردن همه وابستگی‌ها در آن، و تعریف حالت‌های مورد نظرتان به عنوان فرزندان حالت انتزاعی، می‌توانید اشتراکات حالت‌های برنامه را ساده‌تر مدیریت کنید.

حساسیت به حروف بزرگ و کوچک

در سرویس \$route با مقداردهی خصوصیت `caseInsensitiveMatch` می‌توانستیم مشخص کنیم که بزرگ و کوچک بودن حروف در تطبیق آدرس صفحه با پارامتر route در نظر گرفته بشود یا نه. خودمانیش اینکه url به حروف بزرگ و کوچک حساس باشد یا نه. متأسفانه در UI-Router از این امکان خبری نیست (البته فعلاً) و آدرس‌های تعریف شده به حروف بزرگ و کوچک حساس هستند. [اینجا](#) روشی برای حل این مشکل پیشنهاد شده، به این ترتیب که همه url‌های وارد شده به حروف کوچک تبدیل شود (راستش من این راه حل را نمی‌پسندم!). [چند روز قبل هم تغییراتی در کد UI-Router داده شده که امکان حساس نبودن به حروف کوچک و بزرگ فراهم شود](#). این تغییر هنوز در نسخه نهایی فایل UI-Router نیامده است. هرچند اگر بیاید هم آنچه تا امروز (23 اسفند 92) انجام شده مشکل را حل نمی‌کند.

اگر شما هم مثل من می‌خواهید کلاً آدرس‌ها به حروف بزرگ و کوچک حساس نباشند، و فرصت حل کردن اساسی مشکل را هم ندارید به این ترتیب عمل کنید:

در فایل "angular-ui-router.js" عبارت "(new RegExp(compiled, 'i'))" را پیدا کرده و آن را به "(new RegExp(compiled, 'i'))" تبدیل کنید. و یا در "angular-ui-router.min.js" (هرکدام از فایل‌ها که استفاده می‌کنید) عبارت "(new RegExp(o" را پیدا کرده و آن را به "new RegExp(o, 'i'" تبدیل کنید. همین! صدایش را هم در نیاورید!

وقتی پروژه انگیولاری‌تان کمی گسترش پیدا کند، تعداد زیادی فایل شامل کنترلرها، سرویس‌ها، دایرکتیوها و ... خواهید داشت. واضح است که همه این اجزا همراه با هم مورد نیاز نیستند و برای افزایش سرعت بارگذاری سایت و صرفه جویی در مصرف پهنای باند بهتر است هرکدام از آن‌ها را در هنگام نیاز بارگذاری کنیم. این یعنی همان lazy loading خودمان! در AngularJS امکانی برای lazy loading فایل‌ها پیش‌بینی نشده است، پس باید از ابزارهای دیگری که این امکان را فراهم می‌کنند استفاده کرد. من در ادامه از [Script.js](#) برای این کار استفاده خواهم کرد، ولی شما می‌توانید از هر کتابخانه دیگری استفاده کنید.

اما مسئله دیگری که پیش از lazy loading فایل‌ها باید تکلیفش را معلوم کنیم، این است که چطور می‌توانیم اجزایی را به ماژولی که قبلاً راه‌اندازی (bootstrap) شده اضافه کنیم. اگر بخواهیم برای مثال کنترلری را در یک فایل مجزا تعریف کنیم، باید آن را به شکلی در ماژول برنامه‌مان ثبت کنیم. فرض کنید این کار را به این ترتیب انجام دهیم:

```
angular.module('app').controller('SomeLazyController', function($scope)
{
    $scope.key = '...';
});
```

در این صورت اگر این کنترلر را در قسمتی از برنامه به صورت `ng-controller='SomeLazyController'` استفاده کنیم با این خطا مواجه خواهیم شد:

```
Error: Argument 'SomeLazyController' is not a function, got undefined
```

برای این کار (افزودن اجزایی به ماژولی که قبلاً راه‌اندازی شده) می‌توانیم بجای استفاده از API‌های ماژول، از provider های AngularJS استفاده کنیم. به این ترتیب برای ثبت یک کنترلر باید از [\\$controllerProvider](#) ، برای ثبت یک directive از [\\$compileProvider](#) ، برای ثبت فیلترها از [\\$filterProvider](#) و برای ثبت سایر اجزا در ماژول از [\\$provide](#) استفاده کنیم:

```
// Registering a controller after app bootstrap
$controllerProvider.register('SomeLazyController', function($scope)
{
    $scope.key = '...';
});

// Registering a directive after app bootstrap
$compileProvider.directive('SomeLazyDirective', function()
{
    return {
        restrict: 'A',
        templateUrl: 'templates/some-lazy-directive.html'
    }
});

// etc
```

اما نکته‌ای که درباره provider ها وجود دارد این است که آن‌ها تنها در روال config یک ماژول در دسترس هستند. بنا بر این برای دسترسی به آن‌ها پس از اجرای این روال، ارجاعی به آنها را باید نگهداری کنیم:

```
(function () {
    app = angular.module("app", []);

    app.config([
        '$controllerProvider',
        '$compileProvider',
        '$filterProvider',
        '$provide',
```

```
function ($controllerProvider, $compileProvider, $filterProvider, $provide) {
    // برای رجیستر کردن غیر همروند اجزای انگیولاری در آینده
    app.lazy =
    {
        controller: $controllerProvider.register,
        directive: $compileProvider.directive,
        filter: $filterProvider.register,
        factory: $provide.factory,
        service: $provide.service
    };
}());
```

اکنون SomeLazyController را به این ترتیب می‌توانیم ثبت کنیم:

```
angular.module('app').lazy.controller('SomeLazyController', function($scope)
{
    $scope.key = '...';
});
```

نکته دیگر این است که کجا باید lazy loadign را انجام دهیم. به نظر می‌رسد مناسب‌ترین محل برای انجام این کار خصوصیت resolve مسیریابی است. در [این مطلب](#) و [این مطلب](#) resolve در \$route و UI-Router معرفی شده است:

```
$stateProvider
    .state('state1', {
        url: '/state1',
        template: '<div>{{st1Ctrl.msg}}</div>',
        controller: 'state1Controller as st1Ctrl',
        resolve: {
            fileDeps: ['$q', '$rootScope', function ($q, $rootScope) {
                var deferred = $q.defer();
                var deps = [
                    'app/messageService.js',
                    'app/state1Controller.js'];
                $script(deps, function () {
                    $rootScope.$apply(function () {
                        deferred.resolve();
                    });
                });
                return deferred.promise;
            }]
        }
    })
    .state('state2', {
        url: '/state2',
        template: '<div>{{st2Ctrl.msg}}</div>',
        controller: 'state2Controller as st2Ctrl',
        resolve: {
            fileDeps: ['$q', '$rootScope', function ($q, $rootScope) {
                var deferred = $q.defer();
                var deps = [
                    'app/messageService.js',
                    'app/state2Controller.js'];
                $script(deps, function () {
                    $rootScope.$apply(function () {
                        deferred.resolve();
                    });
                });
                return deferred.promise;
            }]
        }
    })
    });
```

کنترلر state1Controller که در فایلی با همین نام پیاده‌سازی شده است تنها در مسیر /state1 مورد نیاز است، و state2Controller تنها در مسیر /state2 لازم است بارگذاری شود. هردوی این کنترلرها به messageService وابستگی دارند که در messageService.js پیاده‌سازی شده است (همانطور که در [این مطلب](#) اشاره شده می‌توانیم یک حالت انتزاعی به عنوان پدر دو حالت موجود تعریف کرده و وابستگی مشترک را به آن منتقل کنیم). برای بارگذاری فایل‌های مورد نیاز در ابتدای کار و راه اندازی اولیه برنامه هم می‌توان به این ترتیب عمل کرد:

```

<script type="text/javascript">
// ----Script.js----
!function(a, b, c) { function t(a, c) { var e = b.createElement("script"), f = j; e.onload =
e.onerror = e[o] = function () { e[m] && !/^c|load/.test(e[m]) || f || (e.onload = e[o] = null, f = 1,
c()) }, e.async = 1, e.src = a, d.insertBefore(e, d.firstChild) } function q(a, b) { p(a, function (a)
{ return !b(a) }) } var d = b.getElementsByTagName("head")[0], e = {}, f = {}, g = {}, h = {}, i =
"string", j = !1, k = "push", l = "DOMContentLoaded", m = "readyState", n = "addEventListener", o =
"onreadystatechange", p = function (a, b) { for (var c = 0, d = a.length; c < d; ++c) if (!b(a[c]))
return j; return 1 }; !b[m] && b[n] && (b[n](l, function r() { b.removeEventListener(l, r, j), b[m] =
"complete" }, j), b[m] = "loading"); var s = function (a, b, d) { function o() { if (!--m) { e[l] = 1,
j && j(); for (var a in g) p(a.split("|"), n) && !q(g[a], n) && (g[a] = []) } } function n(a) { return
a.call ? a() : e[a] } a = a[k] ? a : [a]; var i = b && b.call, j = i ? b : d, l = i ? a.join("") : b, m
= a.length; c(function () { q(a, function (a) { h[a] ? (l && (f[l] = 1), o()) : (h[a] = 1, l && (f[l] =
1), t(s.path ? s.path + a + ".js" : a, o)) }) }, 0); return s }; s.get = t, s.ready = function (a, b,
c) { a = a[k] ? a : [a]; var d = []; !q(a, function (a) { e[a] || d[k](a) }) && p(a, function (a) {
return e[a] }) ? b() : !function (a) { g[a] = g[a] || [], g[a][k](b), c && c(d) }(a.join("|")); return
s }; var u = a.$script; s.noConflict = function () { a.$script = u; return this }, typeof module !=
"undefined" && module.exports ? module.exports = s : a.$script = s }(this, document, setTimeout)

    $script('Scripts/angular.js', function () {
        $script('Scripts/angular-ui-router.js', function () {
            $script('app/app.js', function () {
                angular.bootstrap(document, ['app']);
            });
        });
    });
</script>

```

توجه داشته باشید که لازم نیست بارگذاری فایل‌ها حتماً یکی پس از دیگری باشد. ترتیب بارگذاری فایل‌ها تنها در آن‌هایی که وابستگی به هم دارند باید رعایت شود. همچنین، می‌توانید همه فایل‌های مورد نیاز در این مرحله را Bundle کنید. [از اینجا](#) می‌توانید پروژه بسیار ساده‌ای که در آن lazy loading پیاده شده است را دانلود کرده و مطالب توضیح داده شده را مشاهده کنید.

نظرات خوانندگان

نویسنده:

علی فخرائی

تاریخ:

۱۵:۵۲ ۱۳۹۳/۰۱/۱۰

با تشکر.

اگر ممکن است یک نمونه از تعریف دایرکتیو توسط lazy را هم بنویسید.

نویسنده:

حمید صابری

تاریخ:

۱۸:۲۲ ۱۳۹۳/۰۱/۱۰

سلام.

یک لینک به index.html اضافه کنید:

```
<div style="direction: rtl">
  <a href="#/state1">1 حالت</a> |
  <a href="#/state2">2 حالت</a> |
  <a href="#/state3">3 حالت</a>
  <div ui-view style="font-weight:bold; text-align:center;"></div>
</div>
```

فرض کنید محتویات مورد نظر برای این حالت که در فایل app/state3.html قرار دارد، شامل یک دایرکتیو است: state3.html:

تگ زیر یک دایرکتیو دارد:

```
<br/>
<div ng-hello-directive></div>
```

ng-hello-directive در فایل app/helloDirective.js به این صورت تعریف شده است:

```
angular.module('app').lazy.directive('ngHelloDirective', function () {
  return function (scope, elem, attr) {
    elem.html('سلام دایرکتیو تنبل!');
  };
});
```

و در نهایت حالت state3 را با آدرس /state3 در app.js تعریف کنید:

```
.state('state3', {
  url: '/state3',
  templateUrl: 'app/state3.html',
  resolve: {
    fileDeps: ['$q', '$rootScope', function ($q, $rootScope) {
      var deferred = $q.defer();
      var deps = ['app/helloDirective.js'];
      $script(deps, function () {
        $rootScope.$apply(function () {
          deferred.resolve();
        });
      });
      return deferred.promise;
    }]
  }
});
```

[از اینجا](#) می‌توانید پروژه مثال را که این دایرکتیو به آن افزوده شده دانلود کنید.

دقت کنید که در این حالت، این دایرکتیو تنها در ماژولی با نام app که خصوصییتی به نام lazy به صورت توضیح داده شده دارد ثبت می‌شود. اگر تابحال دایرکتیو آماده‌ای را دریافت کرده باشید، دیده‌اید که این دایرکتیوها به این صورت تعریف می‌شوند:

```
angular.module('moduleOfDirective', []).directive('ngDirectiveName', ...
```

همانطور که می‌بینید یک ماژول جدید تعریف شده و دایرکتیو در آن ثبت شده است. برای استفاده از چنین دایرکتیوی باید ماژول.

دایرکتیو را به وابستگی‌های ماژول خودتان اضافه کنید:

```
app = angular.module("app", ['ui.router', 'moduleOfDirective']);
```

در این حالت حتما باید فایل دایرکتیو را پیش از فایل app خود بارگذاری کرده باشید. یا اینکه تعریف دایرکتیو را تغییر دهید و بجای تعریف ماژول جدید، آن را به همان ماژول خودتان اضافه کنید. یعنی تعریف دایرکتیو را به این شکل تغییر دهید:

```
angular.module('app', []).lazy.directive('ngDirectiveName', ...
```

حالا این دایرکتیو را هم می‌توانید تنبلانه! بارگذاری کنید.

نویسنده: علی فخرائی
تاریخ: ۱۳۹۳/۰۱/۱۴ ۱۲:۴۸

بسیار ممنون جناب صابری.
شما در قسمت مسیریابی هم نام کنترلر را وارد کرده اید:

```
.state('state2', {
  url: '/state2',
  template: '<div>{{st2Ctrl.msg}}</div>',
  controller: 'state2Controller as st2Ctrl',
```

و هم فایل مرتبط را به عنوان وابستگی تعریف کرده اید:

```
var deps = ['app/messageService.js',
  'app/state2Controller.js'];
```

اما چرا محتوای کنترلر state2Controller.js دو بار اجرا میشود؟ یعنی با هر بار تغییر مسیر، 2 بار کل محتوای کنترلر اجرا میشود. مثلا اگر 1 تابع را در کنترلر صدا زده باشیم، این تابع 2 بار اجرا میشود.

نویسنده: ناصر طاهری
تاریخ: ۱۳۹۳/۰۱/۱۴ ۱۳:۳۹

چک کنید ببینید در قسمت کدهای HTML، ویژگی ای به نام ng-controller که به کنترلر شما اشاره کند وجود نداشته باشد

در یکی از پروژه‌هایی که دارم از AngularJS و ASP.NET MVC استفاده میکنم. به هنگام استفاده از درخواست‌های ایجکسی توسط سرویس \$http به مشکل عدم تشخیص ایجکسی بودن درخواست برخوردیم. توسط فیلتری که در [اینجا](#) توضیح داده شده و قرار دادن آن قبل از اکشن مورد نظر، میتوانیم تشخیص بدهیم که آیا درخواست رسیده از سمت کلاینت، ایجکسی است یا خیر؟ که در صورت ایجکسی نبودن درخواست، با صادر کردن یک استثنا مانع از اجرا شدن اکشن شویم. این فیلتر از اکستنشنی به نام IsAjaxRequest برای این تشخیص استفاده میکند:

```
HttpContext.Request.IsAjaxRequest();
```

اما هنگام استفاده از سرویس \$http، اکستنشن IsAjaxRequest() همیشه مقدار False را برمیگرداند. در حالیکه با متدهای ساده ایجکسی JQuery مثل \$.get و ...، مقدار این اکستنشن True میشود و به خوبی هم کار میکند. درخواست‌های هر دو مورد را که با فایرباگ بررسی کردم به این مقادیر برخوردیم. ویژگی‌های درخواست توسط JQuery - \$.get

Request Headers

```
Accept application/json, text/plain, */*
Accept-Encoding gzip, deflate
Accept-Language fa-ir,en-us;q=0.7,en;q=0.3
Cookie MyLanguageCookieName=fa-IR; __RequestVerificationToken=ZuW8imMSUSKOyBH2kih4oI
T38mP9854FoY2y817pXhqjOyO9eQ8VUTslN9A_ufopXaC0btSjSKb8A0auWz5hFd5qiaiVO7eZxYc
; captchastring=55-57-EB-27-6B-06-A7-C4-CC-93-9F-4A-C8-AF-28-63-93-9E-8E-6E-1
3B-7E-5F-1E-32-7D-66-30-3A-1C-FA-A2-DB-B4-43-10-BF-E7-02-56-CF-63-19-56-00-E4
A-2C-E8-45-B8-81-54-3B-60-84-12-9C-AA-19-4F-18-15-D2-82-5E-2D-02-46-C5-5E-D0-
; .403MyApp=CE5EF7C684F13C1FAA8DB8F8B43E14E47587D7E9C590EB0D492620F5E01213F72
0D6E99B1571F1CEC9E3B97BAF77F3FB5C95E92A7462B6490D6B5ECD4FF38F642DE82D86755364
1EF24201B7C6F3D56B9083A145D8F57B15FA3805E766CBA746A4FC16EA2A6726DC96
Host localhost:8417
Referer http://localhost:8417/administrator/dashboard
User-Agent Mozilla/5.0 (Windows NT 6.1; WOW64; rv:28.0) Gecko/20100101 Firefox/28.0
X-Requested-With XMLHttpRequest
```

و ویژگی‌های درخواست توسط سرویس \$http - AngularJS:

Request Headers

```

Accept application/json, text/plain, */*
Accept-Encoding gzip, deflate
Accept-Language fa-ir,en-us;q=0.7,en;q=0.3
Content-Length 112
Content-Type application/json;charset=utf-8
Cookie MyLanguageCookieName=fa-IR; __RequestVerificationToken=ZuW8imMSUSKOyBH2kih4
T38mP9854FoY2y817pXhqjOyO9eQ8VUTslN9A_ufopXaC0btSjSKb8A0auWz5hFd5qiaiVO7eZx
; captchastring=55-57-EB-27-6B-06-A7-C4-CC-93-9F-4A-C8-AF-28-63-93-9E-8E-6E
3B-7E-5F-1E-32-7D-66-30-3A-1C-FA-A2-DB-B4-43-10-BF-E7-02-56-CF-63-19-56-00-
A-2C-E8-45-B8-81-54-3B-60-84-12-9C-AA-19-4F-18-15-D2-82-5E-2D-02-46-C5-5E-D
; .403MyApp=CE5EF7C684F13C1FAA8DB8F8B43E14E47587D7E9C590EB0D492620F5E01213F
0D6E99B1571F1CEC9E3B97BAF77F3FB5C95E92A7462B6490D6B5ECD4FF38F642DE82D867553
; .ASPXROLES=pZCIOMMLs7NqH6S316M05D191zEnr47gvjGCS7ex068PRurznIYsonRQJZ-eAI
OyU0RQWwAYiw-fNGa6q6MK0zCso9uQWGiP-00aezj2yC9wV_xDYY40C0g8kE15yqMlhmMY4hmkW
U6-IlqkRNG9inON8e4ybE5Yc00uZbh-XN0GPgWPM1BvS4i2J3dcMn00af2-f4wPL4qvakegYDJC
kMR2Nax8Ku7HzV2JvpKcYpyfQLF-WMo19yK6fUKJwlTJXgWaZVzs4Jkq-GO9RmRuXBAsDXJTYmJ
MlyWpHICBwkvj_9hQy2Bdf2jggtYTVEE7XLNPw35nuiRUlgbkT3pCVd_Ipoa77aXXz0ClA1j1C-
aQWZq_IERhh-4ysOTa1TJgFP-AloFKsD6Tke3a5Q1zqvKQQIOY9SohCEI-NJSNp161VhpJBtPK
Host localhost:8417
Referer http://localhost:8417/administrator/dashboard
User-Agent Mozilla/5.0 (Windows NT 6.1; WOW64; rv:28.0) Gecko/20100101 Firefox/28.0

```



همینطور که میبینید، در هدر درخواست \$http یک مورد مفقود الاثر شده به نام X-Requested-With داریم و همین مقدار است که مشخص میکند این یک درخواست ایجکسی است یا خیر و اکستنشن IsAjaxRequest() نیز با همین مقدار عمل تشخیص را انجام میدهد. و به همین خاطر بود که این متد مقدار False را برمیگرداند.

بعد از کمی جستجو در این مورد، به [مخزن git](#) انگیلار رسیدم و به صراحت به این موضوع اشاره شده بود که این هدر به صورت پیشفرض از درخواست های \$http برداشته شده است.

بنابراین تنها راه حل این بود که خودمان به صورت دستی این هدر خاص رو به ماژول برنامه اضافه کنیم. به صورت زیر:

```

myAppModule.config(['$httpProvider', function($httpProvider) {
  $httpProvider.defaults.headers.common["X-Requested-With"] = 'XMLHttpRequest';
}]);

```

با اضافه کردن این هدر به درخواست های \$http، اکستنشن IsAjaxRequest() مقدار درست را برمیگرداند.

در این [پست](#) درباره به اشتراک گذاری داده ها بین کنترلرهای Angular بحث شد. اما استفاده از Factory و Service فقط زمانی کاربرد دارد که بخواهیم یک منبع داده مشخص را در اختیار مصرف کننده قرار دهیم. اگر قصد داشته باشیم بر اساس شرایط خاص، داده یا داده های مشخصی در سایر کنترلرها تغییر پیدا کنند چه باید کرد؟ به زبان ساده تر برای ایجاد ارتباط بین کنترلرها به طوری که از تغییرات یکدیگر باخبر باشند چه راهکارهایی وجود دارد. \$on و \$emit و \$broadcast برای این منظور تعبیه شده اند. برای شرح موارد بالا بهترین روش بررسی یک مثال است:

:\$emit

دو کنترلر به نام های FirstCtrl و SecondCtrl داریم. FirstCtrl به عنوان والد کنترلر Second است (در این مورد در این [پست](#) توضیح داده شده است).

پس فایل html نیز به صورت زیر خواهد بود:

```
<body ng-app>
  <div ng-controller="FirstCtrl">
    <p>{{title}}</p>
    <div ng-controller="SecondCtrl">
      <button ng-click="onUpdate()">Update First Ctrl Title</button>
    </div>
  </div>
</body>
```

قصد داریم با کلیک بر روی دکمه Update در کنترلر Second، مقدار خاصیت title در FirstCtrl به روز رسانی شود. اگر دقت کرده باشید حرکت رویداد از پایین به بالاست در نتیجه برای این کار باید از سرویس \$emit استفاده کنیم. کفایت در کنترلر دوم (Second) کد زیر را وارد نمایید:

```
function ChildCtrl($scope){
  $scope.onUpdate = function(){
    this.$emit("Update_Title", "Good Bye");
  };
}
```

به وسیله سرویس \$emit می توان از بروز یک رویداد در کنترلر جاری خبر داد. اگر کنترلر والد یک handler برای این رویداد داشته باشد، می تواند مقدار جدید را دریافت نماید. برای تعریف handler باید از سرویس \$on استفاده کرد. کدهای کنترلر First را به صورت زیر تغییر دهید.

```
function FirstCtrl($scope){
  $scope.title= "Hello";

  $scope.$on("Update_Title", function(event, message){
    $scope.title= message;
  });
}
```

«سرویس \$on برای تعریف handler برای رویداد مورد نظر استفاده می شود.

«پارامتر دوم سرویس \$on برابر با مقدار جدید ارسال شده توسط سرویس \$emit است.

«نام رویدادی که به عنوان پارامتر به \$on پاس داده می شود باید برابر با نام رویداد پاس داده شده به \$emit باشد.

«می توان چندین پارامتر را با استفاده از \$emit ارسال کرد و در سرویس \$on با تعریف متغیر به تعداد پارامترها مقادیر آنها را دریافت نمود.

\$broadcast

همان طور که مشاهده کردید SecondCtrl در محدوده FirstCtrl تعریف شده است. در نتیجه به راحتی با استفاده از سرویس \$emit توانستیم یک رویداد را منتشر نماییم. اما نکته مهم این است که اگر قصد داشته باشیم یک رویداد را از کنترلر والد (در این جا FirstCtrl است) منتشر نماییم به طوری که در کنترلرهای فرزند قابل دریافت باشد (حرکت رویداد بالا به پایین است)، باید از \$broadcast استفاده کنیم.

«\$broadcast فقط از نظر کاربرد با \$emit متفاوت است و در پیاده سازی کاملاً مشابه هستند.

یک مثال:

```
function ParentCtrl($scope){
    $scope.foo = "Hello";

    $scope.$on("UPDATE_PARENT", function(event, message){
        $scope.title= message;

        $scope.$broadcast("DO_BIDDING", {
            buttonTitle : "Taken over",
            onClick : function(){
                $scope.title= "HAHA this button no longer works!";
            }
        });
    });
}

function ChildCtrl($scope){
    $scope.buttonTitle = "Update Parent";
    $scope.onClick = function(){
        this.$emit("UPDATE_PARENT", "Updated");
    };

    $scope.$on("DO_BIDDING", function(event, data){
        for(var i in data){
            $scope[i] = data[i];
        }
    });
}
```

مشاهده خروجی مثال**اگر حالت فرزند و والد بین کنترلرها نباشد چه؟**

در این حالت باید \$rootScope را به کنترلر مورد نظر تزریق نمایید و سپس با استفاده از سرویس \$broadcast یا \$emit رویدادتان را منتشر کنید. مثال:

```
'use strict';
angular.module('myAppControllers', [])
.controller('FirstCtrl', function ($rootScope) {

    $rootScope.$broadcast('UPDATE_ALL');

    Or

    $rootScope.$emit('UPDATE_ALL');
});
```

نکته:

از آن جا که حرکت بالا به پایین event bubbling بسیار هزینه برتر است نسبت به حرکت پایین به بالا در نتیجه سعی کنید تا جای ممکن از \$rootScope.\$broadcast استفاده نکنید. در [این جا](#) توضیح کاملی درباره دلایل عدم استفاده از \$rootScope.\$broadcast داده شده است.

هم چنین می‌توانید یک مثال Live را نیز برای مقایسه بین \$emit و \$broadcast در [این جا](#) مشاهده کنید.

Testing in IE 10.0 32-bit on Windows Server 2008 R2 / 7 64-bit		
	Test	Ops/sec
\$broadcast	<code>window.\$rootScope.\$broadcast('fooHappened');</code>	26,406 ±9.58% 90% slower
\$emit	<code>window.\$rootScope.\$emit('fooHappened');</code>	250,522 ±7.21% fastest

در مطلب [آشنایی با Directive ها در AngularJS](#) با نحوه‌ی ایجاد Directive آشنا شدیم. هدف از این مطلب، آشنایی بیشتر با Directive در AngularJS است؛ یکی از بهترین فریم ورک‌های جاوااسکریپتی، با قابلیت ایجاد کتابخانه‌هایی از کامپوننت‌ها که می‌توانند به HTML اضافه شوند.

کتابخانه‌های جاوااسکریپتی زیادی وجود دارند. به عنوان مثال Bootstrap یکی از محبوب‌ترین "front-end framework" ها است که امکان تغییر در ظاهر المنت‌ها را فراهم می‌کند و شامل تعدادی کامپوننت جاوااسکریپتی نیز می‌باشد. مشکل کار، در هنگام استفاده از کامپوننت‌ها است. شخصی که در حال توسعه‌ی HTML است باید در کد جاوااسکریپتی خود از jQuery استفاده کند و بعنوان مثال یک Popover را فعال یا غیر فعال کند و این، یک فرآیند خسته کننده و مستعد خطا است.

یک مثال ساده از Directives AngularJS و بررسی آن

```
var m = angular.module("myApp");
myApp.directive("myDir", function() {
  return {
    restrict: "E",
    scope: {
      name: "@",
      amount: "=",
      save: "&"
    },
    template:
      "<div>" +
      "  {{name}}: <input ng-model='amount' />" +
      "  <button ng-click='save()'>Save</button>" +
      "</div>",
    replace: true,
    transclude: false,
    controller: [ "$scope", function ($scope) { ... } ],
    link: function (scope, element, attrs, controller) {...}
  }
});
```

به الگوی نامگذاری directive دقت کنید. پیشوند my شبیه به یک namespace است. بنابراین اگر یک Application از دایرکتیوهای قرار گرفته در Module های متفاوت استفاده کند، به راحتی می‌توان محل تعریف یک directive را تشخیص داد. این نام می‌تواند نشان دهنده‌ی این باشد که این directive را خودتان توسعه داده‌اید یا از یک directive توسعه داده شده توسط شخص دیگری در حال استفاده هستید. به هر حال این نحوه‌ی نام گذاری یک اجبار نیست و به عنوان یک پیشنهاد است.

سازنده directive یک شیء را با تعدادی خاصیت باز می‌گرداند که تمامی آنها در سایت AngularJS توضیح داده شده‌اند. در اینجا قصد داریم تا توضیحی مختصر در مورد کاری که این خصوصیات انجام می‌دهند داشته باشیم.

• **restrict** : تشخیص می‌دهد که آیا directive در HTML استفاده خواهد شد. گزینه‌های قابل استفاده 'C' ، 'E' ، 'A' برای attribute ، element ، class و یا comment است. پیش فرض 'A' برای attribute است. اما ما بیشتر علاقه به استفاده از ویژگی element برای ایجاد المنت‌های UI داریم.

• **scope** : ایجاد یک scope ایزوله که متعلق به directive است و موجب ایزوله شدن آن از scope صدا زننده directive می‌شود. متغیرهای scope پدر از طریق خصوصیات تگ directive ارسال می‌شوند. این ایزوله کردن زمانی کاربردی است که در حال ایجاد کامپوننت‌هایی با قابلیت استفاده مجدد هستیم، که نباید متکی به scope پدر باشند. شیء scope در directive نام و نوع

متغیرهای scope را تعیین می‌کنند. در مثال بالا سه متغیر برای scope تعریف شده است:

- **(name: "@" (by value, one-way** : علامت @ مشخص می‌کند که مقدار متغیر ارسال می‌شود. Directive یک رشته را دریافت می‌کند که شامل مقدار ارسال شده از scope پدر می‌باشد. Directive می‌تواند از آن استفاده کند، اما نمی‌تواند مقدار آن را در scope پدر تغییر دهد.

- **(amount: "=" (by reference, two-way** : علامت = مشخص می‌کند این متغیر با ارجاع ارسال می‌شود. Directive یک ارجاع به مقدار متغیر در scope اصلی دریافت می‌کند. مقدار می‌تواند هر نوع داده ای، شامل یک شیء complex یا یک آرایه باشد. Directive می‌تواند مقدار را در scope پدر تغییر دهد. این نوع متغیر، زمانی که نیاز باشد directive مقدار را در scope پدر تغییر دهد، استفاده می‌شود.

- **(save: "&" (expression** : علامت & مشخص می‌کند این متغیر یک expression را که در scope پدر اجرا می‌شود، نگهداری می‌کند. اکنون directive قابلیت انجام کارهایی فراتر از تغییر یک مقدار را دارد. به عنوان مثال می‌توان یک تابع را از scope پدر فراخوانی و نتیجه‌ی اجرا را دریافت کرد.

· **template** : الگوی رشته ای که جایگزین المنت تعریف شده می‌شود. فرآیند جایگزینی تمامی خصوصیات را از المنت قدیمی به المنت جدید انتقال می‌دهد. به نحوه استفاده از متغیرهای تعریف شده در scope ایزوله دقت کنید. این مورد به شما امکان تعریف directive های macro-style را می‌دهد که نیاز به کد اضافی، ندارند. اگرچه در بیشتر موارد الگو یک تگ ساده <div> است که از کدهای **link** که در زیر توضیح داده شده است استفاده می‌کند.

· **replace** : تعیین می‌کند که آیا الگوی directive باید جایگزین المنت شود. مقدار پیش فرض false است.

· **transclude** : تعیین کننده این است که محتوای directive باید در المنت کپی شود یا خیر. در مثال زیر المنت tab شامل المنت‌های HTML دیگر است پس transclude برابر true است.

```
<body ng-app="components">
  <h3>Bootstrap Tab Component</h3>
  <tabs>
    <pane title="First Tab">
      <div>This is the content of the first tab.</div>
    </pane>
    <pane title="Second Tab">
      <div>This is the content of the second tab.</div>
    </pane>
  </tabs>
</body>
```

· **link** : این تابع بیشتر منطق directive را شامل می‌شود. Link وظیفه دستکاری DOM، ایجاد event listener ها و... را دارد. تابع Link پارامترهای زیر را دریافت می‌کند:

- **scope** : ارجاع به scope ایزوله شده directive دارد.

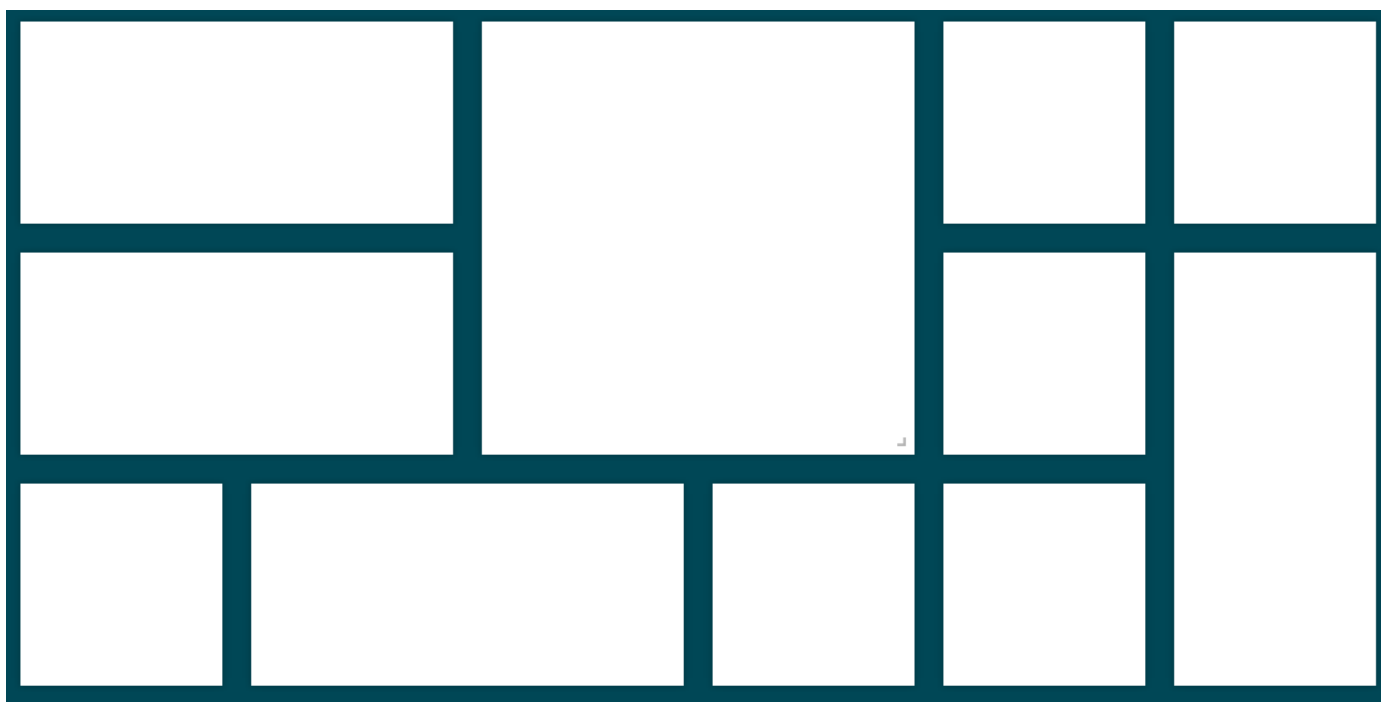
- **element** : ارجاع به المنت‌های DOM که directive را تعریف کرده اند. تابع link معمولاً برای دستکاری المنت از jQuery استفاده می‌کند. (یا از Angular's **jqLite** در صورتی که jQuery بارگذاری نشده باشد)

- **controller** : در مواقعی که از دایرکتیوهای تو در تو استفاده می‌شود کاربرد دارد. این پارامتر یک directive فرزند با ارجاعی به پدر را فراهم می‌کند، بنابراین موجب ارتباط directive ها می‌شود.

به عنوان مثال، [این directive](#) که پیاده سازی bootstrap tab را انجام داده است، می‌توانید مشاهده نمایید.

موفق باشید

حتما تا به حال در وب سایت‌های زیادی قسمت هایی را دیده اید که چیدمان عناصر آن به شکل زیر است:



این گونه چیدمان را حتما در منوی Start ویندوز 8 بارها دیده‌اید! عناصر تشکیل دهنده‌ی این شکل از چیدمان، می‌توانند یک سری عکس باشند که تشکیل یک گالری عکس را داده‌اند و یا یک سری div که محتوای پست‌های یک وبلاگ را در خود جای داده‌اند. چیزی که این شکل از چیدمان عناصر را نسبت به چیدمان‌های معمول متمایز می‌کند این است که طول و عرض هر یک از این عناصر با یکدیگر متفاوت است و هدف از این گونه چیدمان آن است که این عناصر در فضایی که به آن‌ها اختصاص داده شده است، به صورت بهینه قرار گیرند تا کمترین فضا هدر رود.

برای اعمال این شکل از چیدمان در دنیای وب افزونه‌های زیادی بر فراز کتابخانه‌ی jQuery تدارک دیده شده است که از جمله مطرح‌ترین آن‌ها می‌توان به افزونه‌های [Masonry](#)، [Isotope](#) و [Gridster](#) اشاره کرد.

افزونه‌ی Isotope مزایایی را برای من در پی داشت و این افزونه را برای انجام کارهای خود، مناسب دیدم. نکته‌ی مهم اینجا است که هدف من بررسی Isotope نیست، چرا که اگر به وب سایت آن مراجعه کنید، با کوهی از مستندات مواجه می‌شوید که چگونه از آن در وب سایت‌های معمولی استفاده کنید.

در این مقاله قصد من این است که نشان دهم چگونه از افزونه‌ی Isotope در AngularJS استفاده کنیم؛ چگونه چیدمان آن را راست به چپ کنیم و چگونه آن را با محیط‌های واکنش گرا (Responsive) سازگار کنیم.

فرض کنید در یک وب سایت قصد داریم اطلاعات یک سری مطلب خبری را از سرور، به فرمت JSON دریافت کرده و نمایش دهیم. در AngularJS شیوه‌ی کار بدین صورت است که اطلاعاتی که به فرمت JSON هستند را با استفاده از directive ایی به نام

ng-repeat پیمایش کرده و آن‌ها را نمایش دهیم. حال اگر بخواهیم چیدمان مطالب را با استفاده از Isotope تغییر دهیم، می‌بینیم که هیچ چیزی نمایش داده نمی‌شود. دلیل آن بر می‌گردد به مراحل کامپایل کردن AngularJS و نامشخص بودن زمان اعمال چیدمان Isotope به عناصر است.

در AngularJS هنگامیکه با دستکاری DOM سر و کار پیدا می‌کنیم، معمولاً باید به سراغ Directive‌ها رفت و یک Directive سفارشی برای کار با Isotope تعریف کرد تا با مکانیزم‌های Angular سازگار باشد. خوشبختانه [Isotope Directive برای Angular](#) موجود می‌باشد. نکته‌ی مهم این است که این Directive برای نگارش 1 افزونه‌ی Isotope نوشته شده است. البته با نگارش 2 هم کار می‌کند که من برای انجام کار خود نسخه‌ی 1 را ترجیح دادم استفاده کنم.

نکته‌ی بعدی که باید رعایت شود این است که چیدمان عناصر باید از راست به چپ شوند. خوشبختانه این کار در نسخه‌ی 1 Isotope با [تغییر کوچکی در سورس Isotope](#) و تغییر یک تابع انجام می‌شود. گویا نسخه‌ی دوم [امکان پیش فرضی](#) را برای این کار دارد، اما نتوانستم آن را به خوبی پیاده سازی کنم و به همین دلیل ترجیح دادم از همان نسخه‌ی اول استفاده کنم.

برای اینکه در هنگام جابه جا شدن عناصر، انیمیشن‌ها نیز از راست به چپ انجام شوند، باید css‌های زیر را نیز اعمال نمود:

```
.isotope .isotope-item {
  -webkit-transition-property: right, top, -webkit-transform, opacity;
  -moz-transition-property: right, top, -moz-transform, opacity;
  -ms-transition-property: right, top, -ms-transform, opacity;
  -o-transition-property: right, top, -o-transform, opacity;
  transition-property: right, top, transform, opacity;
}
```

Responsive بودن این عناصر مسئله‌ی دیگری است که باید حل گردد. امروزه اکثر فریم ورک‌های مطرح css، واکنشگرا نیز هستند و برای پشتیبانی از سایزهای متفاوت صفحه نمایش، تدابیری در نظر گرفته‌اند. اساس کار واکنش گرا بودن این فریم ورک‌ها در تعیین ابعاد عناصر، بیان ابعاد به صورت درصدی است. مثلاً فلان عرض div برابر 50% باشد بدین معناست که همیشه عرض این div نصف عرض عنصر والد آن باشد.

متأسفانه Isotope میانه‌ی چندانی با این ابعاد درصدی ندارد و باید عرض عناصر به صورت دقیق و بر حسب پیکسل بیان شود. البته نسخه‌ی جدید آن و یا حتی پلاگین‌هایی برای کار با ابعاد درصدی نیز تدارک دیده شده است که به شخصه به نتیجه‌ی با کیفیتی نرسیدم.

برای حل این مشکل می‌توان از امکانات CSS به مانند دستورات زیر استفاده کرد:

```
@media (min-width: 768px) and (max-width: 980px) {
  .card {
    width: 320px;
  }
}

@media (min-width: 980px) and (max-width: 1200px) {
  .card {
    width: 260px;
  }
}

@media (min-width: 1200px) {
  .card {
    width: 340px;
  }
}
```

بدین صورت می‌توان در ابعاد مختلف نمایشگر تعیین کرد که عرض عناصر ما چقدر باشد.

اکنون یک گالری عکس را در نظر بگیرید که در زیر هر عکس توضیحی نیز نوشته شده است و ساختار HTML آن به این صورت است که داخل هر div عکسی نیز موجود است. اگر به شیوه‌ی ذکر شده عمل کنید با یک اشکال مواجه می‌شوید و عناصر روی هم قرار گرفته و اصطلاحاً overlapping می‌افتد. دلیل این امر این است که لود شدن عکس‌ها عملی زمان گیر است و

Isotope قبل از این که عکس لود شود، سایز آن عنصر را محاسبه کرده که در حقیقت این سایز بدون احتساب سایز عکس است و ابعاد واقعی عنصر ما نیست؛ در نتیجه وقتی عکس لود می‌شود آن div فضای بیشتری احتیاج دارد و به همین دلیل به زیر divهای دیگر می‌رود.

برای حل این مشکل باید به این صورت عمل کرد که وقتی عکس‌ها کامل لود شدند، Isotope وارد عمل شده و سایز عناصر را به دست آورده و آن‌ها را بچیند. برای این کار معمولا از افزونه‌ی [imagesLoaded](#) استفاده می‌کنند که با کمک این افزونه می‌توان مشخص کرد که وقتی تمام عکس‌های موجود در فلان div کامل لود شدند، Isotope وارد عمل شده و عناصر را چیدمان کند. البته بدون استفاده از افزونه‌ی imagesLoaded و به کمک امکانات AngularJS و تعریف یک Directive سفارشی می‌توان زمان لود شدن عکس‌ها را کنترل کرد.

```
app.directive('imageOnload', function () {
    return {
        restrict: 'A',
        link: function (scope, element, attrs) {
            element.bind('load', function () {
                scope.$emit('iso-method', { name: 'reLayout', params: null }); // call reLayout
            });
            isotope method prevent overlaaping the items
        }
    };
});
```

کار این directive این است که به ازای بارگذاری هر عکس، متد reLayout را از Isotope، فراخوانی می‌کند. از این جهت فراخوانی reLayout به ازای لود شدن هر عکس بهتر است که لود شدن تمامی عکس‌ها ممکن است مدت زمان زیادی طول بکشد و کاربر برای مدتی با یک ساختار بهم ریخته مواجه شود.

اگر در نمونه کدی که قرار داده‌ام، به انتهای کدهای کنترلر ListController دقت کنید، برای رویداد resize شی window، تابعی تعریف شده است تا به هنگام تغییر سایز صفحه فراخوانی شود. در این رویداد هر بار که سایز پنجره تغییر کرد، پس از یک ثانیه تابع reLayout افزونه‌ی Isotope را فراخوانی می‌کنیم تا مجدداً المنت‌های صفحه چیده شوند. البته ضرورتی وجود نداشته ولی در بعضی مواقع عناصر خوب چیده نمی‌شدند که با فراخوانی reLayout از چیدمان صحیح عناصر مطابق با سایز جدید صفحه اطمینان حاصل پیدا می‌کنیم. دلیل یک ثانیه تأخیر این است که اگر به ساز و کار تعاریف متدها در directive Isotope دقت کنید، از سرویس \$timeout به وفور استفاده شده است. ظاهراً اگر برای فراخوانی reLayout زودتر عمل کنیم با فراخوانی‌هایی این متد در ساختار خودش تداخل پیدا می‌کند.

```
$(window).resize(function () {
    $timeout(function myfunction() {
        $scope.$broadcast('iso-method', { name: 'reLayout', params: null }); // call
        reLayout isotope method prevent overlaaping the items
    },1000);
});
```

در نهایت تمامی نکات گفته شده را به صورت یک نمونه کد آماده کردم: [دانلود نمونه کد](#)

0-گوگل قصد ندارد تولید نکسوس ها را متوقف کند



طنی ماهیهای اخیر بارها شاهد انتشار شایعاتی مبنی بر عدم تمایل گوگل نسبت به تولید و توسعه ابزارهای سری نکسوس بودیم. ابزارهایی که اندروید خالص بر روی آن‌ها نصب شده و به سفارش گوگل توسط شرکای سخت‌افزاری این کمپانی تولید می‌شوند. حال گوگل این شایعات را بطور کامل رد کرده و گفته است که تولید نکسوس‌ها را ادامه خواهد داد.

1-چگونه اندروید L را روی نکسوس 5 یا نکسوس 7 نصب کنیم؟



طنی ماهیهای اخیر بارها شاهد انتشار شایعاتی مبنی بر عدم تمایل گوگل نسبت به تولید و توسعه ابزارهای سری نکسوس بودیم. ابزارهایی که اندروید خالص بر روی آن‌ها نصب شده و به سفارش گوگل توسط شرکای سخت‌افزاری این کمپانی تولید می‌شوند. حال گوگل این شایعات را بطور کامل رد کرده و گفته است که تولید نکسوس‌ها را ادامه خواهد داد.

3-شاتر: تصاویری دیدنی از احساسات والدین در قلمرو حیوانات



طنی ماهیهای اخیر بارها شاهد انتشار شایعاتی مبنی بر عدم تمایل گوگل نسبت به تولید و توسعه ابزارهای سری نکسوس بودیم. ابزارهایی که اندروید خالص بر روی آن‌ها نصب شده و به سفارش گوگل توسط شرکای سخت‌افزاری این کمپانی تولید می‌شوند. حال گوگل این شایعات را بطور کامل رد کرده و گفته است که تولید نکسوس‌ها را ادامه خواهد داد.

2-مقایسه‌ی تصویری اندروید L با اندروید کیتکت



طنی ماهیهای اخیر بارها شاهد انتشار شایعاتی مبنی بر عدم تمایل گوگل نسبت به تولید و توسعه ابزارهای سری نکسوس بودیم. ابزارهایی که اندروید خالص بر روی آن‌ها نصب شده و به سفارش گوگل توسط شرکای سخت‌افزاری این کمپانی تولید می‌شوند. حال گوگل این شایعات را بطور کامل رد کرده و گفته است که تولید نکسوس‌ها را ادامه خواهد داد.

4-اپل قیمت آپل تاج را ضمن مجهز کردن نسخه‌ی 16 گیگابایتی به دوربین کاهش داد



5-تماشا کنید: بازگرداندن توانایی کنترل ماهیچه‌ها به افراد فلج به کمک فناوری نوروبریج

در برنامه‌های مبتنی بر وب رایج، معمولاً تبدیل تاریخ میلادی به شمسی در سمت سرور انجام می‌گیرد و تاریخ شمسی حاصل از تبدیل، به کاربر نمایش داده می‌شود. اما در برنامه‌های Single Page و یا به اختصار SPAها که کلاینت فقط با یک سری داده به فرمت JSON درگیر است، برای نمایش تاریخ شمسی به چه طریقی باید عمل کرد؟ آیا باید تاریخ را در سمت سرور به فرمت مورد نظر تبدیل کرد و یا در سمت کلاینت؟ همه‌ی این‌ها از جمله سوالاتی هست که به هنگام توسعه‌ی SPAها با آن‌ها حتماً درگیر خواهید شد.

شاید بتوان گفت که در SPAها، هدف این است که از بار سرور تا حد ممکن کم کرد و آن را در بین کلاینت‌ها توزیع کرد. در SPAها نقش اصلی سرور تامین داده هاست و بیشتر پردازش‌ها در صورت امکان در سمت کلاینت انجام می‌شود و می‌بینید که حتی رندر کردن HTML نیز به عهده‌ی قالب‌های سمت کلاینت است. البته هنوز هم می‌توان قبل از اینکه داده را به فرمت JSON سریالایز کرد، سمت سرور بر روی آن‌ها پیمایش انجام داده و تاریخ‌های میلادی را به شمسی تبدیل کرد که هدف ما این نیست و می‌خواهیم این کار را بر عهده‌ی مرورگر کاربر قرار دهیم. معرفی **moment.js**

برای کار با داده‌هایی از جنس تاریخ در سمت کلاینت، کتابخانه‌ی جاوا اسکریپتی قدرتمندی به نام **moment.js** وجود دارد. این کتابخانه دارای انواع و اقسام API برای نمایش و پردازش تاریخ هست. حتی می‌تواند relative time را نیز نمایش دهد. منظور از relative time این هست که به جای نمایش تاریخ، اختلاف آن را با زمان حال نمایش دهد. برای مثال می‌نویسند فلان پست در دو ساعت پیش ارسال شده و زمان دقیق ارسال پست را نمایش نمی‌دهد.

خوشبختانه برای افزودن تاریخ شمسی به این کتاب خانه، افزونه‌ای به نام **moment-jalaali** برای آن تدارک دیده شده است. کار با آن نیز بسیار راحت است. کافی است در همان APIهایی که برای فرمت کردن تاریخ در **moment.js** استفاده می‌کردید؛ یک ژ در ابتدای آن‌ها قرار دهید که مثال‌های کامل استفاده از آن را در مستندات آن می‌توانید مشاهده کنید.

نحوه‌ی استفاده از moment.js در AngularJS و ASP.NET

در ASP.NET فیلد هایی که از جنس DateTime هستند به شکل زیر به فرمت JSON سریالایز می‌شوند:

```
\Date(1374222094520)\
```

در **moment.js** احتیاج به کدنویسی برای parse کردن این نوع فرمت و تبدیل کردن آن به تاریخ وجود ندارد؛ چرا که **moment.js** به صورت **تو کار** از این نوع فرمت نیز پشتیبانی می‌کند و احتیاجی به کار اضافه‌تر نیست.

```
moment("/Date(1198908717056-0700)/"); // December 28 2007 10:11 PM
```

در AngularJS هر گاه قصد داشته باشیم که فرمت نمایش داده‌ها را تغییر دهیم از **filter**ها استفاده می‌کنیم. برای مثال فیلتر **uppercase** داده **name** را با حروف بزرگ نمایش می‌دهد.

```
{{ name | uppercase }}
```

حال برای تاریخ نیز می‌خواهیم چنین کاری انجام دهیم؛ بدین صورت که یک فیلتر سفارشی به شکل زیر تعریف کرده تا تاریخ میلادی را به صورت شمسی و با فرمت دلخواهی که می‌خواهیم نمایش دهد:

```
{{post.date | jalaliDate:'jYYYY/jMM/jDD hh:mm' }}
```

تعریف فیلتر **jalaliDate** نیز به شکل زیر است:

```
app.filter('jalaliDate', function () {
    return function (inputDate, format) {
        var date = moment(inputDate);
        return date.fromNow() + " " + date.format(format);
    };
});
```

```
}  
});
```

خروجی این فیلتر نیز به شکل "4 ماه پیش 03:10 07/12/1392" است و مشاهده می‌کنید که به کمک filterها در AngularJS انجام این گونه از کارها بسیار ساده و لذت بخش است.

توجه کنید که این فقط یک ایده‌ی ابتدایی و ساده از پیاده سازی فیلتر فوق است. قطعا با کمک APIهای متنوع momentjs و پارامترهای ورودی فیلتر، می‌توان فیلتری بسیار پیشرفته‌تر تعریف کرد. دریافت کدهای یک مثال پیاده سازی شده با استفاده از [moment-jalali-AngularJs.rar](#) کدهای فوق

نظرات خوانندگان

نویسنده: محمد 92
تاریخ: ۱۳۹۳/۱۰/۱۰ ۷:۵۳

با سلام و ممنون از مقاله خوبتون
سوالی داشتم، [moment-jalaali](#) روی nuget نیست یا من نتونستم پیدااش کنم

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۱۰/۱۰ ۱۶:۶

- کتابخانه‌های جاوا اسکریپتی را بهتر است [در Bower](#) و امثال آن دنبال کنید.
- برای نمونه [آدرس نصب](#) moment-jalaali ([^](#))

```
npm install moment-jalaali
```

- ویزوال استودیو هم افزونه‌ای برای کار با Bower ارائه داده. [اطلاعات بیشتر](#)

یک نکته‌ای که در توسعه سیستم‌ها و نرم افزارها تاکید فراوانی به آن می‌شود استفاده مجدد از کدهای نوشته شده قبلی است. یعنی تا جای ممکن باید ساختار پروژه به گونه‌ای نوشته شود که از تکرار کدها در جای جای پروژه جلوگیری شود. این مورد به خوبی در زبان‌های شیء‌گرا نظیر C# رعایت می‌شود اما در پروژه‌هایی که مبتنی بر Javascript هستند نظیر angular، باید با استفاده از خاصیت prototype جاوا اسکریپت این مورد را رعایت نمود. در [مقاله](#) Dr. Axel Rauschmayer، قدم به قدم و به خوبی روش‌های وراثت در Javascript توضیح داده شده است.

در [این پست](#) با روش‌های وراثت در کنترلرهای انگولار آشنا شدید. این وراثت محدود به ارث بری scope می‌شود. اما یکی از بخش‌های بسیار مهم پروژه‌های انگولار نوشتن سرویس‌هایی با قابلیت توسعه مجدد در سایر بخش‌های پروژه می‌باشد. معادل آن، مفهوم Overriding در OOP است. با ذکر مثالی این مورد را با هم بررسی خواهیم کرد. ابتدا یک سرویس به نام BaseService ایجاد کنید:

```
angular.module('myApp').service('BaseService', function() {
    var BaseService = function(title) {
        this.title = title;
    };

    BaseService.prototype.getMessage = function() {
        var self = this;
        return 'Hello ' + self.title;
    };

    return BaseService;
});
```

سرویس بالا دارای سازنده‌ای است که مقدار title باید در اختیار آن قرار گیرد. با استفاده از خاصیت prototype تابعی تعریف می‌کنیم که این تابع خروجی مورد نظر را برای ما تامین خواهد نمود. حال اگر ماژول و کنترلری جهت نمایش خروجی به صورت زیر ایجاد کنیم:

```
var app= angular.module('myApp', []);

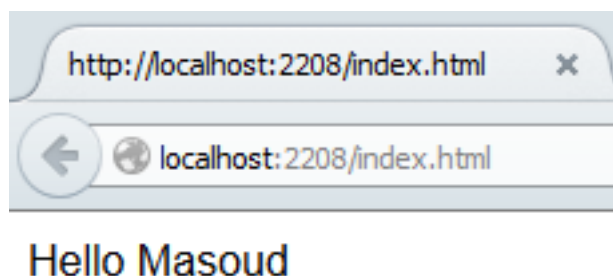
app.controller('myCtrl', function ($scope,BaseService) {
    var instance = new BaseService('Masoud');
    $scope.title = instance.getMessage();
});
```

با کدهای Html زیر:

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" ng-app="myApp">
<head>
    <title></title>
</head>
<body ng-controller="myCtrl">
    <div>
        {{title}}
    </div>
</body>
<script src="Scripts/jquery-2.1.1.min.js"></script>
<script src="Scripts/angular.js"></script>
<script src="App/app.js"></script>
```


</html>

در نهایت خروجی به صورت زیر قابل مشاهده است:



تا اینجا کار روال معمول تعاریف سرویس در انگولار بوده است. اما قصد داریم سرویس جدیدی را ایجاد نمایم تا خروجی سرویس قبلی را اندکی تغییر دهد. به جای اینکه سرویس قبلی را تغییر دهیم یا بدتر از آن سرویس جدیدی بسازیم و کدهای قبلی را در آن کپی کنیم کافیهست به صورت زیر عمل نماییم:

```
app.service('ExtService', function(BaseService) {
    var ExtService = function() {
        BaseService.apply(this, arguments);
    };

    ExtService.prototype = new BaseService();

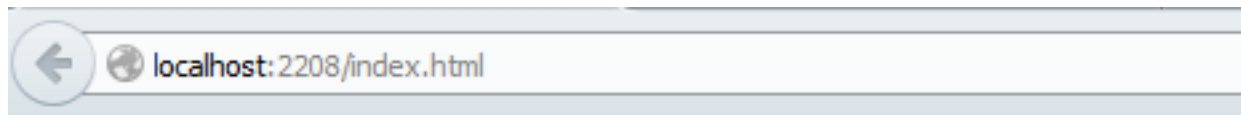
    ExtService.prototype.getMessage = function() {
        var self = this;
        return BaseService.prototype.getMessage.apply(this, arguments) + ' From Ext Service';
    };

    return ExtService;
});
```

حال می‌توان کنترلر را به صورت زیر بازنویسی کرد.

```
app.controller('myCtrl', function ($scope, BaseService , ExtService) {
    var baseInstance = new BaseService('Masoud');
    var extInstance = new ExtService('Dotnettips');
    $scope.title = baseInstance.getMessage() + ' and ' + extInstance.getMessage();
});
```

در کنترلر بالا هر دو سرویس تزریق شده‌اند. خروجی سرویس دوم متن From Ext Service را نیز به همراه خواهد داشت. پس از اجرای برنامه خروجی زیر قابل مشاهده است:



Hello Masoud and Hello Dotnettips From Ext Service

تا پیش از این به احتمال زیاد با `Interceptor` ها در `IOC Container` ها متفاوت آشنا شدید و برای `AOP` از آن‌ها استفاده کرده‌اید. در این جا نیز دقیقاً همان مفهوم و هدف را دنبال خواهیم کرد؛ اضافه کردن و تزریق کدهای نوشته شده به منطق برنامه. کاربرد `Interceptor` ها در انگولار، زمانی است که قصد داشته باشیم یک سری تنظیمات عمومی را برای درخواست‌های `$http` انجام دهیم. همچنین می‌توان انجام برخی مراحل مشترک، نظیر اعتبارسنجی یا مدیریت خطاها را نیز توسط `Interceptor` ها انجام دهیم. سرویس `$http` در `Angular` جهت ارتباط و تبادل اطلاعات با دنیای `Backend` مورد استفاده قرار می‌گیرد. حالت‌هایی بنابر نیاز به وجود می‌آیند که بخواهیم ارسال اطلاعات به سرور و هم چنین پاسخ دریافتی را `capture` کنیم و قبل از این که داده‌ها در اختیار `App` قرار گیرد، آن را مورد بررسی قرار دهیم (برای مثال لاگ اطلاعات) یا حتی نوشتن یک `HTTP error handling` جهت مدیریت خطاهای به وجود آمده حین ارتباط با سرور (برای مثال خطای 404). حال با ذکر مثالی این موارد را بررسی می‌کنیم. برای نوشتن یک `Interceptor` می‌توان با استفاده از سرویس `factory` این کار را به صورت زیر انجام داد.

```
module.factory('myInterceptor', ['$log', function($log) {
    $log.debug('data');

    var myInterceptor = {
        ....
        ....
        ....
    };

    return myInterceptor;
}]);
```

کد بالا یک `Interceptor` بسیار ساده است که وظیفه آن لاگ اطلاعات است. در انگولار چهار نوع `Interceptor` برای سرویس `$http` داریم:

« **request** : قبل از هر فراخوانی سرویس‌های سمت سرور، ابتدا این `Interceptor` فراخوانی می‌شود و `config` سرویس `$http` در اختیار آن قرار می‌گیرد. می‌توان این تنظیمات را با توجه به نیاز، تغییر داد و نمونه ساخته شده جدید را در اختیار سرویس `$http` قرار دهیم.

« **response** : هر زمان که عملیات فراخوانی سرویس‌های سمت سرور به درستی انجام شود و همراه با آن پاسخی از سرور دریافت شود، این `Interceptor` قبل از فراخوانی تابع `success` سرویس `$http`، اجرا خواهد شد.

« **requestError** : از آنجا که سرویس `$http` دارای مجموعه‌ای از `Interceptor` ها است و آن‌ها نیز یکی پس از دیگری حین انجام عملیات اجرا می‌شوند، اگر در `Request Interceptor` قبلی خطایی رخ دهد بلافاصله این `Interceptor` فراخوانی می‌شود.

« **responseError** : درست مانند حالت `requestInterceptor` است؛ فقط خطای مربوطه باید در تابع `response` باشد.

با توجه به توضیحات بالا کد قبلی را به صورت زیر تعمیم می‌دهیم.

```
module.factory('myInterceptor', ['$q', '$log', function($q, $log) {
    $log.debug('data');

    return {
        request: function(config) {
            return config || $q.when(config);
        },
        requestError: function(rejection) {
```

```
return $q.reject(rejection);
},
response: function(response) {
return response || $q.when(response);
},
responseError: function(rejection) {
return $q.reject(rejection);
}
}
}]]);
```

برای رجیستر کردن Interceptor بالا به سرویس‌های \$http باید به صورت زیر عمل نمود.

```
angular.module('myApp')
.config(function($httpProvider) {
$httpProvider.interceptors.push('myInterceptor');
});
```

در Angular می شود یک سری Template و ساختار از پیش تعریف شده داشت و در هر زمان که نیاز بود مدلی را به آنها پاس داد و نمای HTML مورد نظر را تحویل گرفت. بطور مثال در فرم سازها یا همان فرم‌های داینامیک ما نیاز داریم که مدل یک فرم (مثلا در فرمت JSON) را برای View ارسال کنیم و با استفاده از توانایی‌های Angular بتوانیم فرم مورد نظر را نمایش دهیم و در صورت امکان تغییر دهیم. ViewModel فرم شما در MVC میتواند چیزی شبیه این باشد

```
public class Form
{
    public string Name { get; set; }
    public string Title { get; set; }
    public List<BaseElement> Elements { get; set; }
}

public abstract class BaseElement
{
    public string Name { get; set; }
    public string Title { get; set; }
}

public class Section : BaseElement
{
    public List<TextBox> Elements { get; set; }
}

public class TextBox : BaseElement
{
    public string Value { get; set; }
    public string CssClass { get; set; }
}
```

یک کنترلر هم برای مدیریت فرم ایجاد میکنیم

```
public class FormBuilderController : Controller
{
    //
    // GET: /FormBuilder/

    public ActionResult Index()
    {
        var form = new Form();
        var section = new Section() { Title = "Basic Info", Name = "section01" };
        section.Elements.Add(new TextBox() { Name = "txt1", Title = "First Text Box" });
        form.Elements.Add(new TextBox() { Name = "txt1", Title = "Second Text Box" });
        var formJson = JsonConvert.SerializeObject(form);
        return View(formJson);
    }
}
```

در این کنترلر ما تنها یک اکشن داریم که در آن یک فرم خام ساده ایجاد کرده و سپس با استفاده از کتابخانه Json.net آنرا سریال و تبدیل به فرمت Json می‌کنیم و سپس آنرا برای View ایی که از Angular قدرت گرفته است، ارسال می‌نمائیم. پیاده سازی View با Angular به اشکال گوناگونی قابل پیاده سازی و استفاده است که در [اینجا](#) و [اینجا](#) می‌توانید ببینید. اما برای اینکه مشکل کنترلرهای تودرتو (Section) را حل کنید باید بصورت بازگشتی Template را فراخوانی کنید.

```
<script type="text/ng-template" id="ElementTemplate">
    <div ng-if="control.Type == 'JbSection'">
        <h2>{{control.Title}}</h2>
        <ul>
            <li ng-repeat="control in control.Elements" ng-include="'ElementTemplate'"></li>
        </ul>
    </div>
</script>
```

```
<script type="text/ng-template" id="element.html">
  {{data.label}}
  <ul>
    <li ng-repeat="element in data.elements" ng-include="'element.html'"></li>
  </ul>
</script>

<ul ng-controller="NestedFormCtrl">
  <li ng-repeat="field in formData" ng-include="'element.html'"></li>
</ul>
```

در اینجا صفحه element.html یک صفحه بیرونی است که Template ما در آن قرار دارد.

انگیزه اصلی این نوشته شروع کار با AngularJS و استفاده از scope در این کتابخانه است. بیشتر دوستانی که کار با این کتابخانه را شروع می‌کنند و تجربه زیادی با جاوا اسکریپت ندارند، با مفهوم ارث بری scope مشکل پیدا می‌کنند.

ارث بری در scope های AngularJS موضوع پیچیده و عجیب و غریبی نیست. در واقع همان ارث بری prototype ای است که جاوا اسکریپت پشتیبانی می‌کند.

این روش توضیح خیلی ساده ای دارد.

در هنگام دسترسی به مقدار یک خصوصیت روی یک شی اگر آن خصوصیت در شی مورد نظر وجود نداشته باشد جاوا اسکریپت یک سطح در زنجیره‌ی prototype ها بالا رفته و به شی پدر دسترسی پیدا کرده و در آن به دنبال مقدار خصوصیت می‌گردد. این کار را آن قدر ادامه می‌دهد تا به بالاترین سطح برسد و دیگر چیزی پیدا نکند.

این بالا رفتن در زنجیره‌ی prototype ها عملاً با دسترسی به خصوصیت prototype انجام می‌شود.

فرض کنید دو شی (دقت کنید که می‌گوییم شی) به نام‌های employee و person داریم. این دو شی را به صورت زیر تعریف می‌کنیم.

```
var person = { type: '', name: 'No Name' };
var employee = { };
```

شی employee الان هیچ خصوصیت ای ندارد. و دسترسی به هر خصوصیت ای از آن هیچ نتیجه‌ای در بر نخواهد داشت.

```
console.log('Before Inheritance -> employee.name = ' + employee.name);
```

با مقدار دهی کردن خصوصیت prototype مربوط به employee به person این شی را از person ارث بری می‌کنیم.

```
employee.__proto__ = person;
```

بعد از اجرا شدن این خط از برنامه هنگام دسترسی پیدا کردن به مقدار name، مقدار اصلی آن که در شی person وارد شده بود را خواهیم دید.

ملاحظه کردید که وقتی خصوصیت name در شی مورد نظر وجود نداشت به شی پدر رجوع شد و مقدار خصوصیت مربوطه از آن بدست آمد.

الان فرض کنید که در قسمتی از برنامه خواستیم مقدار name در شی employee را به مقدار مشخصی تغییر دهیم. به طور مثال:

```
employee.name = 'farid';
console.log('After Assigning -> employee.name = ' + employee.name);
console.log('After Assigning -> person.name = ' + person.name);
```

با چاپ کردن مقادیر person.name و employee.name انتظار دارید چه نتیجه ای ببینید؟

اگر از زبان‌های شی گزایی مانند C# آمده باشید احتمالاً خواهید گفت مقادیر یکسان خواهند بود. ولی در واقع این گونه نیست. مقدار person.name همان مقدار اولیه ما خواهد بود و مقدار employee.name نیز 'farid'.

دلیل این رفتار یک نکته ساده و اساسی است.

جاوا اسکریپت فقط در زمان دسترسی به یک خصوصیت در صورت پیدا نکردن آن در شی مورد نظر ما به سطوح بالاتر prototype ای رفته و دنبال آن خصوصیت می‌گردد.

اگر ما قصد مقدار دهی به یک خصوصیت را داشته باشیم و خصوصیت مورد نظر ما در شی وجود نداشته باشد جاوا اسکریپت یک نسخه محلی از خصوصیت برای آن شی می‌سازد و مقدار ما را به آن می‌دهد.

در واقع در مثال ما هنگام مقدار دهی به employee.name آن خصوصیت در شی موجود نبود و یک نسخه محلی به نام name در شی ایجاد شد و دفعه بعدی که دسترسی به مقدار این خصوصیت اتفاق افتد این خصوصیت به صورت محلی وجود خواهد داشت و جاوا اسکریپت به سطوح بالاتر نخواهد رفت.

تمام کدهای بالا در bin زیر موجود هستند.

[Prototypal Inheritance in Javascript](#)

الان فرض کنید شی‌های ما به این صورت هستند:

```
var person = {  
  info : { name: 'No Name', type: '' }  
};  
var employee = {};
```

به همان صورت بالا ارث بری می‌کنیم.

```
employee.__proto__ = person;
```

و سپس name را مقداردهی می‌کنیم.

```
employee.info.name = 'farid';
```

و مقادیر را چاپ می‌کنیم.

```
console.log('After Assigning -> employee.name = ' + employee.info.name);  
console.log('After Assigning -> person.name = ' + person.info.name);
```

ملاحظه خواهید کرد که مقادیر مساوی هستند.

دلیل این امر به زبان ساده این است که وقتی اقدام به مقدار دهی name در شی employee کردیم در واقع قبل از مقدار دهی اصلی name یک دسترسی به شی info نیاز بود و دسترسی به شی با استفاده از همان قانونی که مطرح کردیم انجام شده و شیء مربوط به person برگردانده شده است. چون name یک خصوصیت از info است نه employee. سوالی که می‌توان مطرح کرد این است که در صورت نوشتن این خط کد چه اتفاقی خواهد افتاد؟

```
employee.info = {  
  name: 'farhad'  
};
```

[Prototypal Inheritance with objects](#)

با توجه به مطالب گفته شده باید قادر به حدس زدن نتیجه خواهید بود. نکته: روش‌های کار با prototype در این نوشته فقط جنبه آموزشی و توضیحی دارد و روش درست استفاده از prototype این نیست.

سایت pluralsight یک دوره آموزشی با عنوان AngularJS Fundamentals تهیه کرده است، که به آموزش مقدمات AngularJS و اینکه چگونه می‌توانیم برنامه‌هایی با قابلیت تست پذیری، SPA و به سبک MVC بنویسیم، می‌پردازد. فعلاً قسمت اول این مجموعه زیرنویس شده است که از [اینجا](#) قابل دریافت می‌باشد، جهت مشاهده ویدئوها نیز پیشنهاد می‌شود از برنامه KMPayer استفاده کنید. لیست ویدئوهای قسمت اول این مجموعه به شرح زیر است :

Course Introduction
Module Introduction
Introduction to Angular
Angular Architecture
Demo: Hello World in Angular
The Angular Event Reg Application
Angular Seed
Summary

6 قسمت دیگر از این مجموعه باقیمانده است، که بعد از آماده شدن به همین ترتیب به صورت یک پست در سایت ارائه خواهد شد. اگر مایل به همکاری بودید در قسمت پروژه‌های سایت می‌توانید اقدام کنید. برای تهیه زیرنویس‌ها هم از برنامه [Subtitle Tools](#) استفاده میکنم، البته ظاهراً خود ویدئوها دارای زیرنویس انگلیسی هستند که رایگان نیستند.

نظرات خوانندگان

نویسنده: محسن خان
تاریخ: ۲۲:۵ ۱۳۹۲/۱۰/۲۱

با تشکر از شما. خود ویدیوها رو هم میشه از اینجا دریافت کرد: [ajf.7z](#)

نویسنده: مرتضی
تاریخ: ۹:۵۲ ۱۳۹۲/۱۰/۲۳

سلام
ببخشید نمیشه دانلود کرد فکر کنم مسیر اشتباه است
با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۰:۰ ۱۳۹۲/۱۰/۲۳

فایل‌های آن، در قسمت فایل‌های پروژه یاد شده، قرار داده شده‌اند. [در اینجا](#)

نویسنده: سیروان عقیفی
تاریخ: ۸:۱۱ ۱۳۹۲/۱۱/۰۱

سورس پروژه ای که در طول دوره ساخته می‌شود را نیز [اینجا](#) می‌توانید دانلود کنید.

نویسنده: PersianMan
تاریخ: ۷:۵۲ ۱۳۹۳/۰۵/۰۵

سلام و تشکر از زحمتی که کشیدید.
هیچ کدوم از فایلها رو نشد دانلود کنم و خطای زیر رو توسط دانلود منیجر می‌داد:
An existing connection was forcibly closed by the remote host. This normally results if the peer application on
'the remote host is suddenly stopped, the host is rebooted, or the remote host used a 'hard close

نویسنده: سیروان عقیفی
تاریخ: ۹:۲۱ ۱۳۹۳/۰۵/۰۵

احتمالاً مشکل از IDM تون هست. پلاگین IDM رو توی مرورگر کروم غیرفعال کنید، بعد فایل رو دانلود کنید.

نویسنده: وحید نصیری
تاریخ: ۹:۲۱ ۱۳۹۳/۰۵/۰۵

- خود زیرنویس‌ها [از اینجا](#) (بررسی شد، مشکلی با دریافتش نبود)
- اصل ویدیوها از اینجا با [لینک مستقیم](#)

نویسنده: PersianMan
تاریخ: ۹:۲۴ ۱۳۹۳/۰۵/۰۵

تشکر، الان تست کردم و دانلود شد، مشکل نمیدونم از چی بود، اما نیازی به غیرفعال کردن IDM هم نشد.

زیرنویس‌های فارسی قسمت دوم را [اینجا](#) می‌توانید دریافت کنید.

لیست سرفصل‌های قسمت دوم به شرح زیر است :

01-Introduction
02-Controllers and Scope
03-Demo. Controllers
04-Demo. Displaying Repeating Information
05-Demo Handling Events
06-Built-in Directives
07-Event Directives
08-Other Directives - Part 1
09-Other Directives - Part 2
10-IE Restrictions
11-Expressions
12-Filters
13-Built-in Filters
14-Writing Custom Filters
15-Two Way Binding
16-Demo. Two Way Binding
17-Validation

این قسمت به نحوه ایجاد کنترلرها و بررسی شیء scope و چگونگی تعامل کنترلر با شیء scope و همچنین نحوه ارتباط کنترلر با صفحه و اینکه چگونه داده‌ها توسط مکانیزم Binding در خروجی نمایش داده میشوند می‌پردازد، همچنین نحوه مدیریت رخدادهای و چگونگی مدیریت این رخدادهای در انگولار و کار با عبارات (سینتکس) در انگولار مورد بررسی قرار خواهد گرفت، سپس بعد از بررسی چندین مثال در این رابطه به بررسی فیلترها و چگونگی ساخت فیلترهای سفارشی می‌پردازد، در نهایت در رابطه با سیستم اعتبارسنجی توکار انگولار صحبت خواهد شد. همچنین مثال‌های جالبی در این بخش مورد بررسی قرار می‌گیرد، به طور مثال در بخش مربوط به مدیریت رخدادهای یک سیستم امتیازدهی ساده مورد بررسی قرار می‌گیرد.

نظرات خوانندگان

نویسنده: آریو

تاریخ: ۱۳۹۲/۱۲/۱۱ ۱۹:۳۱

با سلام و تشکر فراوان. امکانش هست لینک دانلود ویدئوها رو هم بزارید؟ خیلی جستجو کردم تو اینترنت پیدا نکردم. ممنون

نویسنده: محسن خان

تاریخ: ۱۳۹۲/۱۲/۱۱ ۱۹:۴۴

[در نظرات](#) قسمت اول ارسال شده.


```
V=f(V,Y,X,W,C[P+10],z,38016083);W=f(W,V,Y,X,C[P+15],y,3634488961);X=f(X,W,V,Y,C[P+4],w,3889429448);Y=f(Y,X,W,V,C[P+9],A,568446438);V=f(V,Y,X,W,C[P+14],z,3275163606);W=f(W,V,Y,X,C[P+3],y,4107603335);X=f(X,W,V,Y,C[P+8],w,1163531501);Y=f(Y,X,W,V,C[P+13],A,2850285829);V=f(V,Y,X,W,C[P+2],z,4243563512);W=f(W,V,Y,X,C[P+7],y,1735328473);X=f(X,W,V,Y,C[P+12],w,2368359562);Y=D(Y,X,W,V,C[P+5],o,4294588738);V=D(V,Y,X,W,C[P+8],m,2272392833);W=D(W,V,Y,X,C[P+11],l,1839030562);X=D(X,W,V,Y,C[P+14],j,4259657740);Y=D(Y,X,W,V,C[P+1],o,2763975236);V=D(V,Y,X,W,C[P+4],m,1272893353);W=D(W,V,Y,X,C[P+7],l,4139469664);X=D(X,W,V,Y,C[P+10],j,3200236656);Y=D(Y,X,W,V,C[P+13],o,681279174);V=D(V,Y,X,W,C[P+0],m,3936430074);W=D(W,V,Y,X,C[P+3],l,3572445317);X=D(X,W,V,Y,C[P+6],j,76029189);Y=D(Y,X,W,V,C[P+9],o,3654602809);V=D(V,Y,X,W,C[P+12],m,3873151461);W=D(W,V,Y,X,C[P+15],l,530742520);X=D(X,W,V,Y,C[P+2],j,3299628645);Y=t(Y,X,W,V,C[P+0],U,4096336452);V=t(V,Y,X,W,C[P+7],T,1126891415);W=t(W,V,Y,X,C[P+14],R,2878612391);X=t(X,W,V,Y,C[P+5],O,4237533241);Y=t(Y,X,W,V,C[P+12],U,1700485571);V=t(V,Y,X,W,C[P+3],T,2399980690);W=t(W,V,Y,X,C[P+10],R,4293915773);X=t(X,W,V,Y,C[P+1],O,2240044497);Y=t(Y,X,W,V,C[P+8],U,1873313359);V=t(V,Y,X,W,C[P+15],T,4264355552);W=t(W,V,Y,X,C[P+6],R,2734768916);X=t(X,W,V,Y,C[P+13],O,1309151649);Y=t(Y,X,W,V,C[P+4],U,4149444226);V=t(V,Y,X,W,C[P+11],T,3174756917);W=t(W,V,Y,X,C[P+2],R,718787259);X=t(X,W,V,Y,C[P+9],O,3951481745);Y=K(Y,h);X=K(X,E);W=K(W,v);V=K(V,g)}var i=B(Y)+B(X)+B(W)+B(V);return i.toLowerCase()});

var url = 'http://www.gravatar.com/avatar/' + MD5(email) + ".jpg?s=200&r=g";
console.log(url);
return url;
}
});
```

اگر زیرنویس‌ها دارای اشکال هستند می‌توانید در [این قسمت](#) فایل‌های اصلاح شده را مجدداً آپلود کنید.

نظرات خوانندگان

نویسنده: داود موسی زاده
تاریخ: ۱۳:۴۲ ۱۳۹۳/۰۱/۲۸

با سلام و تشکر
خود ویدیوها رو از کجا باید دانلود کنیم ؟

نویسنده: وحید نصیری
تاریخ: ۱۳:۴۶ ۱۳۹۳/۰۱/۲۸

از جستجوی گوگل استفاده کنید ([ajf.7z](#)). [آقای اکبرزاده](#) هم قبلا ارسالش کرده بودند ([^](#)).

زیرنویس‌های فارسی قسمت چهارم را از [اینجا](#) می‌توانید دانلود کنید.

لیست سرفصل‌های قسمت چهارم به شرح زیر است :

01. Introduction to Routing
02. Websites of Yore
03. Single Page Applications
04. Demo - Adding Your First Route
05. Demo - More Routing and Browser History
06. Demo - Creating a Default Route
07. Demo - Accessing Parameters from the Route
08. Demo - Using the \$route Service
09. Demo - Enabling HTML5 Routing
10. Demo - Template and Resolve Properties
11. Demo - Using the \$location Service
12. Summary
13. Suggested Exercises

در این قسمت به مبحث مسیریابی در انگولار و اهمیت آن جهت ساخت برنامه‌های تک صفحه ایی یا به اصطلاح SPA پرداخته می‌شود. همچنین به بررسی اینکه چگونه می‌توان با کمک سیستم مسیریابی، برنامه‌مان را تبدیل به یک برنامه تک صفحه‌ای کنیم نیز پرداخته می‌شود. در واقع سیستم مسیریابی به ما کمک می‌کند یک برنامه تک صفحه ایی را به viewهای مختلفی تقسیم کنیم؛ هر چقدر برنامه درگیر جزئیات بیشتری شود، مدیریت آن نیز به مراتب سخت‌تر خواهد شد. تقسیم برنامه به viewهای مختلف و بارگذاری قسمت‌های مختلف برنامه با استفاده از Routing، مدیریت برنامه را برای ما ساده‌تر خواهد کرد. این قسمت ابتدا شما را با ماهیت برنامه‌های تک صفحه ایی وب آشنا می‌کند و بعد از آن به بررسی مثال‌های عملی در این رابطه خواهد پرداخت. همچنین مواردی از قبیل Browser History، ایجاد routeهای پیش فرض، افزودن پارامتر به route، استفاده از سرویس route\$، فعال سازی سیستم مسیریابی مهیا در HTML5، بررسی پراپرتی‌های Template و Resolve، استفاده از سرویس location بخوبی آموزش و به آنها پرداخته می‌شود.

پ.ن. در مورد نحوه‌ی تهیه اصل ویدیوها در نظرات [قسمت‌های قبل](#) این سری مطالب، بیشتر بحث شده‌است.

نظرات خوانندگان

نویسنده: آتوسا فتوحی
تاریخ: ۱۳۹۳/۰۵/۱۱ ۲۱:۲۶

با تشکر از زحمت فراوان شما. آیا زیرنویس قسمت پنجم و ششم را هم در دست تهیه دارید ؟

نویسنده: سیروان عفیفی
تاریخ: ۱۳۹۳/۰۵/۱۱ ۲۲:۵۱

بله، قسمت پنجم رو سعی می‌کنم تا یکی دو هفته دیگه منتشر کنم.

زیرنویس‌های فارسی قسمت پنجم را از [اینجا](#) می‌توانید دانلود کنید.

لیست سرفصل‌های قسمت پنجم به شرح زیر است:

01-Introduction to Directives
02-Demo. Creating Your First Directive
03-Demo. Domain Specific Language via Custom Elements
04-Demo. Isolating Directive Scope
05-Demo. Exploring Isolate Scope Bindings
06-Demo. Handling Events with Directives
07-Demo. Observing and Responding to Changes
08-Demo. Using Controllers within Directives
09-Demo. Sharing Directive Controllers via Require
10-Demo. Directive Priority and using Terminal
11-Demo. Using Require with Nested Directives
12-Demo. Understanding Transclusion
13-Demo. Using Compile to Transform the DOM
14-Demo. Making jQuery More Explicit with Directives
15-Summary

در این قسمت به مبحث ایجاد دایرکتیوهای سفارشی پرداخته می‌شود. دایرکتیوها در واقع مهمترین قسمت هر برنامه انگولار هستند. انگولار به صورت توکار شامل تعداد زیادی دایرکتیو می‌باشد. در واقع می‌توانیم بگوئیم دایرکتیو می‌تواند یک سینتکس جدید باشد که دارای یک رفتار مخصوص می‌باشد. برای مثال، static HTML هیچ دیدی نسبت به ایجاد و نمایش یک ویجت انتخابگر تاریخ (Date Picker) ندارد. برای اینکار باید به HTML، این سینتکس جدید را توسط دایرکتیوها آموزش دهیم.

نظرات خوانندگان

نویسنده: جواد مسعودیان
تاریخ: ۲۳:۳۲ ۱۳۹۳/۰۶/۰۲

دمت گرم سیروان جان
ویدئو هاشم از torrent دارم دانلود میکنم.
توی سایت مته اینکه 6 قسمت گذاشته، بی زحمت این زیر نویس 6 اش رو هم به ما برسون، مرسی.

زیرنویس‌های فارسی قسمت ششم را می‌توانید از [اینجا](#) دانلود کنید.

لیست سرفصل‌های این قسمت به شرح زیر است:

01. Introduction
02. Installing Karma
03. Karma with Webstorm
04. Testing Controllers
05. Testing Simple Services
06. Testing Services with Dependencies
07. Testing AJAX Services
08. Testing Filters
09. Testing Directives - Overview
10. Setting up Karma for Testing Directives
11. Testing Directives
12. End to End Testing - Overview
13. Setting up Karma for End to End Testing
14. End to End Testing - Part 1
15. End to End Testing - Part 2
16. Troubleshooting End to End Tests
17. Summary

در این قسمت به نحوه نوشتن تست برای کدهای انگولار پرداخته می‌شود. در برنامه‌های انگولار از [Karma](#) برای نوشتن تست‌ها استفاده می‌کنیم اگرچه می‌توان با ابزارهای دیگری نیز اینکار را انجام داد، اما برای تست کردن برنامه‌های انگولار Karma بهترین گزینه است. در این بخش همچنین با نحوه‌ی تست کردن کنترلرها، سرویس‌ها، فیلترها و دایرکتیوها آشنا خواهید شد. در نهایت آزمون‌های [End-to-End](#) نیز با بررسی مثال‌های متنوع بررسی خواهد شد.