AngularJS

قوی‌ترین فریم‌ورک جاوااسکریپتی MVC

ویرایش نخست

مهندس سید منصور عمرانی

انتشارات پندار پارس

سرشناسه : گرین، برد
Green, Brad
عنوان و نام پدیدآور : AngularJS؛ مترجم سیدمنصور عمرانی.
مشخصات نشر : تهران : پندار پارس، ۱۳۹۲.
مشخصات ظاهری : ۲۰۰ص.: مصور، جدول.
شابک : ۱۲۰۰۰۰ ریال: ۲-٤٦-٦٥٢٩-٦٠٠-۹۷۸
وضعیت فهرست نویسی : فیپا
یادداشت : عنوان اصلی: AngularJS.
ترجمه عنوان : آنگولار جی اس.
موضوع : نرم‌افزار کاربردی -- طراحی و توسعه
موضوع : نرم‌افزار متن‌باز
موضوع : جاوا اسکریپت (زبان برنامه‌نویسی کامپیوتر)
شناسه افزوده : سشادری، شیام
شناسه افزوده : Seshadri, Shyam
شناسه افزوده : عمرانی، سیدمنصور، ۱۳۵۶ -، مترجم
رده بندی کنگره : ۰۴۴ن/ QA۷۶/۷۶گ۴ ۱۳۹۲
رده بندی دیویی : ۰۰۵/۳
شماره کتابشناسی ملی : ۳۳۵۸۸۹۹

انتشارات پندارپارس

دفتر فروش: انقلاب، ابتدای کارگرجنوبی، کوی رشتچی، شماره ١٤، واحد ١٦ **www.pendarepars.com**

تلفن: ٦٦٥٧٢٣٣٥ - تلفکس: ٦٦٩٢٦٥٧٨همراه: ۰۹۱۲۲٤٥٢٣٤٨ **info@pendarepars.com**

نام کتاب : **AngularJS، قوی‌ترین فریم‌ورک جاوااسکریپتی MVC**

ناشر : **انتشارات پندار پارس**

ترجمه و تالیف : **سید منصور عمرانی**

چاپ نخست : **بهمن ۹۲**

شمارگان : **۵۰۰ نسخه**

طرح جلد : **فرزانه روزبهانی**

لیتوگرافی، چاپ، صحافی : **ژولان**

قیمت : **۱۲۰۰۰ تومان** **شابک : ۲-۴۶-۶۵۲۹-۶۰۰-۹۷۸**

فهرست

فصل ۱. معرفی AngularJS ۱

مفاهیم اولیه ۱

الگوهای سمت کلاینت ۲

الگوی MVC یا Model-View-Controller ۳

مقیدسازی داده ۴

تزریق وابستگی ۶

رهنمودها ۷

یک مثال: سبد خرید ۷

پایان سخن ۱۱

فصل ۲. آناتومی یک برنامه‌ی AngularJS ۱۳

فعال کردن Angular ۱۳

بارگذاری کتابخانه‌ی AngularJS ۱۳

تعریف محدودیت‌های کاری Angular با استفاده از ng-app ۱۳

Model View Controller ۱۴

الگوها و مقیدسازی داده ۱۶

نمایش متن ۱۷

ورودی فرم‌ها ۱۸

چند کلمه‌ای در مورد Unobtrusive Javascript ۲۲

لیست‌ها، جدول‌ها و سایر المان‌های تکرارپذیر ۲۵

پنهان‌سازی و نمایش اجزای صفحه ۲۷

کلاس‌ها و شیوه‌های CSS ۲۸

ملاحظات ویژگی‌های src و href ۳۱

عبارت‌ها ۳۱

استفاده از کنترلر برای جدا کردن مسئولیت واسط کاربر ۳۲

قابل دسترس کردن مدل‌ها با استفاده از حوزه‌های دسترسی یا scope ۳۳

زیر نظر گرفتن تغییرات مدل‌ها با استفاده از $watch ۳۴

ملاحظات کارایی در $watch() ۳۶

ردگیری تغییرات چند چیز مختلف به صورت همزمان ۳۸

سازمان‌دهی وابستگی‌ها در ماجول‌ها ۳۹

به چه تعداد ماجول نیاز داریم؟ ۴۲

قالب‌بندی داده‌ها با فیلترها ۴۳

تغییر دادن نماها با استفاده از مسیرها و $location ۴۴

ارتباط با سرورهای راه دور ۴۸

دستکاری DOM ۴۹

اعتبارسنجی مقادیر وارد شده توسط کاربر ۵۱

خلاصه ۵۳

فصل ۳. مقدمات برنامه‌نویسی در AngularJS ۵۵
سازمان‌دهی پروژه ۵۵
ابزارها ۵۸
محیط برنامه‌نویسی (IDE) ۵۸
اجرای برنامه ۵۹
با استفاده از Yeoman ۵۹
بدون استفاده از Yeoman ۵۹
اجرای آزمون‌های واحد برای برنامه‌های AngularJS ۶۰
Karma ۶۱
آزمون‌های واحد ۶۳
آزمون‌های مجتمع و سر به سر ۶۴
کامپایل ۶۶
سایر ابزارهای مفید ۶۷
اشکال‌زدایی ۶۸
Batarang ۶۸
برگه‌ی Model ۶۸
برگه‌ی Performance ۶۹
برگه‌ی Dependencies ۷۰
خصوصیت المان‌ها و دسترسی به کنسول ۷۱
Yeoman: بهبود گردش کار توسعه‌ی برنامه ۷۲
نصب Yeoman ۷۳
شروع یک پروژه‌ی AngularJS ۷۳
اجرای سرور ۷۳
افزودن مسیرها، نماها و کنترلرها ۷۳
داستان آزمون‌ها ۷۴
ساخت پروژه ۷۴
مجتمع کردن AngularJS با RequireJS ۷۴
فصل ۴. بررسی یک برنامه‌ی Angular ۸۵
برنامه ۸۵
ارتباط بین مدل‌ها، کنترلرها و الگوها ۸۶
مدل ۸۷
کنترلرها، رهنمودها و سرویس‌ها، آه خدای من! ۸۸
سرویس‌ها ۸۸
رهنمودها ۹۱
کنترلرها ۹۳
الگوها ۹۷
آزمون‌ها ۱۰۳
آزمون‌های واحد ۱۰۳
آزمون‌های سناریو ۱۰۷
فصل ۵. ارتباط با سرور راه دور ۱۰۹
برقراری ارتباط با سرور از طریق سرویس $http ۱۰۹

تنظیم دقیق‌تر درخواست ۱۱۱
مشخص کردن هِدِرهای HTTP ۱۱۲
کَش کردن پاسخ درخواست‌ها ۱۱۳
تغییر دادن درخواست و پاسخ ۱۱۳
آزمون واحد ۱۱۴
کار کردن با منابع RESTful ۱۱۶
تعریف منبع ۱۱۸
متدهای شخصی ۱۱۹
callback نداریم (مگر آن که واقعا چنین چیزی را بخواهید!) ۱۱۹
عملیات سروری ساده ۱۲۰
آزمون واحد کردن ngResource ۱۲۰
$q و وعده‌ها ۱۲۱
مداخله در پاسخ ۱۲۳
ملاحظات امنیتی ۱۲۳
آسیب‌پذیری JSON ۱۲۴
حمله‌ی XSRF ۱۲۵
فصل ۶. رهنمودها ۱۲۷
رهنمودها و اعتبارسنجی HTML ۱۲۷
رهنمودها API ۱۲۸
نام‌گذاری رهنمود ۱۲۹
شیء تعریف رهنمود ۱۳۰
نحوه‌ی استفاده ۱۳۰
اولویت‌بندی ۱۳۰
الگوسازی ۱۳۱
انتقال محتوا ۱۳۴
توابع کامپایل و لینک ۱۳۴
Scope ها ۱۳۶
دستکاری المان‌های DOM ۱۴۰
کنترلرها ۱۴۲
ادامه ۱۴۵
فصل ۷. سایر مسائل ۱۴۷
$location ۱۴۷
خداحافظی با وضعیت سراسری ۱۴۷
API ۱۴۷
مجتمع‌سازی با AngularJS ۱۴۷
مجتمع‌سازی با HTML5 ۱۴۷
مُد HTML5 و Hashbang ۱۵۲
پیکربندی سمت سرور ۱۵۳
بازنویسی لینک‌ها ۱۵۳
آدرس‌های نسبی ۱۵۴

متدهای ماجول‌های AngularJS ۱۵۴
متد main کجا است؟ ۱۵۴
بارگذاری و وابستگی‌ها ۱۵۵
متدهای مفید ۱۵۵
تابع ()factory ۱۵۶
تابع ()service ۱۵۷
تابع ()provider ۱۵۷
ارتباط scope ها با یکدیگر با استفاده از $on، $emit و $broadcast ۱۵۸
کوکی‌ها ۱۶۰
محلی‌سازی و بین‌المللی‌سازی ۱۶۰
در AngularJS چه می‌توانیم بکنیم؟ ۱۶۱
چگونه قابلیت i18n و L10n را فراهم کنم؟ ۱۶۱
چند ترفند رایج ۱۶۲
امنیت HTML و ماجول Sanitize ۱۶۲
فیلتر linky ۱۶۴
فصل ۸. نکته‌ها و ترفندها ۱۶۵
ساخت یک مولفه‌ی AngularJS برای انتخاب تاریخ با استفاده از مولفه‌ی jQuery DatePicker ۱۶۵
ng-model ۱۶۶
ngModel.$setViewValue(dateText) ۱۶۷
ngModel.$render ۱۶۷
select مقید شده ۱۶۷
فراخوانی select ۱۶۷
مابقی مثال ۱۶۸
برنامه‌ی لیست تیم‌های ورزشی: نمونه‌ای از فیلترسازی و مکالمه‌ی بین کنترلرها ۱۶۹
جعبه‌ی جستجو ۱۷۲
کومبوها ۱۷۲
چک‌باکس ۱۷۲
تکرارگر ۱۷۳
آپلود فایل توسط AngularJS ۱۷۳
fileupload ۱۷۵
data-url ۱۷۵
multiple ۱۷۵
done ۱۷۵
استفاده از Socket.IO ۱۷۶
یک سرویس صفحه‌بندی ساده ۱۷۸
کار با سرورها و Login ۱۸۲
خاتمه ۱۸۵

مقدمه‌ی مترجم

دنیای وب در طول سال‌های اخیر دستخوش تحولات بیشماری بوده است. یکی از این دگرگونی‌ها، پیدایش فریم‌ورک‌های جاوااسکریپتی مبتنی بر الگوی MVC مانند Backbone، Ember، Knockout، Agility، Maria، Batman، Sammy، Spine و ... بوده که تغییری اساسی در ساخت برنامه‌های تحت وب و کارکرد و کارایی برنامه‌ها ایجاد کرده است.

ایده‌ی کلی فریم‌ورک‌های MVC این است که بخشی از منطق برنامه را از سمت سرور به سمت کلاینت منتقل کنیم، مدیریت برنامه را سمت کلاینت انجام بدهیم و به جای روش سنتی تولید همه‌ی خروجی صفحه در سمت سرور و تحمیل بار اضافی به سرور، بخش عمده‌ی واسط کاربر را سمت کلاینت بسازیم. در این حالت برنامه‌ی سمت سرور بیشتر به تعدادی سرویس (معمولا REST) تبدیل خواهد شد که داده‌ها را بر اساس منطق کسب و کار برنامه مدیریت می‌کند.

در این کتاب، فریم‌ورک AngularJS یکی از مشهورترین فریم‌ورک‌های جاوااسکریپتی MVC آموزش داده می‌شود که توسط تعدادی از برنامه‌نویسان گوگل پس از سال‌ها تجربه در نوشتن برنامه‌های مختلف ابداع شده است. حقیقتا AngularJS فریم‌ورک بسیار زیبا و ارزشمندی است و افزون بر ساده‌تر کردن برنامه، بار زیادی از پردازش سمت سرور را حذف می‌کند. در نتیجه برنامه‌ی نوشته شده بسیار سریع‌تر، سبک‌تر و پاسخ‌پذیرتر شده و افزون بر این که می‌تواند به ترافیک بیشتری سرویس بدهد، می‌تواند تعداد بسیار بیشتری از کاربران همزمان را نیز اداره کند.

چند نکته‌ی احتیاطی

با وجود زیبایی و سبکی AngularJS، چند نکته‌ی مهم یا تعدادی ریسک در مورد برنامه‌های Angular وجود دارد که پیش از این که خود را درگیر چنین برنامه‌هایی کنید حتما باید از آنها آگاه باشید.

- **AngularJS تنها اسکلت و چهارچوب برنامه را فراهم می‌کند**

Angular سکویی است که تنها چهارچوب و اسکلت برنامه را *فراهم می‌کند*، اما خودش از پیش در این زمینه چیزی ندارد. لذا همه‌ی چیزها را خودتان باید درست کنید (ماجول‌ها، دایرکتیوها، فیلترها، الگوها).

- **به سایر فریم‌ورک‌ها هم نیاز دارید**

برای بسیاری از دیگر جنبه‌های برنامه به استفاده از فریم‌ورک‌های جاوااسکریپتی یا CSS نیاز خواهید داشت. لذا نباید تصور کنید Angular بسته‌ای است که همه چیز در آن وجود داشته و می‌تواند همه‌ی نیازهای برنامه‌ی شما را فراهم کند. آن بیرون در اینترنت چیزهای بسیار زیادی اطراف AngularJS پیدا خواهید کرد که بر پایه‌ی AngularJS ساخته شده‌اند و می‌توانید آنها را در برنامه‌های خود به کار ببرید. چیزی که در این کتاب توضیح داده می‌شود تنها خود AngularJS است.

- **AngularJS فاقد کنترل‌های واسط کاربر است**

AngularJS بستر ساخت کنترل‌های واسط کاربر را فراهم می‌کند، اما خودش هیچ کنترل واسط کاربر از پیش آماده‌ای ندارد (حداقل فعلا). در حقیقت ماجرای AngularJS و کنترل‌های واسط کاربر چیزی شبیه

رابطه‌ی jQuery و jQuery UI است با این تفاوت که در حال حاضر چیزی به نام Angular UI که به طور رسمی توسط خود تیم AngularJS ساخته شده باشد وجود ندارد. لذا یا باید این بسته‌ی کنترل‌های واسط کاربر را خودتان درست کنید که قطعا کار چندان راحتی نیست و و افزون بر داشتن تجربه و مهارت بسیار زیاد در این کار، پروسه‌ی زمان‌بری است. راه دیگر هم این است که اینترنت را بگردید و از بسته‌هایی که دیگران ساخته‌اند استفاده کنید (مانند AngularUI در آدرس http://angular-ui.github.io). در این حالت هزینه‌ی یادگیری آنها نیز افزون بر یادگیری خود AngularJS به شما تحمیل خواهد شد. از این رو هنگام شروع یک برنامه‌ی وب Angular باید زمان و هزینه‌ی این کارها را هم در نظر بگیرید.

- **برنامه‌های Angular مشکل SEO دارند**

متاسفانه این یک حقیقت است. برنامه‌های Angular با تمام زیبایی و سبکی‌شان از نظر SEO مشکل دارند. علت مساله نیز واضح است. روبات‌های موتورهای جستجو تنها سورس صفحه‌ی HTML را پردازش کرده و اساسا کُدهای جاوااسکریپت را اجرا نمی‌کنند. از آنجایی که واسط کاربر برنامه‌های Angular به طور کامل بر پایه‌ی جاوااسکریپت ساخته می‌شود، محتوای برنامه رتبه‌ی پایینی در موتورهای جستجو خواهد داشت و حتی گاهی برخی از محتوا اصلا ایندکس نخواهد شد. البته راه‌حل‌هایی برای بهبود این وضعیت وجود دارد. برای نمونه می‌توانید به آدرس‌های زیر مراجعه کنید:

http://www.yearofmoo.com/2012/11/angularjs-and-seo.html

http://www.angularjsseo.com

با توجه به این مساله شاید بتوان گفت Angular بیشتر برای برنامه‌های تحت وب محلی و خصوصی مناسب است که نیازی ندارند به طور عمومی قابل دسترس بوده و توسط موتورهای جستجو ایندکس شوند.

- **مدل AngularJS جدید است و اجزای آن ممکن است پاسخگوی همه‌ی نیازهای معماری برنامه‌های بزرگ نباشد**

قابلیت‌های AngularJS محدود به چند بخش عمده و محدود به نام ماجول، دایرکتیو، فیلتر، سرویس و کنترلر است. با وجودی که با این اجزاء می‌توان ساختار و چهارچوب برنامه‌ها را بر پا کرد، اما معماری و ساختار برنامه تنها محدود به این اجزاء نیست و برای آن به مدل‌های دیگری هم نیاز دارید. اما در این زمینه راه‌کار و رهیافت از پیش مشخصی وجود ندارد و خودتان هستید که باید آنها را ابداع کنید. این مساله باعث می‌شود ایجاد یک بستر عمومی مانند یک سیستم مدیریت محتوای دارای قابلیت استفاده‌ی مجدد بر پایه‌ی AngularJS که با استفاده از آن بتوان سایت‌های مختلفی را توسعه داد، از ریسک زیادی برخوردار باشد. از این رو باید گفت AngularJS در حال حاضر بیشتر برای برنامه‌های خصوصی و محدود که قرار نیست برنامه‌های دیگری بر پایه‌ی ساختار آنها شکل بگیرد مناسب است.

توضیحی در مورد این ترجمه

کتاب فعلی ترجمه‌ی کتاب AngularJS از انتشارات OReilly در سال ۲۰۱۳ است که توسط دو تن از اعضای تیم AngularJS تالیف شده است. بدون تعارف باید بگویم نسخه‌ی لاتین کتاب از نگارش خوبی برخوردار نبود و مترجم در ترجمه‌ی آن (به دلیل مبهم بودن بسیاری از جملات یا بیان نامناسب مطلب) دردسر زیادی کشید. به همین دلیل مترجم حداکثر تلاش خود را به کار برد تا با تغییر جمله‌بندی، مقصود مطلب به درستی بیان شود و امیدوار است در این زمینه موفق بوده باشد.

آیا این، کتاب کاملی است؟

کم حجم بودن کتاب ممکن است این تصور را ایجاد کند که مطالب این کتاب، سطحی و ساده است و در این کتاب آنقدرها به عمق AngularJS پرداخته نمی‌شود. این مطلب صحیح نیست. کتاب مزبور از لحاظ محتوا و پوشش دادن فریم‌ورک AngularJS کامل است و می‌توانید مطمئن باشید پس از مطالعه‌ی کتاب، همه‌ی پیچ و خم‌های AngularJS را فرا خواهید گرفت. اما تردیدی وجود ندارد که هیچ چیزی مانند تجربه‌ی عملی نیست. در اینترنت نیز سایت‌ها و منابع زیادی برای آموزش و یادگیری پیدا خواهید کرد. برای نمونه می‌توانید به سایت‌های زیر مراجعه کنید:

http://egghead.io

http://www.reddit.com/r/angularjs

http://www.angularjsdaily.com

سایر کتاب‌ها

در بازار نشر حال و حاضر دنیا، کتاب‌های دیگری هم وجود دارند که شاید مفصل‌تر به AngularJS پرداخته باشند. اما زمانی که مترجم این کتاب را ترجمه نمود (بیش از ۵ ماه پیش)، هنوز نگارش آنها تکمیل نشده یا موجود نبودند. در این مدت نیز کتاب‌های دیگری در خصوص AngularJS نوشته شد و قطعا در آینده کتاب‌های دیگری هم وارد بازار خواهد شد. در اینجا برای آگاهی خواننده‌ی علاقمند، برخی از کتاب‌هایی را که تاکنون درباره‌ی AngularJS نگاشته شده بیان می‌کنم:

عنوان	ناشر	تعداد صفحه	تاریخ چاپ
Mastering Web Application Development with AngularJS	Packt Publishing	۳۷۲	آگوست ۲۰۱۳
AngularJS Directives	Packt Publishing	۱۱۰	سپتامبر ۲۰۱۳
ng-book The Complete Book on AngularJS	http://www.ng-book.com	۵۹۰	در حال نگارش
AngularJS Web Component	Addison-Wesley	۴۰۰	در حال نگارش (ژوئن

Development			۲۰۱۴)
AngularJS in Action	Manning Publishing	۳۲۵	در حال نگارش (مارس ۲۰۱۴)

سخن پایانی

هیچ کاری عاری از نقص نیست. مترجم امیدوار است ترجمه‌ی کتاب، کیفیت مطلوب را داشته باشد. در صورتی که ایرادی در ترجمه‌ی کتاب مشاهده کردید، حتما آن را با مترجم در میان بگذارید. آدرس تماس من mansoor.omrani@yahoo.com می‌باشد. همچنین از شنیدن سوالات، نظرات، انتقادات و پیشنهادات شما خوشحال خواهم شد. موفق باشید.

سید منصور عمرانی

پاییز ۱۳۹۲

فصل ۱. معرفی AngularJS

چندین سال است که برنامه‌های تحت وب در دنیای اینترنت در حال تاخت و تاز هستند، اما همانقدر که این برنامه‌ها شگفت‌انگیز و فوق‌العاده هستند و توانایی بالایی در ساخت این برنامه‌ها داریم، پیچیدگی ساخت این برنامه‌ها نیز بسیار زیاد است. بدون معطلی و مقدمه‌چینی یک راست به موضوع این کتاب یعنی Angular می‌پردازیم. انگیزه‌ی ما به عنوان تیم توسعه‌ی Angular این بود که دردسرهای ساخت برنامه‌های Ajax را کم کنیم. در شرکت Google ما در پروژه‌های بزرگی مانند Gmail، Maps، Calendar و بسیاری برنامه‌های دیگر شرکت داشتیم و درس‌های زیادی یاد گرفتیم. فکر کردیم شاید بتوانیم از تجربیات‌مان دیگران را هم منتفع کنیم.

هدف اصلی‌مان این بود که کاری کنیم نوشتن برنامه‌های وب به سادگی نوشتن برنامه‌های کوچکی شود که آنها را با چند خط کُد می‌نوشتیم، کمی عقب می‌رفتیم و بعد با شگفتی به آن نگاه کرده و آن را تحسین می‌کردیم. می‌خواستیم کُدنویسی برنامه‌های وب بیشتر به نوشتن خود برنامه متمرکز شده و کمتر به مسائل حاشیه‌ای مانند رعایت سازگاری بین مرورگرها یا سر و کله زدن با جزئیات عجیب و غریب مرورگرها و استانداردهای وب صرف شود.

همچنین می‌خواستیم محیطی فراهم کنیم که ساخت و درک برنامه را از همان ابتدا ساده کند و در عین حال انتخاب‌های درستی برای برنامه‌نویس فراهم کند که آزمایش، توسعه و نگهداری برنامه نیز همگام با رشد و بزرگ شدنش ساده باشد.

ما تمام این اهداف را در فریم‌وُرک Angular عملی کردیم و باید بگوییم از نتایج به دست آمده نیز بسیار هیجان‌زده شدیم! بیشترین اعتباری که دریافت کردیم از جانب جامعه‌های اُپن‌سورس فریم‌ورک‌های جاوااسکریپتی بود که اطراف Angular فعالیت می‌کنند، جوامعی که در پشتیبانی کردن از یکدیگر بسیار فوق‌العاده هستند و چیزهای بسیار زیادی هم به ما یاد دادند. امیدواریم شما هم به جامعه‌ی ما بپیوندید و به ما کمک کنید کاری کنیم Angular حتی از این هم بهتر شود.

مثال‌های بزرگتر و سنگین‌تر کتاب و نمونه کُدها را در یک انبار GitHub قرار داده‌ایم. لذا می‌توانید آنها را در GitHub ببینید، fork کنید و با آنها در GitHub ور بروید.

مفاهیم اولیه

پیش از شروع، چند ایده یا مفهوم اصلی وجود دارد که در همه‌ی برنامه‌های Angular استفاده می‌شود. ما خودمان هم این مفاهیم را ابداع نکرده‌ایم. بلکه آنها را از اصطلاح‌های محیطهای برنامه‌نویسی موفق دیگر قرض گرفته و طوری پیاده‌سازی کرده‌ایم که HTML، مرورگرها و بسیاری از استانداردهای آشنای دیگر در وب را به شکل زیبایی با هم ترکیب کرده و در کنار هم قرار می‌دهند.

الگوهای سمت کلاینت

در برنامه‌های وب چند صفحه‌ای، صفحه‌ها در سمت سرور ترکیب شلوغی از تگ‌های HTML و کدهای سمت سرور هستند. سپس صفحه‌ی سمت سرور اجرا می‌شود، خروجی نهایی HTML صفحه تولید شده و توسط وب‌سرور به سمت مرورگرها ارسال می‌شود. در این روش، موقع تولید خروجی صفحه، داده‌های سمت سرور داخل تگ‌های HTML گنجانده شده، از قبل با آنها ترکیب شده و به سمت مرورگر ارسال می‌شود. حتی بیشتر برنامه‌های تک صفحه‌ای در وب – که به آنها برنامه‌های Ajax گفته می‌شود – تا حدی چنین هستند. اما این همان جایی است که Angular با این ایده کاملا فرق دارد.

در Angular الگوی صفحه‌ها و داده‌های استفاده شده در آنها به طور مجزا به مرورگر داده می‌شود و این خود مرورگر است که با استفاده از Angular آنها را سر هم می‌کند. در واقع در یک برنامه‌ی Angular نقش سرور فقط سرویس دادن به الگوها و منابع استاتیک (عکس‌ها، اسکریپت‌ها، استایل‌ها، ...) و فراهم کردن داده‌های مورد نیاز الگوها است!

بگذارید ببینیم Angular چطور الگوها و داده‌ها را در سمت مرورگر با یکدیگر ترکیب می‌کند تا خروجی بصری نهایی را به دست بدهد. برای این کار از مثال سنتی و معروف Hello World استفاده می‌کنیم. محض کمی تفریح به جای این که تنها متن "Hello, World" را در صفحه نمایش بدهیم، برنامه را طوری می‌نویسیم که کلمه‌ی Hello، قابل تغییر باشد.

ابتدا الگوی برنامه را در فایلی به نام hello.html ایجاد می‌کنیم:

```
<html ng-app>
  <head>
    <script src="angular.js"></script>
    <script src="controllers.js"></script>
  </head>
  <body>
    <div ng-controller='HelloController'>
      <p>{{greeting.text}}, World</p>
    </div>
  </body>
</html>
```

سپس منطق این صفحه یا این برنامه‌ی ساده را در فایل جاوااسکریپتی controller.js قرار می‌دهیم. کُدهای Angular از همین جا شروع می‌شود:

```
function HelloController($scope) {
  $scope.greeting = { text: 'Hello' };
}
```

تمام شد! اگر صفحه‌ی hello.html را در یک مرورگر باز کنید چیزی شبیه شکل ۱–۱ خواهید دید.

Hello, World

شکل ۱–۱. مثال ساده‌ی Hello, World در Angular

در مقایسه با سایر روش‌های مختلف و رایجی که امروزه در نوشتن برنامه‌های تحت وب استفاده می‌شود چند نکته یا تفاوت جالب در این مثال وجود دارد:

- در تگ‌های HTML صفحه‌ی این مثال به هیچ وجه از ویژگی class یا id استفاده نشده است:
معمولا این ویژگی‌ها درون تگ‌ها، المان‌ها و اجزای مختلف صفحه قرار داده می‌شود تا بعدا بتوان در کُدهای جاوااسکریپت به اجزا صفحه ارجاع کرد و مثلا اداره‌گرهای رویدادی را به آنها وصل کرد.
- هیچ کُدی برای اداره کردن رویدادها نوشته نشده است:
در فایل controller.js وقتی داخل تابع HelloController می‌خواهیم کلمه‌ی Hello را در قسمت {greeting.text} که از قبل در الگوی hello.html تعریف کرده‌ایم قرار بدهیم، مجبور نیستیم رویدادهای کیبورد مانند keydown را اداره کنیم، callback تعریف کنیم یا به چیزی بر اساس id یا class ارجاع کنیم. به جای آن به طور ساده تنها greeting.text را مقداردهی می‌کنیم. همین. AngularJS خودش می‌داند چه باید بکند!
- تابع HelloController یک تابع ساده است:
تابع HelloController یک کلاس جاوااسکریپتی محض است و به هیچ تمهیداتی نیاز ندارد. برای نمونه نیازی نیست از هیچ چیزی که مثلا AngularJS فراهم می‌کند ارث ببرد.
- نیازی نیست خودمان HelloController را فراخوانی کنیم:
مجبور نیستیم خودمان HelloController را فراخوانی کرده یا حتی بدانیم چه موقع باید آن را فراخوانی کنیم. AngularJS خودش این تابع را صدا می‌زند.
- از پارامتر ویژه‌ای به نام $scope استفاده کرده‌ایم:
تنها نکته‌ی کلیدی تابع HelloController این است که پارامتری به نام $scope برای آن تعریف کرده‌ایم. این پارامتر یکی از اشیاء Angular است. اما نگران نباشید. به هیچ وجه نیازی نیست HelloController بداند این شیء از کجا می‌آید، چه موقع و چطور ایجاد می‌شود یا چه کسی این کار را انجام می‌دهد. AngularJS موقع فراخوانی HelloController خودش این شیء را ایجاد کرده و به HelloController پاس می‌دهد. HelloController تنها کافی است پارامتر $scope خود را استفاده کند. همین.

به نظرم از همین جا و با همین مثال ساده، مشخص است که ساخت برنامه‌های Angular نسبت به برنامه‌های مشابه گذشته، بسیار متفاوت است. کمی جلوتر به طور جزئی‌تری این تفاوت‌ها را بررسی می‌کنیم. ممکن است بپرسید چرا فریم‌ورک AngularJS را این گونه طراحی کرده‌ایم و اساسا Angular چگونه کار می‌کند؟ ابتدا بگذارید نگاهی به ایده‌های ارزشمندی بیندازیم که در نوشتن Angular از اینور و آنور کِش رفته‌ایم!

الگوی MVC یا Model-View-Controller

الگوی MVC در دهه‌ی ۱۹۷۰ به عنوان بخشی از زبان SmallTalk معرفی شد. پس از آن الگوی MVC در برنامه‌های رومیزی محبوبیت زیادی پیدا کرد و تقریبا در هر محیط برنامه‌نویسی رومیزی که یک جور واسط کاربر در آن درگیر بود استفاده شد. به طوری که خواه از C++، جاوا یا Objective-C استفاده می‌کردید، طعم و مزه‌ای از MVC در آن موجود بود. با این حال این الگو به جز این اواخر با برنامه‌نویسی وب بیگانه بود.

ایده‌ی اصلی MVC این است که در کُدهای برنامه، داده‌ها و مدیریت آنها (یعنی مدل آنها یا Model)، منطق برنامه (یعنی کنترلر یا Controller) و نمایش داده‌ها به کاربر (یعنی نماهای یا View) را به روشنی از یکدیگر تفکیک کنید.

کاری که کنترلرها انجام می‌دهند این است که داده‌ها را از مدل‌ها دریافت کرده و به نماها می‌دهند. نماها نیز آنها را به کاربر نمایش می‌دهند. وقتی کاربری از طریق کلیک یا تایپ، با برنامه کار می‌کند و مثلا چیزی را درخواست می‌کند، نماها درخواست را گرفته و به کنترلر می‌دهند. کنترلر نیز داده‌های مدل را تغییر داده و به نما اطلاع می‌دهد چیزی تغییر کرده است. لذا نما نیز خودش را به روز می‌کند.

در حالت کلی در برنامه‌های Angular نمای برنامه همان درختواره‌ی DOM است. کنترلرها نیز به شکل کلاس‌های جاوااسکریپتی نوشته می‌شوند. مدل‌ها نیز در قالب اشیاء جاوااسکپریتی تعریف می‌شوند.

به دلایل زیادی بر این باور هستیم که الگوی MVC بسیار زیبا است. اول این که به روشنی مشخص می‌کند هر چیزی را کجا باید بگذارید. لذا مجبور نیستید هر بار در خصوص مدل یا معماری برنامه‌ی خود فکر کنید و هر بار چنین مدلی را اختراع کنید. به علاوه با توجه به شناخته شده بودن الگوی MVC، سایر افرادی که در پروژه‌ی شما کار می‌کنند نیز به سرعت متوجه کُدهایی که می‌نویسید می‌شوند. اما مهمتر از همه این است که الگوی MVC واقعا ساخت برنامه را ساده‌تر کرده و توسعه، آزمون و نگهداری آن را راحت می‌کند.

مقیدسازی داده

پیش از برنامه‌های وب تک صفحه‌ای، سکوهایی مانند Rails، PHP یا JSP کمک می‌کردند واسط کاربر صفحات را در سمت سرور از ترکیب داده‌ها و تگ‌های HTML ایجاد کرده و سپس خروجی تولید شده را به سمت مرورگر ارسال کنید. کتابخانه‌هایی مانند jQuery نیز این مدل را در سمت کلاینت توسعه دادند و با استفاده از آن توانستیم سبک مشابهی را در سمت کلاینت دنبال کنیم، به طوری که به جای به روز رسانی کل صفحه، تنها بخشی از DOM را به روز می‌کردیم. روش کلی این بود که داده‌ها و تگ‌های HTML را به صورت رشته‌های جاوااسکریپتی با هم ترکیب کرده و سپس رشته‌ی به دست آمده را با استفاده از innerHtml در محل مورد نظرمان در DOM قرار می‌دادیم.

تمام این روش‌ها به خوبی کار می‌کنند. تنها ایرادشان این است که در این سبک پس از اضافه کردن اجزائی به DOM اگر می‌خواستیم آن اجزاء را دستکاری کرده و مثلا داده‌های تازه‌تری درون واسط کاربر قرار بدهیم یا آن را بر اساس ورودی و کلیک‌های کاربر تغییر بدهیم، باید کمی کُدنویسی می‌کردیم. این کُدها نیز چیزی جزئی و پیش پا افتاده نبودند. زیرا باید اطمینان حاصل می‌کردیم داده‌های برنامه در واسط کاربر و در اشیاء جاوااسکریپتی، با هم هماهنگ هستند. لذا گاهی اوضاع بسیار پیچیده می‌شد.

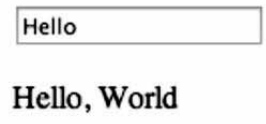
اگر می‌توانستیم کاری کنیم تمام این کارها بدون کُدنویسی به طور خودکار انجام شود خیلی خوب می‌شد. یعنی فقط مشخص می‌کردیم مثلا فلان قسمت از صفحه به فلان شیء جاوااسکریپتی نگاشت می‌شود و پس از آن، هر دوی آنها به طور خودکار با هم هماهنگ می‌بودند. به این مفهوم یا ایده، *مقیدسازی داده* گفته می‌شود. در Angular ما دقیقا همین ایده را پیاده‌سازی کردیم. زیرا متوجه شدیم این ایده در کنار الگوی MVC نوشتن نماها و مدل‌ها را بسیار ساده می‌کند. طبق این مدل، بیشتر کاری که برای منتقل کردن داده‌ها از محلی به محل دیگر (مثلا از کُدهای جاوااسکریپتی به DOM یا بالعکس) یا هماهنگ نگه داشتن DOM و اشیاء جاوااسکریپتی لازم است کاملا به شکل خودکار انجام می‌شود و دیگر نیازی نیست برنامه‌نویس، دغدغه‌ی این کار را داشته باشد.

بگذارید این مسئله را به طور عملی ببینیم. برای این کار از مثال قبلی استفاده می‌کنیم و کمی آن را پویا می‌کنیم. همان طور که دیدید تنها کاری که تابع HelloController انجام می‌داد این بود که خصوصیت text شیء جاوااسکریپتی greeting را که در $scope تعریف شده بود با مقدار 'Hello' تنظیم می‌کرد. اما پس از آن کلمه‌ی Hello دیگر تغییر نمی‌کرد. برای پویا کردن این مثال بگذارید جعبه متنی اضافه کنیم تا وقتی کاربر چیزی در آن وارد می‌کند، همزمان با تایپ او، مقدار وارد شده درون greeting.text نیز قرار بگیرد.

الگوی جدید صفحه بدین صورت است:

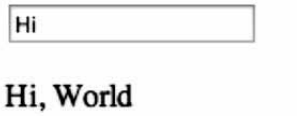
```
<html ng-app>
  <head>
    <script src="angular.js"></script>
    <script src="controllers.js"></script>
  </head>
  <body>
    <div ng-controller='HelloController'>
      <input ng-model='greeting.text'>
      <p>{{greeting.text}}, World</p>
    </div>
  </body>
</html>
```

همان گونه که می‌بینید چیز خاصی در مورد این الگو وجود ندارد. تنها یک تگ <input> برای جعبه متن به صفحه اضافه کرده‌ایم. ممکن است باور نکنید. اما کنترلر جاوااسکریپتی HelloController به هیچ تغییری نیاز ندارد. اگر این مثال را دوباره در مرورگرتان اجرا کنید شکلی مانند شکل ۲-۱ خواهید دید.



شکل ۲-۱. وضعیت پیش فرض برنامه‌ی ساده‌ی Hello, World

اکنون اگر داخل جعبه متن، کلمه‌ی Hello را با Hi عوض کنید، چیزی شبیه شکل ۳-۱ خواهید دید.



شکل ۳-۱. برنامه‌ی ساده‌ی Hello, World پس از تغییر محتوای جعبه متن

همان گونه که می‌بینید بدون این که حتی اداره‌گر رویدادی برای جعبه متن بنویسیم، واسط کاربر صفحه به طور خودکار به روز می‌شود! اما مسئله به همین جا ختم نمی‌شود. اگر داخل تابع HelloController که کنترلر صفحه را تشکیل می‌دهد مقداری را داخل $scope.greeting.text قرار بدهیم، این مقدار به طور خودکار داخل جعبه متن قرار می‌گیرد! مهم نیست مقدار را از کجا به دست می‌آوریم. حتی می‌توانیم با استفاده از Ajax درخواستی به سمت سرور

بفرستیم و پاسخ دریافتی را داخل $scope.greeting.text قرار بدهیم. مسئله این است که Angular خودش به طور خودکار، جعبه متن و محتوای صفحه را با یکدیگر هماهنگ نگه می‌دارد.

تزریق وابستگی[1]

باز هم تکرار می‌کنیم. اتفاق‌های زیادی در پشت صحنه رخ می‌دهد که کنترلر HelloController می‌تواند به این سبک کار کند. اما نیازی نیست از آنها خبر داشته باشید یا خودتان چیزی را سر هم کنید تا HelloController بتواند کارش را انجام بدهد. برای نمونه شیء $scope که همین جا بگوییم تمام مقیدسازی داده زیر سر همین شیء است، توسط Angular به HelloController داده می‌شود و نیازی نیست خودمان آن را ایجاد کنیم یا بدانیم چطور ایجاد می‌شود. به این تکنیک، *تزریق وابستگی* گفته می‌شود. یعنی نیازمندی HelloController موقع فراخوانی این تابع به طور خودکار از بیرون به آن تزریق می‌شود.

همان گونه که در فصل‌های بعد خواهید دید، شیء $scope یکی از اشیاء بیشماری است که Angular فراهم می‌کند. برای نمونه شیء دیگری به نام $location وجود دارد که توسط آن می‌توانیم مقیدسازی را بر اساس آدرس URL مرورگر انجام بدهیم. یعنی می‌توانیم به صورتی که در زیر نشان داده‌ایم پارامتر دیگری به نام $location برای HelloController تعریف کنیم و داخل HelloController از آن استفاده کنیم. بار دیگر AngularJS موقع فراخوانی HelloController به طور خودکار شیء $location را به کنترلر HelloController پاس می‌دهد.

```
function HelloController($scope, $location) {
  $scope.greeting = { text: 'Hello' };
  // use $location for something good here...
}
```

همان طور که گفتیم به این تکنیک تزریق وابستگی گفته می‌شود. در این تکنیک، به جای این که وابستگی‌های یک کلاس را تابع را خودمان ایجاد کنیم، تنها آنها را درخواست می‌کنیم. تکنیک تزریق وابستگی بر اساس الگویی به نام *قانون دیمیتر*[2] یا *قانون کمترین دانش*[3] کار می‌کند. قانون دیمیتر یک راهبرد طراحی و حالت ویژه‌ای از *اتصال سُست*[4] است. به طور خلاصه این راهبرد می‌گوید هر واحد نرم‌افزاری باید کمترین دانش و اطلاعات را نسبت به سایر واحدها داشته باشد.

در مثال فعلی ما نیز از آنجایی که تنها وظیفه‌ی کنترلر HelloController این است که وضعیت اولیه‌ی مدل (یعنی شیء greeting) را فراهم کند، طبق قانون دیمیتر نباید به چیز دیگری کار داشته باشد. مثلا بداند $scope چگونه ساخته می‌شود یا از کجا به دست می‌آید.

---

[1] Dependency Injection
[2] Law of Demeter
[3] Prnciple of Least Knowledge
[4] Loose Coupling

رهنمودها[5]

یکی از بهترین خصلت‌های AngularJS این است که می‌توانید الگوها را به صورت HTML محض بنویسید. در قلب فریم‌وُرک AngularJS موتور قدرتمندی برای کار با DOM گنجانده‌ایم که با استفاده از آن حتی می‌توانید تگ‌های HTML را توسعه بدهید و برای خودتان تگ جدیدی تعریف کنید.

اگر تا اینجا دقت کرده باشید در الگوی HTML مثال Hello World چیزهایی به کار بردیم که جزو استاندارد HTML نیست. برای نمونه داخل تگ و عبارتی را بین کاراکترهای آکلاد قرار داده و برای مقیدسازی از آن استفاده کردیم، تگ را با ویژگی ng-controller تزئین کردیم و بدین ترتیب مشخص کردیم چه کنترلری قرار است آن قسمت را مدیریت کند. همچنین با استفاده از ویژگی ng-model در تگ <input>، جعبه متن ایجاد شده را به یک شیء جاوااسکریپتی یا در واقع مدل برنامه مقید کردیم. در AngularJS به همه‌ی این ویژگی‌های عجیب و غریب، دایرکتیو (directive) یا به طور رسمی‌تر *«رهنمود توسعه‌ی HTML»*[6] می‌گوییم.

در Angular رهنمودهای بسیار زیادی وجود دارد که می‌توانید با استفاده از آنها در الگوها کارهای زیادی انجام بدهید. کمی جلوتر تعداد بیشتری از این رهنمودها را نشان می‌دهیم. با استفاده از این رهنمودها می‌توانید شکل واسط کاربر یا الگوی برنامه را مشخص کرده یا مولفه‌های پیچیده‌ای درست کنید و آنها را در نماها به کار ببرید. حتی می‌توانید خودتان هم رهنمود بنویسید و با توسعه‌ی قابلیت‌های HTML، هر چیزی را که در ذهن و خیال‌تان می‌پرورانید پیاده‌سازی کنید!

یک مثال: سبد خرید

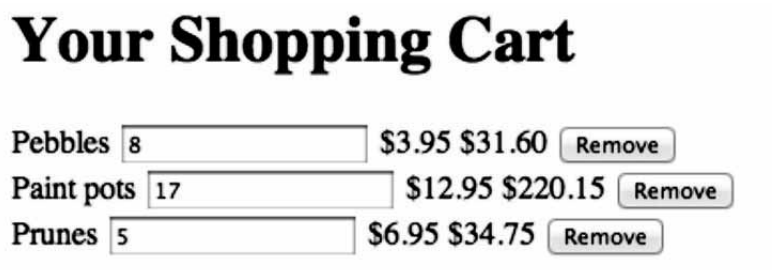
برای نشان دادن قابلیت‌های بیشتری از Angular، در این قسمت یک سبد خرید ساده درست می‌کنیم که کاربر با استفاده از آن بتواند محتویات سبد خرید را تغییر بدهد. کاری با زیبایی سبد خرید نداریم. لذا آن را بسیار ساده طراحی می‌کنیم. به جهت ساده‌تر شدن این مثال، کنترلر سبد خرید را که اسمش را CartController گذاشته‌ایم داخل همین الگو با استفاده از تگ <script> تعریف می‌کنیم و مانند مثال قبلی، کُد این تابع جاوااسکریپتی را داخل یک فایل مجزا قرار نمی‌دهیم. اما فراموش نکنید این تنها یک مثال است و در یک برنامه‌ی واقعی توصیه می‌کنیم کنترلرها را از الگوها جدا کنید. کُد این مثال بدبن صورت است:

```
<html ng-app>
  <head>
    <title>Your Shopping Cart</title>
  </head>
  <body ng-controller='CartController'>
    <h1>Your Order</h1>
    <div ng-repeat='item in items'>
      <span>{{item.title}}</span>
      <input ng-model='item.quantity'>
      <span>{{item.price | currency}}</span>
      <span>{{item.price * item.quantity | currency}}</span>
```

[5] Directives
[6] HTML extension directive

```
      <button ng-click="remove($index)">Remove</button>
    </div>
    <script src="angular.js"></script>
    <script>
        function CartController($scope) {
          $scope.items = [
            {title: 'Paint pots', quantity: 8, price: 3.95},
            {title: 'Polka dots', quantity: 17, price: 12.95},
            {title: 'Pebbles', quantity: 5, price: 6.95}
          ];
          $scope.remove = function(index) {
            $scope.items.splice(index, 1);
          }
        }
    </script>
  </body>
</html>
```

اگر این صفحه را در یک مرورگر بارگذاری کنید، چیزی مانند شکل ۴–۱ را خواهید دید.



شکل ۴–۱. طراحی سبد خرید با استفاده از AngularJS

اکنون اجزاء Angular استفاده شده در الگوی این صفحه را به طور سریع توضیح می‌دهیم. اگر توضیحات را خیلی متوجه نشدید نگران نباشید. در مابقی کتاب به طور دقیق و مفصل به همه‌ی آنها خواهیم پرداخت.

از بالای صفحه شروع می‌کنیم.

```
<html ng-app>
```

اولین چیزی که در برنامه‌های Angular باید بدانید ویژگی ng-app است. با استفاده از این ویژگی، حوزه‌ی فعالیت Angular را مشخص کرده و به Angular اعلام می‌کنیم چه بخشی از صفحه را باید مدیریت کند. از آنجایی که این ویژگی را در تگ <html> به کار برده‌ایم، از Angular می‌خواهیم کل صفحه را مدیریت کند. بیشتر مواقع هم همین را می‌خواهیم. اما اگر بخواهید Angular را در برنامه‌ای به کار ببرید که از روش‌های دیگری برای مدیریت صفحه استفاده می‌کند می‌توانید این ویژگی را به تنها بخشی از صفحه (مثلا یک المان) اعمال کنید.

ویژگی بعدی ng-controller است.

```
<body ng-controller='CartController'>
```

در Angular برای مدیریت بخش‌های قسمتی از صفحه که با ng-app مشخص شده، از کلاس‌های جاوااسکریپتی استفاده می‌شود که کنترلرهای برنامه را در الگوی MVC تشکیل می‌دهند. در اینجا ویژگی ng-controller را به تگ <body> اعمال کرده و مشخص کرده‌ایم Angular برای مدیریت هر چیزی که بین <body> و </body> قرار دارد باید از کنترلر CartController استفاده کند.

ویژگی بعدی ng-repeat است.

```
<div ng-repeat='item in items'>
```

این ویژگی یکی از ویژگی‌های بسیار زیبا و مفید Angular است. در اینجا با استفاده از این ویژگی به Angular می‌گوییم به ازای هر یک از عناصر آرایه‌ی items، المان فعلی (یعنی تگ) را در موقعیت فعلی‌اش در DOM تکرار کند. موقع کپی کردن المان ، Angular ویژگی‌ای به نام item با مقدار المان جاری آرایه‌ی items به آن اضافه می‌کند که می‌توانیم برای مقیدسازی‌های بعدی از آن استفاده کنیم. همان طور که خروجی صفحه نشان می‌دهد، از آنجایی که در کنترلر CartController درون $scope آرایه‌ای به نام items با سه عنصر تعریف کرده‌ایم، ویژگی ng-repeat باعث شده سه المان در صفحه ایجاد شود که داخل هر کدام آنها عنوان هر یک از کالاهای سبد خرید، تعداد سفارش، قیمت واحد، جمع کل و دکمه‌ای که برای حذف کالا از سبد خرید به کار می‌رود قرار می‌گیرد.

عنصر بعدی یک عبارت مقیدسازی است که داخل تگ‌های به کار برده‌ایم:

```
<span>{{item.title}}</span>
```

همان طور که در مثال ساده‌ی Hello, World دیدید با استفاده از {{ }} می‌توانیم عبارت مقیدسازی تعریف کنیم و مقدار یک متغیر جاوااسکریپتی را در آن نمایش بدهیم. بدین ترتیب المان HTML به متغیر جاوااسکریپتی مقید شده و محتوای آن همگام با تغییرات متغیر جاوااسکریپتی به طور خودکار به روز می‌شود. در اینجا نیز عبارت مقیدسازی {{item.title}} مقدار خصوصیت title مربوط المان جاری آرایه (یعنی عنوان کالا) را درون تگ قرار می‌دهد.

ویژگی بعدی ng-model است.

```
<input ng-model='item.quantity'>
```

با استفاده از ویژگی ng-model، مقدار جعبه متن مربوط به تگ <input> را به item.quantity مقید کرده‌ایم. این مقیدسازی دو طرفه است. یعنی اگر item.quantity تغییر کند، محتوای جعبه متن نیز تغییر می‌کند. از آن سو اگر محتوای جعبه متن تغییر کند، مقدار item.quantity نیز تغییر خواهد کرد. در حالی که مقیدسازی {{ }} یک طرفه است و تنها به طور ساده می‌گوید مقداری را در اینجا درج کن. از آنجایی که می‌خواهیم به کاربر امکان بدهیم بتواند تعداد سفارش هر کالا را تغییر بدهد، مقیدسازی {{ }} کارمان را راه نمی‌اندازد. لذا از ng-model استفاده کرده‌ایم.

همان گونه که گفتیم ویژگی ng-model باعث می‌شود متغیر جاوااسکریپتی item.quantity و جعبه متن ایجاد شده همیشه با یکدیگر هماهنگ باشند.

در مرحله‌ی بعد برای نمایش قیمت جزء و قیمت کل هر کالا (که از ضرب قیمت جزء در تعداد سفارش کالا به دست می‌آید) از دو عبارت مقیدسازی به شیوه‌ی آکلادی استفاده کرده‌ایم. اما این عبارت‌ها کمی با عبارت‌های قبلی فرق دارند. زیرا در آنها از عملگرهای ریاضی و قالب‌بندی استفاده کرده‌ایم.

```
<span>{{item.price | currency}}</span>
<span>{{item.price * item.quantity | currency}}</span>
```

در اینجا با استفاده از currency | به Angular اطلاع می‌دهیم که می‌خواهیم قیمت واحد و جمع کل با استفاده از کاراکتر دلار قالب‌بندی شود. یکی از قابلیت‌های Angular، اعمال فیلتر است و توسط آن می‌توانید رشته‌های متنی را پیش از استفاده قالب‌بندی کرده و تبدیلی را روی آنها اجرا کنید. از بین فیلترهای درون–ساخت Angular فیلتری به نام currency وجود دارد که یک رشته‌ی عددی را با کاراکتر دلار قالب‌بندی می‌کند و در اینجا از این فیلتر استفاده کرده‌ایم. در فصل بعد به طور مفصل‌تر با فیلترها آشنا خواهید شد.

آخرین جزء Angular در الگوی این مثال ویژگی ng-click است:

```
<button ng-click='remove($index)'>Remove</button>
```

توسط تگ <button> دکمه‌ای با برچسب Remove کنار هر کالا ایجاد می‌کنیم و بدین ترتیب به کاربر امکان می‌دهیم بتواند کالاها را با کلیک کردن دکمه‌ی Remove کنار آنها از سبد خرید حذف کند. دکمه‌ی Remove را طوری تنظیم کرده‌ایم که کلیک کردن آن باعث فراخوانی تابعی به اسم ()remove شود. موقع فراخوانی متد ()remove، متغیری به نام $index به آن پاس می‌دهیم که اندیس آیتم جاری آرایه در حلقه‌ی ng-repeat را در بر دارد ($index یکی دیگر از اشیاء درون–ساخت Angular مشابه $scope و $location است). بدین ترتیب با استفاده از $index متوجه می‌شویم کدام کالا را باید حذف کنیم.

اکنون بگذارید کنترلر CartController را بررسی کنیم.

```
function CartController($scope) {
```

کنترلر CatController منطق سبد خرید را مدیریت می‌کند. مانند گذشته برای این کنترلر نیز پارامتری به نام $scope تعریف کرده و بدین ترتیب به Angular می‌گوییم کنترلر ما به شیء $scope نیاز دارد. شیء $scope همان چیزی است که توسط آن می‌توانیم مقیدسازی اشیاء و داده‌های جاوااسکریپتی به واسط کاربر را انجام بدهیم.

داخل $scope آرایه‌ای به نام items تعریف کرده‌ایم. این آرایه توسط ویژگی ng-repeat استفاده می‌شود و لیست کالاهای سبد خرید را در بر دارد.

```
$scope.items = [
  {title: 'Paint pots', quantity: 8, price: 3.95},
  {title: 'Polka dots', quantity: 17, price: 12.95},
  {title: 'Pebbles', quantity: 5, price: 6.95}
];
```

البته در یک برنامه‌ی واقعی، طبعا اقلام سبد خرید نمی‌تواند بدین شکل تعریف شود و برای نگهداری طولانی مدت آنها باید روش دیگری به کار ببرید. مثلا کالاها را در سمت سرور یا داخل کوکی نگهداری کنید که حتی با بستن

پنجره‌ی مرورگر باز هم بتوانید آنها را به کاربر نشان بدهید. اما فعلا کاری با این مسائل نداریم و در این رابطه در فصل‌های بعد صحبت خواهیم کرد.

آخرین چیزی که می‌ماند تابع ()remove است که آن را به عنوان اداره‌گر رویداد کلیک دکمه‌های Remove مشخص کرده‌ایم. این تابع را نیز باید در $scope قرار بدهیم.

```
$scope.remove = function(index) {
  $scope.items.splice(index, 1);
}
```

در این سبد خرید کالاها فقط در حافظه نگهداری می‌شود. از این رو برای حذف یک کالا تنها کافی است آن را از آرایه‌ی $scope.items حذف کنیم. برای این کار از تابع ()splice استفاده می‌کنیم. با توجه به این که ng-repeat به ازای هر کالا یک المان تولید کرده، آن المان‌ها به آرایه‌ی $scope.items مقید شده‌اند. لذا با حذف آیتمی از آرایه‌ی $scope.items، به طور خودکار المان مربوط به آن نیز از DOM حذف می‌شود.

پایان سخن

در این فصل نگاهی سریع و سطحی به پایه‌ای‌ترین اصطلاحات Angular انداخته و چند مثال بسیار ساده را مشاهده کردیم. در مابقی فصل‌های کتاب به طور مفصل‌تر نشان می‌دهیم فریم‌وُرک Angular واقعا چه چیزهایی برای عرضه دارد.

فصل ۲. آناتومی یک برنامه‌ی AngularJS

در AngularJS بر خلاف سایر کتابخانه‌های جاوااسکریپتی که هر تابعی را می‌توانید بردارید و استفاده کنید، تمام چیزها بر اساس یک مدل مشخص طراحی شده که در کنار هم و به کمک هم کار می‌کنند. در این فصل تمام زیربنای ساختمان Angular را توضیح می‌دهیم تا به طور دقیق ببینید اجزای Angular چطور در کنار هم چیده شده و با هم کار می‌کنند.

فعال کردن Angular

هر برنامه‌ای برای فعال کردن Angular باید دو کار انجام بدهد.

۱. کتابخانه‌ی angular.js را بارگذاری کند.
۲. با استفاده از رهنمود ng-app به Angular بگوید کدام بخش از صفحه یا DOM را باید مدیریت کند.

بارگذاری کتابخانه‌ی AngularJS

بارگذاری کتابخانه‌ی Angular کار راحتی است و آن را مانند هر کتابخانه‌ی جاوااسکریپتی دیگری می‌توانید بارگذاری کنید. می‌توانید اسکریپت Angular را از شبکه‌ی تحویل محتوای Google یا Google CDN به صورت زیر در صفحه‌ی خود بارگذاری کنید:

```
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.0.4/angular.min.js"></script>
```

توصیه هم می‌کنیم از شبکه‌ی تحویل محتوای Google استفاده کنید، زیرا سرورهای Google بسیار سریع و قدرتمند هستند. بدین ترتیب کتابخانه‌ی Angular بین برنامه‌های مختلفی که می‌نویسید توسط مرورگر کاربر کَش می‌شود و اگر کاربر شما از برنامه‌ی Angular دیگری استفاده کند، مرورگر وی قبلا کتابخانه‌ی Angular را دانلود کرده و نیازی نیست دوباره آن را بار کند. لذا برنامه‌ی وب شما سریع‌تر برای کاربر اجرا می‌شود.

با این حال می‌توانید کتابخانه‌ی Angular را به صورت محلی (یا هر جای دیگری) هم میزبانی کنید. فقط دقت کنید آدرس درست را در ویژگی src تگ <script> خود به کار ببرید.

تعریف محدودیت‌های کاری Angular با استفاده از ng-app

همان گونه که دیدید با استفاده از رهنمود ng-app به Angular می‌گوییم چه بخشی از صفحه را باید مدیریت کند. اگر می‌خواهید برنامه‌ی شما به طور کامل با Angular کار کند، آن را به صورت زیر به تگ <html> اعمال کنید.

```
<html ng-app>
...
</html>
```

اما اگر از قبل برنامه‌ای دارید که تکنولوژی دیگری مانند Java یا Rails نیز قرار است DOM را مدیریت کند، می‌توانید ویژگی ng-app را به بخشی از صفحه مانند یک تگ اعمال کرده و به Angular بگویید فقط همان بخش را مدیریت کند.

```
<html>
  ...
  <div ng-app>
    ...
  </div>
  ...
</html>
```

Model View Controller

در فصل ۱ گفتیم Angular از الگوی MVC پشتیبانی می‌کند. با وجودی که در برنامه‌های Angular انعطاف‌پذیری بسیار زیادی دارید اما بیشتر مواقع برنامه‌ی شما چنین رنگ و بویی دارد:

- اشیاء مدلی دارد که وضعیت جاری برنامه را نگهداری می‌کنند.
- نماهایی دارد که برای نمایش داده‌های مدل به کار می‌روند.
- کنترلرهایی دارد که رابطه بین مدل‌ها و نماها بوده و آنها را مدیریت می‌کنند.

برای ایجاد مدل‌ها از اشیاء جاوااسکریپتی یا حتی نوع داده‌های پایه (متغیرهای ساده) استفاده می‌شود. هیچ چیز به خصوصی در رابطه با داده‌های مدل وجود ندارد. برای نمونه اگر تنها می‌خواهید متنی را به کاربر نمایش بدهید می‌توانید به طور ساده از رشته‌ای مانند زیر استفاده کنید:

```
$scope.someText = 'You have started your journey.';
```

برای ایجاد نما از الگوهای HTML استفاده شده و اجزای واسط کاربر به همراه رهنمودهای Angular در آنها گنجانده می‌شود. وقتی صفحه بارگذاری شده و Angular نما را با داده‌های مدل ترکیب می‌کند، نمای نهایی قابل مشاهده به دست می‌آید. همان گونه که دیدید می‌توانید به صورت زیر، محلی را در DOM تعریف کنید که داده‌های جاوااسکریپتی در آن قرار بگیرد.

```
<p>{{someText}}</p>
```

در Angular به این قاعده‌ی دو آکلادی، *درون‌یابی* یا interpolation می‌گوییم.

کنترلرها نیز کلاس‌های جاوااسکریپتی هستند که داده‌های مدل را تعریف کرده و منطق برنامه را در بر می‌گیرند. برای این که مدل‌ها توسط نما قابل دسترس و استفاده باشند باید آنها را داخل $scope قرار بدهید. همان طور که دیدید موقع اجرای یک کنترلر، شیء $scope به طور خودکار توسط Angular به کنترلر پاس داده می‌شود.

```
function TextController($scope) {
  $scope.someText = 'You have started your journey.';
}
```

اگر این سه جزء را به یکدیگر بچسبانیم چنین چیزی خواهیم داشت:

```
<html ng-app>
```

```
  <body ng-controller="TextController">
    <p>{{someText}}</p>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.0.1/angular.min.js"></script>
    <script>
      function TextController($scope) {
        $scope.someText = 'You have started your journey.';
      }
    </script>
  </body>
</html>
```

اکنون اگر این صفحه را توسط یک مرورگر باز کنید چنین چیزی خواهید دید:

You have started your journey.

با وجودی که این سبک پایه‌ای و ساده برای تعریف مدل‌ها جواب می‌دهد، اما در بیشتر برنامه‌ها برای نگهداری داده‌های مدل خود به جای استفاده از نوع داده‌های پایه‌ای (مانند رشته و اعداد صحیح و غیره)، بهتر است یک شیء واقعی تعریف کنید. در این قسمت یک شیء مدل به نام messages ایجاد می‌کنیم و رشته‌ی مورد نمایش را در آن تعریف می‌کنیم. یعنی به جای ...

```
$scope.someText = 'You have started your journey.';
```

چنین چیزی می‌نویسیم:

```
var messages = { someText : 'You have started your journey.' };
$scope.messages = messages;
```

سپس آن را به صورت زیر در الگوی برنامه به کار می‌بریم:

```
<p>{{messages.someText}}</p>
```

همان طور که کمی جلوتر خواهید دید، وقتی مدل خود را به این شکل ایجاد می‌کنید از رفتارهای غیر منتظره‌ای که ممکن است به دلیل وراثت پروتوتایپی در اشیاء $scope رخ بدهد جلوگیری می‌شود.

در طول این کتاب رهیافت‌هایی به شما یاد می‌دهیم که کار شما را ساده می‌کند. در مثال فعلی، کنترلر TextController را در حوزه‌ی دید سراسری تعریف کردیم. با وجودی که در مثال‌های ساده چنین کاری ایراد ندارد و جواب می‌دهد، اما روش صحیح تعریف کنترلرها این است که آنها را به عنوان بخشی از واحد سازمان‌دهی بزرگ‌تری به نام *ماجول* یا *پیمانه* تعریف کنید. یک *پیمانه* فضای نامی برای بخش‌های مرتبط با هم در یک برنامه است. اگر بخواهیم مثال قبلی را با ماجول بنویسیم چنین چیزی خواهیم داشت:

```
<html ng-app='myApp'>
  <body ng-controller='TextController'>
    <p>{{messages.someText}}</p>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.0.1/angular.min.js"></script>
    <script>
        var myAppModule = angular.module('myApp', []);
        myAppModule.controller('TextController',
        function($scope) {
          var messages = { someText : 'You have started your journey.' };
```

```
            $scope.messages = messages;
        });
    </script>
  </body>
</html>
```

در اینجا برای ویژگی ng-app نام یک ماجول به نام myApp را مشخص کرده‌ایم. سپس با استفاده از شیء angular، ماجولی به نام myApp ایجاد کرده و با فراخوانی تابع ()controller برای شیء ماجول، کنترلر خود را به آن پاس داده‌ایم.

کمی جلوتر به همه‌ی سوال‌های شما در این باره که چرا باید از ماجول استفاده کنیم و چگونه باید از آنها استفاده کنیم پاسخ خواهیم داد. فعلا فقط بدانید تعریف نکردن چیزهای مختلف مانند کنترلرها در فضای نام سراسری رهیافت بسیار خوبی است و ماجول‌ها نیز مکانیزمی هستند که در Angular برای این کار از آنها استفاده می‌کنیم.

الگوها و مقیدسازی داده

در برنامه‌های Angular الگوها به طور ساده چیزی جز فایل‌های HTML نیستند که یا از سمت سرور بارگذاری می‌شوند یا مانند سایر منبع استاتیک، آنها را داخل تگ <script> تعریف می‌کنید. داخل این الگوها واسط کاربر برنامه با استفاده از المان‌های معمولی HTML و رهنمودهای Angular تعریف می‌شود. بعد از بارگذاری یک الگو، Angular با ترکیب نمودن تگ‌های آن با داده‌های مدل‌های جاوااسکریپتی، الگو را به یک برنامه‌ی کامل و قابل نمایش تبدیل می‌کند. در فصل ۱ موقع نمایش لیست کالاهای سبد خرید مثالی از این مسئله را نشان دادیم.

```
<div ng-repeat="item in items">
  <span>{{item.title}}</span>
  ...
</div>
```

همان گونه که دیدید ویژگی ng-repeat کُد بالا باعث می‌شود به ازای هر یک از آیتم‌های لیست items که مدل برنامه را تشکیل می‌دهد، المان و همه‌ی محتویات داخل آن تکرار شود. شیء مدل items را نیز به صورت یک آرایه در خود کنترلر تعریف کردیم. چنین کاری برای آزمایش واسط کاربر و بررسی سریع کارکرد آن مفید است. اما در برنامه‌های واقعی، داده‌های مدل باید در یک جور منبع داده‌ی ماندگار در سمت سرور نگهداری شود. پس از آن برنامه به سرور متصل شده و داده‌های مورد نیاز قسمتی از برنامه را که کاربر فعلی در حال مشاهده‌ی آن است درخواست می‌کند. در نهایت پس از دریافت داده‌ها، Angular آنها را با الگوی صفحه ترکیب کرده و نمای قابل مشاهده‌ی صفحه را تولید یا به روز می‌کند.

بگذارید این سیر را ببینیم. روند این کار بدین صورت است:

۱. کاربری برای اولین بار برنامه را درخواست می‌کند.
۲. مرورگر او یک اتصال HTTP به سرور ایجاد کرده و منتظر پاسخ می‌ماند.
۳. با توجه به این که مرورگر صفحه‌ی مشخصی را درخواست نکرده، وب‌سرور فایل پیش فرض index.html را به سمت او می‌فرستد. این فایل نیز حاوی الگوی کلی برنامه است.
۴. سپس مرورگر فایل index.html دریافت شده را بارگذاری می‌کند.

۵. پس از آن مرورگر سایر اجزای صفحه مانند اسکریپت‌های خارجی و تصاویر را بارگذاری می‌کند. یکی از این اسکریپت‌ها نیز کتابخانه‌ی Angular است.
۶. پس از بارگذاری Angular، کُدهای آن توسط مرورگر اجرا می‌شود. Angular نیز صبر می‌کند سایر اجزای صفحه به طور کامل بارگذاری شود. پس از آن در DOM دنبال المانی با ویژگی ng-app می‌گردد تا ناحیه‌ای را که باید مدیریت کند پیدا کند.
۷. سپس سایر المان‌های DOM را به دنبال سایر ویژگی‌ها و رهنمودهای Angular بررسی کرده و تغییرات لازم را اعمال می‌کند. این مسئله شامل وصل کردن اداره‌گرهای رویداد، دستکاری المان‌ها، ایجاد المان‌های جدید و همچنین بازیابی اطلاعات از سرور نیز می‌شود. نتیجه‌ی این پردازش این است که الگو به شکل نمای نهایی در می‌آید و DOM نیز پایدار می‌شود.
۸. پس از پایدار شدن صفحه می‌توانید در صورت تمایل خودتان نیز به سرور راه دور متصل شوید و سایر داده‌های مورد نیاز برنامه را بازیابی کنید.

مرحله‌ی ۱ تا ۶ برای همه‌ی برنامه‌های Angular مشترک بوده و قابل تغییر نیست. تنها در مرحله‌ی ۷ و ۸ است که می‌توانید کاری مطابق میل خود انجام بدهید. این مراحل نیز می‌تواند به شکل همزمان یا غیر همزمان صورت بگیرد. برای تسریع بارگذاری صفحه و افزایش کارایی برنامه می‌توانید داده‌هایی را که قرار است بی‌درنگ پس از بارگذاری صفحه به کاربر نمایش داده شود درون خود الگوی HTML بگنجانید و از درخواست اضافی به سمت سرور خودداری کنید. پس از آن می‌توانید سایر داده‌ها را همگام با استفاده‌ی کاربر از برنامه به تناسب از سرور درخواست کنید.

مدلی که Angular برای برنامه فراهم می‌کند باعث می‌شود الگوها از داده‌های مورد نیاز آنها مجزا شوند. مهمترین دستاورد این مدل این است که اکنون الگوها نیز کاملا استاتیک هستند و در نتیجه مانند کُدهای جاوااسکریپت، تصاویر، شیوه‌های CSS و سایر منابع استاتیک قابل کَش شدن هستند. بدین ترتیب پس از آن که صفحه برای اولین بار بارگذاری می‌شود در مراجعات بعدی کاربر، فقط کافی است داده‌های برنامه دانلود شود. این مسئله می‌تواند سرعت و کارایی برنامه را به میزان چشمگیری افزایش بدهد.

نمایش متن

یکی از کاربردهای مهم برنامه‌های وب، نمایش متن در صفحه بر اساس داده‌های جاوااسکریپتی است. برای این کار می‌توانید از رهنمود ng-bind استفاده کنید و رشته‌های متنی را در هر جایی از صفحه که بخواهید نمایش بدهید یا هر متنی را در هر جایی از صفحه به روز کنید. این رهنمود به دو شکل قابل استفاده است. یکی از شکل‌های آن استفاده از کاراکترهای دو آکلادی یا عبارت‌های مقیدسازی بود که قبلا دیدیم.

```
<p>{{greeting}}</p>
```

شکل دیگر استفاده از رهنمود ng-bind این است که آن را به صورت یک ویژگی در تگی که قرار است متن را در بر بگیرد استفاده کنید:

```
<p ng-bind="greeting"></p>
```

خروجی هر دو حالت یکسان است و تفاوتی بین این دو روش نیست.

اکنون اگر متغیر greeting را با مقدار "Hi there" تنظیم کنیم، Angular کُد HTML زیر را تولید می‌کند:

```
<p>Hi there</p>
```

مرورگر نیز متن "Hi there" را نمایش می‌دهد.

اما واقعا از بین این دو حالت کدام یک را باید استفاده کنیم و علت استفاده‌ی یکی نسبت به دیگری چیست؟ مزیت قاعده‌ی درون‌یابی دوآکلادی این است که خواناتر است و کمتر به تایپ نیاز دارد. اما یک نقطه ضعف دارد. وقتی الگوی index.html برای اولین بار در حال بارگذاری است ممکن است (مثلا به دلیل کُندی شبکه یا زیر بار بودن وب‌سرور) بارگذاری صفحه‌ی index.html با کُندی پیش برود (مثلا مرورگر نیمی از آن را دریافت کرده، آن را نمایش بدهد و منتظر دریافت مابقی الگو باشد). همان گونه که دیدید تنها پس از بارگذاری کامل الگو است که Angular آن را تفسیر می‌کند. در چنین شرایطی این احتمال وجود دارد که کاربر، الگوی پردازش نشده و در واقع خود عبارت‌های مقیدسازی را مشاهده کند. طبعا چنین چیزی زیبا نیست.

البته این مسئله فقط اولین باری که الگو بارگذاری می‌شود رخ می‌دهد. پس از آن الگو توسط مرورگر کَش شده و دیگر چنین مشکلی پیش نمی‌آید. لذا بیشتر مواقع قاعده‌ی دوآکلادی مشکلی ایجاد نمی‌کند (به ویژه اگر الگوی شما سبک باشد جای نگرانی نیست). با این حال اگر احتمال می‌دهید که الگو با کُندی بارگذاری شود می‌توانید برای مقیدسازی از ویژگی ng-bind استفاده کنید. در این حالت تا زمانی که صفحه به طور کامل بارگذاری نشده عبارت‌های مقیدسازی دیده نخواهد شد.

ورودی فرم‌ها

در Angular کار با المان‌های ورودی فرم‌ها مانند جعبه متن، دکمه‌ی رادیویی، چک‌باکس، لیست و نظایر آن بسیار ساده است. همان طور که در مثال‌های پیشین دیدید، می‌توانید با استفاده از ویژگی ng-model المان‌ها را به خصوصیت‌های مدل مقید کنید. این مسئله برای انواع و اقسام المان‌های ورودی مانند جعبه متن، دکمه‌ی رادیویی، چک‌باکس، لیست و نظایر آن قابل اجرا است. برای نمونه می‌توانیم یک چک‌باکس را به صورت زیر به خصوصیت یک مدل جاوااسکریپتی مقید کنیم:

```
<form ng-controller="SomeController">
  <input type="checkbox" ng-model="youCheckedIt">
</form>
```

نتیجه‌ی این مقیدسازی این است که:

۱. وقتی کاربر چک‌باکس را تیک می‌زند مقدار خصوصیت youCheckedIt در شیء $scope در کنترلر SomeController به طور خودکار true می‌شود. همچنین برداشتن تیک چک‌باکس، مقدار آن خصوصیت را false می‌کند.
۲. از آن سو اگر خصوصیت $scope.youCheckedIt را از طریق کُدهای جاوااسکریپت true کنید، چک‌باکس نیز توسط Angular تیک زده می‌شود و false کردن $scope.youCheckedIt، تیک چک‌باکس را بر می‌دارد.

اکنون فرض کنید بخواهیم موقع کلیک شدن چک‌باکس، خودمان هم کاری انجام بدهیم. در روش سنتی در چنین حالتی باید رویداد کلیک را اداره کنیم. اما در Angular راه مشابه دیگری برای این کار وجود دارد. در جعبه متن‌ها با استفاده از ویژگی ng-change می‌توانید یک تابع جاوااسکریپتی مشخص کنید که وقتی کاربر مقدار جعبه متن را تغییر می‌دهد به طور خودکار فراخوانی شود. بگذارید یک مثال ساده نشان بدهیم. محاسبه‌گر ساده‌ای درست می‌کنیم که بر اساس مقداری که کاربر مشخص می‌کند، سرمایه‌ی مورد نیاز او را محاسبه کند:

```
<form ng-controller="StartUpController">
  Starting: <input ng-change="computeNeeded()" ng-model="funding.startingEstimate">
  Recommendation: {{funding.needed}}
</form>
```

برای ساده‌تر شدن مثال، مقدار سرمایه را با ۱۰ برابر کردن مقدار وارد شده توسط کاربر محاسبه کرده و از نوشتن کُد یک کسب وکار واقعی برای این مسئله خودداری می‌کنیم. برای شروع نیز از مقدار پیش فرض صفر استفاده می‌کنیم. داریم:

```
function StartUpController($scope) {
  $scope.funding = { startingEstimate: 0 };

  $scope.computeNeeded = function() {
    $scope.funding.needed = $scope.funding.startingEstimate * 10;
  };
}
```

برنامه مشکلی ندارد. وقتی کاربر مقداری را در جعبه متن وارد می‌کند، مقدار سرمایه به طور خودکار محاسبه شده و جلوی برچسب Recommendation نمایش داده می‌شود. با این حال برنامه هنوز کامل نیست و یک مشکل بالقوه دارد. مشکل برنامه این است که مقیدسازی آن یک طرفه است. اگر خصوصیت funding.startingEstimate جای دیگری در برنامه تغییر کند (مثلا پس از دریافت داده‌ای از سمت سرور، مقدار آن را به روز کنیم)، خصوصیت funding.needed به روز نمی‌شود. لذا هماهنگی بین funding.startingEstimate و funding.needed به هم می‌خورد. زیرا در این حالات دیگر تابع ()computeNeeded اجرا نمی‌شود.

باید کاری کنیم صرفنظر از تغییرات funding.startingEstimate همیشه تابع ()computeNeeded اجرا شود تا مقدار برچسب Recommendation همیشه به روز باشد. در چنین حالتی باید از تابع ویژه‌ای به نام $watch در Angular استفاده کنیم. این تابع از طریق شیء $scope قابل دسترس است.

کمی جلوتر $watch را بیشتر توضیح می‌دهیم، اما فعلا بدانید با استفاده از $watch می‌توانید کاری کنید Angular تغییرات یک عبارت را زیر نظر بگیرد و به محض مشاهده‌ی هر تغییری، کاری را که برایش مشخص کرده‌ایم انجام بدهد. در واقع به طور خلاصه یک عبارت و یک callback به $watch می‌دهیم تا Angular آن عبارت را زیر نظر بگیرد و به محض مشاهده‌ی هر تغییری در آن، callback را فراخوانی کند.

بدین ترتیب می‌توانیم خصوصیت funding.startingEstimate را با استفاده از $watch زیر نظر بگیریم و برای callback نیز تابع ()computeNeeded را مشخص کنیم. برای این کار کنترلر StartUpController را به صورت زیر تغییر بدهید:

```
function StartUpController($scope) {
  $scope.funding = { startingEstimate: 0 };

  computeNeeded = function() {
    $scope.funding.needed = $scope.funding.startingEstimate * 10;
  };

  $scope.$watch('funding.startingEstimate', computeNeeded);
}
```

بدین ترتیب وقتی funding.startingEstimate تغییر می‌کند، Angular به طور خودکار تابع ()computeNeeded را فراخوانی می‌کند و مقدار funding.needed به روز می‌شود. لذا برچسب Recommendation همیشه مقدار درست را نمایش می‌دهد.

توجه کنید عبارتی که برای ردگیری به $watch می‌دهیم باید از جنس رشته باشد (بین گیومه قرار بگیرد). این رشته به عنوان یک عبارت Angular ارزیابی می‌شود. داخل این رشته می‌توانید هر عبارتی که بخواهید تعریف کرده و به هر چیزی در $scope دسترسی داشته باشید. از عبارت‌های ساده مانند مثال بالا که فقط از یک خصوصیت $scope تشکیل می‌شود تا عبارت‌های پیچیده. کمی جلوتر در این فصل بیشتر درباره‌ی عبارت‌ها صحبت خواهیم کرد.

با استفاده از $watch می‌توانید مقدار برگشتی توابع را هم زیر نظر بگیرید. البته در مثال فعلی نمی‌توانیم از تابع استفاده کنیم (مثلا تابعی که خصوصیت funding.startingEstimate را بر می‌گرداند). زیرا چنین تابعی همیشه مقدار اولیه‌ی این خصوصیت یعنی مقدار صفر را بر خواهد گرداند.

با توجه به این که با تغییر خصوصیت funding.startingEstimate مقدار خصوصیت funding.needed به طور خودکار تغییر می‌کند نیازی هم نیست در المان <input> با استفاده از ng-change تابع ()computeNeeded را صدا بزنیم. لذا می‌توانیم الگوی برنامه را ساده‌تر کنیم:

```
<form ng-controller="StartUpController">
  Starting: <input ng-model="funding.startingEstimate">
  Recommendation: {{funding.needed}}
</form>
```

البته گاهی ممکن است نخواهیم به ازای هر تغییر بلافاصله کاری انجام بدهید. به جای آن ممکن است بخواهیم به روز رسانی و هماهنگ‌سازی را به صورت دستی انجام بدهیم. مثلا صبر کنیم خود کاربر درخواست به روز رسانی را اعلام کند (مانند یک برنامه‌ی چت یا تکمیل فرم خرید). انتخاب بین این دو حالت به خودتان و نیاز برنامه بستگی دارد.

یکی دیگر از رهنمودهای دیگری که در کار با فرم‌ها مفید واقع می‌شود، رهنمود ng-submit است که در تگ <form> استفاده می‌شود. با استفاده از این رهنمود می‌توانید برای یک فرم، تابعی مشخص کنید که موقع ارسال فرم