Summary of Changes

Using Angular 4

The changes in Angular 4 are largely internal, which means that there are few breaking changes to the examples in Pro Angular. For most projects, the move to Angular 4 will be as simple as upgrading the `@angular` NPM packages and checking that everything works. There are changes to the process for Ahead-Of-Time compilation but those are most easily handled by using the angular-cli package, as described in the next section.

Using angular-cli

Since the release of Angular 2, the angular-cli package has become the standard way to create and manage Angular projects. The release of version 1 of angular-cli has been coordinated with Angular 4 and has matured to the point where it is stable enough to be used for real projects.

The advantage of using angular-cli is that it simplifies the setup process required for Angular projects, meaning that you don't have to manually manage a set of NPM packages, configure the TypeScript compiler and set up a development HTTP server. The process for preparing an Angular 4 application for deployment is also much simpler.

The drawback is that you won't see the low-level detail of how an Angular application works and the way that the different pieces fit together. To strike a balance, this update uses angular-cli to set up projects and for development tooling but doesn't take advantage of its scaffolding features or its support for integrating other packages into the compiled JavaScript modules that it creates. I hope that this will give readers the advantages of a simplified project setup without losing some of the important knowledge about how Angular applications really work.

Using the Revised Chapters

To help you migrate, this update contains complete replacement chapters for the example applications in Part 1 of the book, revised to use both Angular 4.0 and angular-cli. This will give you a step-by-step walkthrough to get used to the angular-cli workflow and to the small changes that Angular 4 requires. You will find complete projects for each of these chapters in the Source Code folder in case you don't want to create them yourself. For these chapters, here are the most substantial changes:

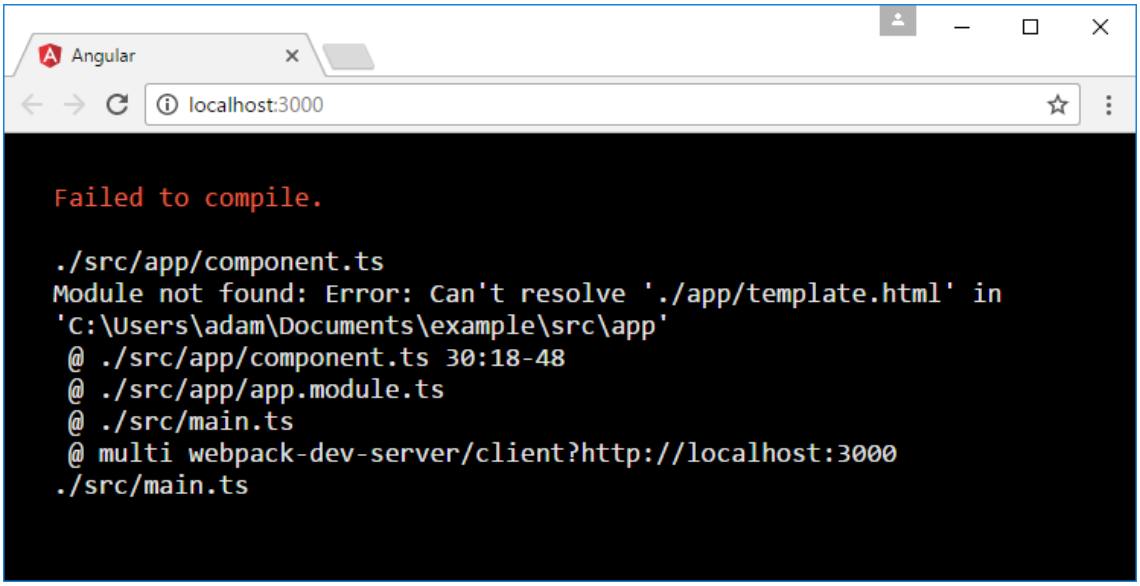
1. Using angular-cli means that you don't have to configure a module loader, TypeScript compiler, NPM packages or set up a development HTTP server.
2. The project structure created by angular-cli puts the application functionality into the `<project>/src/app` folder rather than the `<project>/app` folder that I used in the original examples.
3. You don't have to include `app` in a component's `templateUrl` property because angular-cli uses a different configuration to resolve file paths.

Changes for Other Chapters

For the rest of the book, I have included project source code for each chapter. The following is a detailed list of the changes made to get the source code working, most of which reflect differences in the way that the angular-cli tools are set up.

Common Changes for All Chapters

The way that `angular-cli` resolves the paths to external templates is different from the one I used in the original examples. When following the examples, you may see an error message like this one:



To fix this problem, remove the `app/` prefix from the `templateUrl` property of the component specified in the error.

Changes for Chapter 13

Angular has deprecated the `template` element and replaced it with `ng-template`. All other template features remain unchanged. You can still use the `template` element but the compiler will produce a warning message.

Changes for Chapters 16 and 19

The API that I used to demonstrate how change tracking can performed has changed, which is entirely reasonable since this is an internal feature. The changes require type parameters to be specified for the `DefaultIterableDiffer` class and require the `create` method to be called without parameters, as shown in this replacement Listing 16-16.

Listing 16-16. Updating the Change Detection Code

```
import { Directive, ViewContainerRef, TemplateRef,
             Input, SimpleChange, IterableDiffer, IterableDiffers,
             ChangeDetectorRef, CollectionChangeRecord, DefaultIterableDiffer
} from "@angular/core";
```

```
@Directive({
    selector: "[paForOf]"
})
export class PaIteratorDirective {
    private differ: DefaultIterableDiffer<any>;

    constructor(private container: ViewContainerRef,
        private template: TemplateRef<Object>,
        private differs: IterableDiffers,
        private changeDetector: ChangeDetectorRef) {
    }

    @Input("paForOf")
    dataSource: any;

    ngOnInit() {
        this.differ =
           <DefaultIterableDiffer<any>> this.differs.find(this.dataSource).create();
    }

    ngDoCheck() {
        let changes = this.differ.diff(this.dataSource);
        if (changes != null) {
            console.log("ngDoCheck called, changes detected");
            changes.forEachAddedItem(addition => {
                this.container.createEmbeddedView(this.template,
                     new PaIteratorContext(addition.item,
                         addition.currentIndex, changes.length));
            });
        }
    }
}

class PaIteratorContext {
    odd: boolean; even: boolean;
    first: boolean; last: boolean;

    constructor(public $implicit: any,
            public index: number, total: number ) {

        this.odd = index % 2 == 1;
        this.even = !this.odd;
        this.first = index == 0;
        this.last = index == total - 1;
    }
}
```

Corresponding changes must be made in Chapter 19, as shown in this replacement for Listing 19-18.

Listing 19-18. Updating the Change Detection Code

```
import { Directive, HostBinding, Input,
    SimpleChange, KeyValueDiffer, KeyValueDiffers,
    ChangeDetectorRef } from "@angular/core";
import { DiscountService } from "./discount.service";

@Directive({
    selector: "td[pa-price]",
    exportAs: "discount"
})
export class PaDiscountAmountDirective {
    private differ: KeyValueDiffer<any, any>;

    constructor(private keyValueDiffers: KeyValueDiffers,
        private changeDetector: ChangeDetectorRef,
        private discount: DiscountService) { }

    @Input("pa-price")
    originalPrice: number;

    discountAmount: number;

    ngOnInit() {
        this.differ =
            this.keyValueDiffers.find(this.discount).create(this.changeDetector);
    }

    ngOnChanges(changes: { [property: string]: SimpleChange }) {
        if (changes["originalPrice"] != null) {
            this.updateValue();
        }
    }

    ngDoCheck() {
        if (this.differ.diff(this.discount) != null) {
            this.updateValue();
        }
    }

    private updateValue() {
        this.discountAmount = this.originalPrice
            - this.discount.applyDiscount(this.originalPrice);
    }
}
```

Changes for Chapter 21

A change in the way that angular-cli resolves paths means that you should omit the `moduleId` property from the `@Component` decorator.

You can ignore the instructions to delete JavaScript files in this chapter because angular-cli configures the compiler to work in-memory, so that it doesn't generate any files.

Changes for Chapter 24

The error handling for HTTP requests has changed in Angular 4, as shown in this replacement for Listing 24-7.

Listing 24-7. Updating the HTTP Error Handling Code

```
import { Injectable, Inject, OpaqueToken } from "@angular/core";
import { Http, Request, RequestMethod, Headers, Response } from "@angular/http";
import { Observable } from "rxjs/Observable";
import { Product } from "./product.model";
import "rxjs/add/operator/map";
import "rxjs/add/operator/catch";
import "rxjs/add/observable/throw";
import { Subject } from "rxjs/Subject";

export const REST_URL = new OpaqueToken("rest_url");

@Injectable()
export class RestDataSource {

    constructor(private http: Http,
        @Inject(REST_URL) private url: string) { }

    getData(): Observable<Product[]> {
        return this.sendRequest(RequestMethod.Get, this.url);
    }

    saveProduct(product: Product): Observable<Product> {
        return this.sendRequest(RequestMethod.Post, this.url, product);
    }

    updateProduct(product: Product): Observable<Product> {
        return this.sendRequest(RequestMethod.Put,
            `${this.url}/${product.id}`, product);
    }

    deleteProduct(id: number): Observable<Product> {
```

```
        return this.sendRequest(RequestMethod.Delete, `${this.url}/${id}`);
    }

    private sendRequest(verb: RequestMethod,
        url: string, body?: Product): Observable<Product> {

        let headers = new Headers();
        headers.set("Access-Key", "<secret>");
        headers.set("Application-Names", ["exampleApp", "proAngular"]);

        return this.http.request(new Request({
            method: verb,
            url: url,
            body: body,
            headers: headers
        }))

        .map(function(response) {
            console.log("++++++++++++ " + response);
            return response.json();
        })
        .catch(function(error) {
            return
              Promise.reject(`Network Error: ${error.statusText} (${error.status})`);
        })
    }
}
```

Changes for Chapter 25

You can omit the `moduleId` property from the `@Component` decorator.

Changes Chapter 26

The change detection API has changed, as shown in this replacement Listing 26-4.

Listing 26-4. Updating the Change Detection Code

```
import { Component, KeyValueDiffer,
         KeyValueDiffers, ChangeDetectorRef } from "@angular/core";
import { Model } from "../model/repository.model";

@Component({
    selector: "paProductCount",
    template: `<div class="bg-info p-a-1">There are {{count}} products</div>`
})
```

```
export class ProductCountComponent {
    private differ: KeyValueDiffer<any, any>;
    count: number = 0;

    constructor(private model: Model,
        private keyValueDiffers: KeyValueDiffers,
        private changeDetector: ChangeDetectorRef) {}

    ngOnInit() {
        this.differ = this.keyValueDiffers
            .find(this.model.getProducts())
            .create(this.changeDetector);
    }

    ngDoCheck() {
        if (this.differ.diff(this.model.getProducts()) != null) {
            this.updateCount();
        }
    }

    private updateCount() {
        this.count = this.model.getProducts().length;
    }
}
```

The way that child routes are defined has changed, which means they can no longer be specified without at least one action property. This replacement Listing 26-17 shows the change that is required to the example application's routing configuration.

Listing 26-17. Changing the Way that Child Routes Are Defined

```
import { Routes, RouterModule } from "@angular/router";
import { TableComponent } from "./core/table.component";
import { FormComponent } from "./core/form.component";
import { NotFoundComponent } from "./core/notFound.component";
import { ProductCountComponent } from "./core/productCount.component";
import { CategoryCountComponent } from "./core/categoryCount.component";


const routes: Routes = [
    { path: "form/:mode/:id", component: FormComponent },
    { path: "form/:mode", component: FormComponent },
    { path: "does", redirectTo: "/form/create", pathMatch: "prefix" },

    {
        path: "table",
        component: TableComponent,
        children: [
            { path: "products", component: ProductCountComponent },
```

```
            { path: "categories", component: CategoryCountComponent }
        ]
    },


    { path: "table/:category", component: TableComponent },
    { path: "table", component: TableComponent },
    { path: "", redirectTo: "/table", pathMatch: "full" },
    { path: "**", component: NotFoundComponent }
]

export const routing = RouterModule.forRoot(routes);
```

Changes Chapter 28

The animation functionality has been moved into a separate module. This requires an addition to the `package.json` file, like this:

```
...
  "dependencies": {
    "@angular/animations": "^4.0.0",
    "@angular/common": "^4.0.0",
    "@angular/compiler": "^4.0.0",
    "@angular/core": "^4.0.0",
    "@angular/forms": "^4.0.0",
    "@angular/http": "^4.0.0",
    "@angular/platform-browser": "^4.0.0",
    "@angular/platform-browser-dynamic": "^4.0.0",
    "@angular/router": "^4.0.0",
    "core-js": "^2.4.1",
    "rxjs": "^5.1.0",
    "zone.js": "^0.8.4",
    "bootstrap": "4.0.0-alpha.4"
  },
...
```

To enable animations, you must import the new module in your root module, like this:

```
import { NgModule } from "@angular/core";
import { BrowserModule } from "@angular/platform-browser";
import { ModelModule } from "./model/model.module";
import { CoreModule } from "./core/core.module";
import { TableComponent } from "./core/table.component";
import { FormComponent } from "./core/form.component";
import { MessageModule } from "./messages/message.module";
import { MessageComponent } from "./messages/message.component";
import { routing } from "./app.routing";
import { AppComponent } from "./app.component";
import { TermsGuard } from "./terms.guard"
```

```
import { LoadGuard } from "./load.guard";
import { BrowserAnimationsModule } from "@angular/platform-browser/animations";

@NgModule({
    imports: [BrowserModule, CoreModule, MessageModule, routing,
BrowserAnimationsModule],
    declarations: [AppComponent],
    providers: [TermsGuard, LoadGuard],
    bootstrap: [AppComponent]
})
export class AppModule { }
```

Changes for Chapter 29

When angular-cli creates a project, it adds support for unit testing. To run the tests, just open a new command prompt, navigate to the project folder and run `ng test`.