Angular 9 Cheat Sheet

Introduction

Angular 9 is one of the most anticipated releases of Angular to date. And for good reason. This will be the first release of Angular where the Ivy renderer is enabled by default. So what can you do to prepare for the pending release? Here are some things you can do today!

Each Cheat will include an **ACTION:** that you can take today to prepare for Angular 9

Full List of Cheats

- Template Class Binding Fixes
- ModuleWithProviders
- TestBed.inject
- Removed APIs including ngForm and Renderer
- Template Only Variable Errors
- @Component and @Directive host Binding Inheritance
- @ContentChild and @ContentChildren Query Matching
- TypeScript Compiler Options

Template Class Binding Fixes

Inline class binding and [class] bindings now work together. If you have code that is using both of these approaches, you will see a change in behavior when you migrate to Angular 9. Currently, only the last binding in the template is rendered. Consider the following example.

```
<p [class]="'bold'" class="{{'red'}}">
  Start editing to see some magic happen :)
</p>
```

Open in StackBlitz

Currently, the rendered html only contains the “red” class and NOT the bold class. With the Ivy renderer in Angular 9, the rendered html will be BOTH bold and red.

ACTION: You could search for instances of [class] in your templates to make sure that you’re not going to run into any unexpected rendering issues once you migrate.

ModuleWithProviders

ModuleWithProviders now requires that you provide a generic type. Previously, the generic was optional.

```
// Before

@NgModule({...})
export class MyModule {
  static forRoot(config: SomeConfig): ModuleWithProviders {
    return {
      ngModule: SomeModule,
      providers: [
        {provide: SomeConfig, useValue: config}
      ]
    };
  }
}
```

```
// After

@NgModule({...})
export class MyModule {
  static forRoot(config: SomeConfig): ModuleWithProviders<SomeModule> {
    return {
      ngModule: SomeModule,
      providers: [
        {provide: SomeConfig, useValue: config }
      ]
    };
  }
}
```

If your application uses the CLI, running `ng update` will automatically take care of this for you. Otherwise, you'll need to make these updates manually.

ACTION: Add generics today to ModuleWithProviders usages in your applications

TestBed.inject

TestBed.get now changes to TestBed.inject, which requires a generic type. Currently, the generic in TestBed.get is optional.

```
inject<T>(token: Type<T> | InjectionToken<T> | AbstractType<T>, notFoundValue: null,
flags?: InjectFlags): T | null
```

ACTION: You could simplify your migration by adding the generic type information to your current usages of TestBed.get

ngForm replaced with ng-form

These selectors can be globally replaced very easily, but this is mentioned specifically because the `ng update` command will NOT automatically make this change for you.

```
// Before
<ngForm #myForm="ngForm">
// After
<ng-form #myForm="ngForm">
```

ACTION: Change all instances of ngForm to ng-form in your applications

Renderer replaced with Renderer2

The schematic within `ng update` will automatically update any instances of Renderer in your application, with the exception of Renderer.animate() and Renderer.setDebugInfo(), which already aren't supported.

If you use Renderer extensively in your application, you should read the migration guide in the Angular Docs at https://next.angular.io/guide/migration-renderer

ACTION: Read the Angular Docs on Renderer2 migration

Additionally Removed APIs

If you use any of the following APIs in your application, you will need to manually replace them.

• @angular/core RootRenderer	Replaced with RendererFactory2
• @angular/core RenderComponentType	Replaced with RendererType2
• @angular/core WtfScopeFn	No Replacement
• @angular/core wtfCreateScope	No Replacement
• @angular/core wtfStartTimeRange	No Replacement
• @angular/core wtfEndTimeRange	No Replacement
• @angular/core wtfLeave	No Replacement
• @angular/common DeprecatedI18NPipesModule	Replaced with CommonModule
• @angular/common DeprecatedCurrencyPipe	Replaced with CurrencyPipe
• @angular/common DeprecatedDatePipe	Replaced with DatePipe
• @angular/common DeprecatedDecimalPipe	Replaced with DecimalPipe
• @angular/common DeprecatedPercentPipe	Replaced with PercentPipe
• @angular/forms NgFormSelectorWarning	No Replacement
• @angular/service-worker versionedFiles	files

More information can be found at https://next.angular.io/guide/updating-to-version-9

ACTION: Replace any deprecated APIs in your applications with their Angular 9 replacements

Template Only Variable Errors

In Angular 9, it will be an error to assign values to template-only variables like item in ngFor="let item of items" (previously, the compiler would ignore these assignments).

Additionally, dynamically assigning values to template only variables is no longer supported. Consider the following example where we are using 'show' on #pToggle to toggle visibility of an element in the template.

```
<p *ngIf="pToggle.show">
  Start editing to see some magic happen :)
</p>
<button #pToggle (click)="pToggle.show = true">Show P Tag</button>
{{pToggle.show}}
```

ACTION: Remove template only variables and create variables in your TypeScript class that can be referenced in the template.

@Component and @Directive host Binding Inheritance

Properties like host inside @Component and @Directive decorators can be inherited (previously, only properties with explicit field decorators like @HostBinding would be inherited).

```
@Component({
  selector: 'myComponent',
  template: '<p></p>',
  host: {
    [key: string]: string;
    }
})
```

In Angular 8, if you extended this class, the new class would not inherit the values specified by host. In Angular 9, the host values will be inherited.

ACTION: Look for instances of host values in your components and directives that are being extended in your applications and analyze the code for side effects when these values are inherited in Angular 9

@ContentChild and @ContentChildren Query Matching

@ContentChild and @ContentChildren queries will no longer be able to match their directive's own host node (previously, these queries would match the host node in addition to its content children).

ACTION: Be sure to review your usage of these queries to make sure that you are not currently matching the host node to avoid bugs when migrating.

New Optional TS Compiler Options

A new "strict" mode has been added when creating an application with the CLI using the –strict flag in Angular 9. These options are extremely powerful in maintaining safety in your application, but existing applications are almost certain to have problems if you enable these compiler options all at once.

For existing applications, turn these flags on one at a time and then work through the compilation errors.

- "noImplicitAny"
  - Read more at https://basarat.gitbooks.io/typescript/docs/options/noImplicitAny.html
- "noImplicitReturns"
- "noImplicitThis"
  - Read more at https://www.logicbig.com/tutorials/misc/typescript/no-implicit-this.html
- "noFallthroughCasesInSwitch"
- "strictNullChecks"
  - Read more at https://www.typescriptlang.org/docs/handbook/release-notes/typescript-2-0.html
  - Note: This may cause problems with third party libraries in your code that are not compatible

Additionally, TypeScript 3.6 is now the default. So you should get familiar with the new capabilities. You can read about them at https://devblogs.microsoft.com/typescript/announcing-typescript-3-6/

ACTION: For new applications, always use the –strict options from the beginning to improve the safety of the component API surfaces and to reduce the testing requirements of your application.

ACTION: For existing applications, begin turning on these compiler options one at a time. For large applications, consider creating a folder specific tsconfig file to iteratively enable these options.