Angular 2/4 Component Styles :host, :host-context, /deep/ Selector Example

By Arvind Rai, June 24, 2017

This page will walk through Angular component styles :host, :host-context, /deep/ selector example. Component styles can use few special selectors such as :host, :host-context and /deep/ that works using shadow DOM scoping. Shadow DOM makes things separate from DOM of main document. Shadow DOM provides encapsulation for DOM and CSS. Shadow DOM can also be used itself outside of the web component. Shadow DOM makes CSS management easy and maintainable in the big applications. In Angular :host, :host-context, and /deep/ selectors are used in components that are in parent child relationship. The `:host` selector in Angular component plays the role to apply styles on host element. By default component style works within its component template. But by using `:host` selector we can apply styles to host element that resides in parent component. `:host-context` selector works same as `:host` selector but based on a given condition. `/deep/` selector forces styles in its own components and in all child components. On this page we will discuss complete example of :host, :host-context, /deep/ selector step by step.

Contents

Technologies Used

Find the technologies being used in our example.
1. Angular 4.0.0
2. TypeScript 2.3.3
3. Node.js 6.10.1
4. Angular CLI 1.1.2
5. Angular Compiler CLI 4.0.0

Project Structure

Find the project structure of our demo application.

```
angular-demo
|
|--src
|   |
|   |--app
|   |   |
|   |   |--address.component.css
|   |   |--address.component.ts
|   |   |--company.component.css
|    |    |--company.component.ts
|    |    |--person.component.css
|    |    |--person.component.ts
|    |    |--app.component.css
|    |    |--app.component.ts
|    |    |--app-routing.module.ts
|    |    |--app.module.ts
|    |
|    |--main.ts
|    |--index.html
|    |--styles.css
|
|--node_modules
|--package.json
```

Component Styles

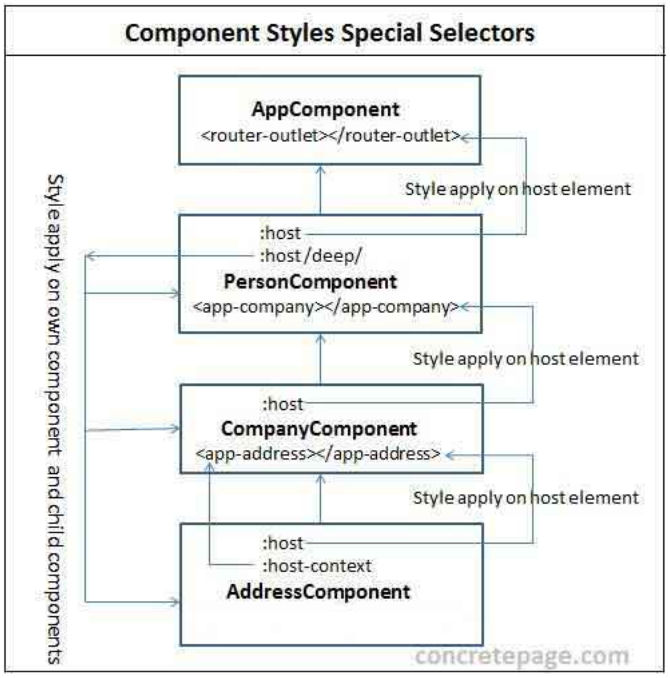
Every component can have its own HTML file as well as CSS file. Component can also use inline HTML and CSS code. Find the use of some `@Component` metadata.
template: Write inline HTML code.
templateUrl: Configure HTML file.
styles: Write inline CSS.
styleUrls: Configure CSS file.

Find the advantage of component style over traditional CSS.
1. Class names and selectors are local to component and don't affect other part of application.
2. Changing style to other part of application does not affect the component style.
3. We can create CSS file component specific and can co-locate with them.
4. Removing and updating component CSS is easy because we need not to worry about whether else the CSS is being used in the application.

Component styles have few special selectors that has been taken from shadow DOM style scoping. We will discuss here :host, :host-context, /deep/ selector with examples.

Diagram for :host, :host-context, /deep/ Selector

Let us understand the working of :host, :host-context, /deep/ selector using diagram.


Component Styles Special Selectors
AppComponent
<router-outlet></router-outlet>
Style apply on host element
:host
:host /deep/
PersonComponent
<app-company></app-company>
Style apply on host element
:host
CompanyComponent
<app-address></app-address>
Style apply on host element
:host
:host-context
AddressComponent
Style apply on own component and child components
concretepage.com

In the above diagram we have few components in parent-child relationship. Find the explanation of pseudo selectors.
:host : Style is written in child component but it applies on host element in parent component. Look at the

Popular Post

Angular 2 NgClass Example

Angular 2 Custom Event Binding - EventEmitter Example

Spring REST Client with RestTemp Consume RESTful Web Service Ex for XML and JSON

Angular 2 NgFor Example

Angular 2 @Input and @Output Example

Find us on Facebook

CP Concretepage.com
1,014 likes

Liked Use

You like this

Featured Post

Angular 2 Decimal Pipe, Percent P and Currency Pipe Example

Angular 2 Date Pipe Example

Angular 2 Http get() Example

Spring 4 + SOAP Web Service Pro and Consumer Example with Tom

Java 8 Stream sorted() Example

diagram. The style written in `:host` selector in `PersonComponent` will be applied in `<router-outlet>` element of `AppComponent`. The style written in `:host` selector in `CompanyComponent` will be applied in `<app-company>` host element in `PersonComponent` and so on.

:host-context : It applies style on host element based on some condition outside of a component view. Condition can be like if the given CSS class is available anywhere in parent tree, only when the style written in `:host-context` will be applied to host element in parent component.

/deep/: The style written in `/deep/` selector will be applied on its own component template and down through the child component tree into all the child component views. In the diagram `PersonComponent` is using `/deep/` selector. The style written in `/deep/` selector will be applied in `PersonComponent`, `CompanyComponent` and `AddressComponent` templates.

:host

`:host` is a pseudo-class selector that applies styles in the element that hosts the component. It means if a component has a child component using component binding then child component will use `:host` selector that will target host element in parent component. `:host` selector can be used in component with `styles` metadata as well as with `styleUrls` metadata of `@Component` decorator.

1. Find the example of `:host` selector with `styles`

```
@Component({
  ---
  styles: [ ':host { position: absolute; top: 10%; }' ]
})
```

2. Find the example of `:host` selector with `styleUrls`. If we use CSS file as follows.

address.component.css

```
:host {
   position: absolute;
   top: 10%;
}
```

Then it is configured as follows.

```
@Component({
  ---
  styleUrls: ['./address.component.css']
})
```

Now suppose we have a parent component as given below.

person.component.ts

```
@Component({
  template: `
      <app-company></app-company>
  `
})
export class PersonComponent {
}
```

Find the child component

company.component.ts

```
@Component({
  selector: 'app-company',
  template: `
      <h3>Company</h3>
  `,
  styleUrls: ['./company.component.css']
})
export class CompanyComponent {
}
```

company.component.css

```
:host  {
    position: absolute;
    margin: 5% 5%;
    border: 5px solid blue;
    width: 250px;
    height: 250px;
    background-color: grey;
}
```

The style written in `:host()` selector will be applied to `<app-company>` element.

:host-context

`:host-context` selector is used in the same way as `:host` selector but `:host-context` is used when we want to apply a style on host element on some condition outside of the component view. For the example a style could be applied on host element only if a given CSS class is found anywhere in parent tree up to the document root. In our example we have following components in parent-child relationship.

AppComponent -> PersonComponent -> CompanyComponent -> AddressComponent

Suppose we have a CSS class in `AppComponent` as follows.
app.component.css

```
.my-theme {
   background-color: blue;
}
```

Now in the last component of the child tree i.e. `AddressComponent`, we will use `:host-context` selector as follows.
address.component.css

```
:host-context(.my-theme) h3 {
    background-color: green;
    font-style: normal;
}
```

The style given above will be applied on host element only when a CSS class named as `my-theme` will be found anywhere in parent tree up to the document root.

/deep/

`/deep/` selector has alias as `>>>` . Component style normally applies only to the component's own template. Using `/deep/` selector we can force a style down through the child component tree into all child component views. `/deep/` selector forces its style to its own component, child component, nested component, view children and content children. Suppose we have components with parent child relationship as follows.

PersonComponent -> CompanyComponent -> AddressComponent

In `PersonComponent` we are using following CSS file with `/deep/` selector.
person.component.css

```
:host /deep/ h3 {
  color: yellow;
  font-style: italic;
}
:host  >>> p {
  color: white;
  font-style: Monospace;
  font-size: 20px;
}
```

The above style will be forced on templates of `PersonComponent`, `CompanyComponent` and `AddressComponent`.

View Encapsulation

Component CSS are encapsulated into the component's view and don't affect the rest of the application. We can change this behavior using `ViewEncapsulation` enum. There are three types of encapsulation.

1. Native

`Native` view encapsulation uses browser's native encapsulation mechanism.

2. Emulated

`Emulated` is the **default** view encapsulation in our angular application.

3. None

`None` means no view encapsulation.

In our component we can configure view encapsulation as follows. Suppose we want to enable `Native` encapsulation.

my-component.ts

```
import { Component, ViewEncapsulation } from '@angular/core';

@Component({
  ---
  encapsulation: ViewEncapsulation.Native
})
export class MyComponent { }
```

Find the reference link.

Complete Example

app.component.ts

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  template: `
    <div [ngClass]="'my-theme'">
      <a routerLink="/person" routerLinkActive="active">Home</a>
      <router-outlet></router-outlet>
    </div>
  `,
  styleUrls: ['./app.component.css']
})
export class AppComponent {
}
```

app.component.css

```
.my-theme {
    background-color: blue;
}
.my-theme .active {
    color: white;
}
```

person.component.ts

```
import { Component } from '@angular/core';

@Component({
  template: `
          <h3>Person</h3>
          <p>Welcome to Person Home</p>
          <app-company></app-company>
  `,
  styleUrls: ['./person.component.css']
})
export class PersonComponent {
}
```

person.component.css

```
:host /deep/ h3 {
    color: yellow;
    font-style: italic;
}
:host  >>> p {
    color: white;
    font-style: Monospace;
    font-size: 20px;
}

:host  {
    position: absolute;
    top: 10%;
    border: 5px solid red;
    background-color: silver;
    width: 300px;
    height: 400px;
}
```

company.component.ts

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-company',
  template: `
          <h3>Company</h3>
          <p>Welcome to Company Home</p>
          <app-address></app-address>
  `,
  styleUrls: ['./company.component.css']
})
export class CompanyComponent {
}
```

company.component.css

```
:host  {
        position: absolute;
        margin: 5% 5%;
        border: 5px solid blue;
        width: 250px;
        height: 250px;
        background-color: grey;
}
h3 {
        font-size: 15px;
        background-color: black;
}
```

address.component.ts

```
import { Component, ViewChild } from '@angular/core';

@Component({
  selector: 'app-address',
  template: `
          <h3>Address</h3>
          <p>Welcome to Address Home</p>
  `,
  styleUrls: ['./address.component.css']
})
export class AddressComponent {
}
```

address.component.css

```
:host  {
        position: absolute;
        margin: 5% 5%;
        border: 5px solid blue;
        width: 200px;
        height: 120px;
        background-color: blue;
}
:host-context(.my-theme) h3 {
        background-color: green;
        font-style: normal;
}
:host-context(.my-theme) p {
        color: red;
        font-size: 18px;
}
```

app-routing.module.ts

```
import { NgModule }      from '@angular/core';
import { RouterModule, Routes } from '@angular/router';

import { PersonComponent }  from './person.component';

const routes: Routes = [
        {
           path: 'person',
           component: PersonComponent
        },
        {
           path: '',
           redirectTo: '/person',
           pathMatch: 'full'
        }
];
@NgModule({
  imports: [
          RouterModule.forRoot(routes)
  ],
  exports: [
          RouterModule
  ]
})
export class AppRoutingModule{ }
```

app.module.ts

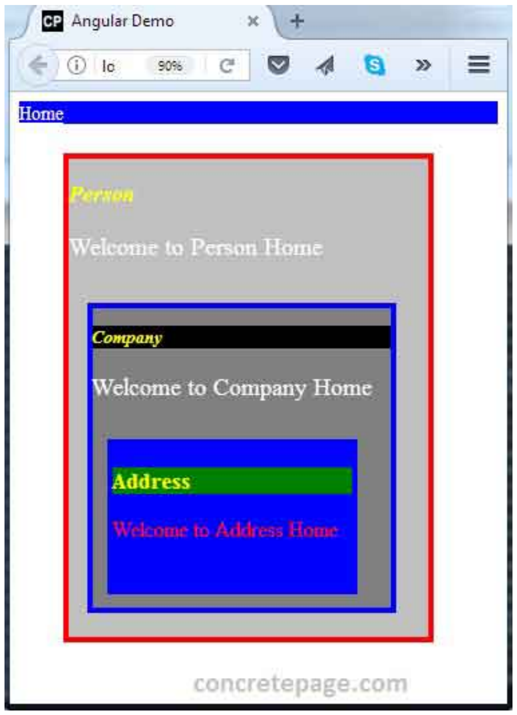
```
import { NgModule }   from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { AppComponent }  from './app.component';
import { PersonComponent }  from './person.component';
import { CompanyComponent }  from './company.component';
import { AddressComponent }  from './address.component';
import { AppRoutingModule }  from './app-routing.module';

@NgModule({
  imports: [
        BrowserModule,
        AppRoutingModule
  ],
  declarations: [
        AppComponent,
        PersonComponent,
        CompanyComponent,
        AddressComponent
  ],
  providers: [ ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
```

Run Application

To run the application, find following steps.
1. Download source code using download link given on this page.
2. In your angular CLI application, replace **src** folder from the downloaded one.
3. Run **ng serve** using command prompt.
4. Now access the URL **http://localhost:4200**
Find the print screen of the output.



References

Component Styles
Shadow DOM

Download Source Code

angular-2-4-component-styles-host-host-context-deep-selector-example.zip

Share Tweet G+

POSTED BY

ARVIND RAI

Popular Tutorials: Java 8 | Spring 4 | Struts 2 | Hibernate 3 | Android

FIND MORE TUTORILAS

HIBERNATE 4 PRIMEFACES 5 MYBATIS 3 FREEMARKER QUARTZ 2

Login

Leave your comment...

Comments Sort by Newest

Be the first to comment!

ADD WIDGETPACK TO YOUR SITE POWERED BY **WIDGET PACK™**

Favorite Links

Java Technology
Hibernate Annotations
Spring Framework
JQuery
Apache Struts 2
MyBatis
Quartz Scheduler
Angular
Thymeleaf
FreeMarker

About Us

We are a group of software developers.
We enjoy learning and sharing technologies.
To improve the site's content,
your valuable suggestions
are most welcome. *Thanks*
Email : **concretepage@gmail.com**

Mobile Apps

ConcretePage.com

Get it on Google play

SCJP Quiz

Get it on Google play