Clean up your DOM by using the full power of Angular Host elements

herodevs

What is the host element?

```
export function Button() {
  return <button>Click</button>
}
```

```jsx
export function Button() {
  return <button>Click</button>
}

// Renders:
<button>Click</button>
```

```
@Component({
  selector: 'app-button',
  standalone: true,
  template: '<button>Click</button>'
})
export class AppButtonComponent {}
```

```
@Component({
  selector: 'app-button',
  standalone: true,
  template: '<button>Click</button>'
})
export class AppButtonComponent {}

// Renders:
<app-button><button>Click</button></app-button>
```

```
<app-button></app-button>
```

```
<div>
  <div>
    <div>
      <div>
        <div>
          <div>
            <div>
              <div>
                <div>
                  <div>
                    <div>
                      <div>
                        <div>
                          <!-- to be continued... -->
```

```
<app-root>
  <div>
    <div>
      <app-page>
        <div>
          <div>
            <app-grid>
              <div>
                <div>
                  <app-row>
                    <div>
                      <div>
                        <app-cell>
                          <!-- to be continued... -->
```

```
<app-root>
  <app-page>
    <app-grid>
      <app-row>
        <app-cell>
```

About me

- Rafael Mestre
- Senior Software Engineer @ HeroDevs
- https://mestre.dev
- @mestre_dev
- /in/rlmestre

herodevs

Backstory

```
<div class="row">
  <span class="cell">
    <button>{{ condition ? 'Create' : 'Update' }}</button>
  </span>
</div>
```

```
<div ng-if="condition === true">
  <div class="row">
    <span class="cell">
      <button>Create</button>
    </span>
  </div>
</div>
<div ng-if="condition === false">
  <div class="row">
    <span class="cell">
      <button>Update</button>
    </span>
  </div>
</div>
```

```
<div class=“sticky”>
  <div class=“nav”>
    <div class="left-side">
      <a>Home</a>
      <a>Feed</a>
    </div>

    <div class="right-side">
      <a>Friends</a>
      <a>Profile</a>
    </div>
  </div>
</div>
```

```
<nav class=“super-special-grid”>
  <a>Home</a>
  <a>Feed</a>
  <span class=“spacer"></span>
  <a>Friends</a>
  <a>Profile</a>
</nav>
```

Problems with an excessive DOM

- Impacts performance negatively

Impacts performance negatively

Chrome for Developers
https://developer.chrome.com › docs › performance · Block this site

Avoid an excessive DOM size - Chrome for Developers

Oct 4, 2019 — A large **DOM** tree in combination with complicated style rules can severely slow down rendering. Memory **performance**. If your JavaScript uses ...

How the Lighthouse DOM size... · How to optimize the DOM size

jim-nielsen.com
https://blog.jim-nielsen.com › thoughts-on-avoiding-a... · Block this site

Thoughts on Avoiding an Excessive DOM Size

Oct 19, 2021 — I had never seen this before. I knew a large **DOM** could create **performance** problems, but I'd only encountered them when trying to script against ...

web.dev
https://web.dev › articles › dom-size-and-interactivity · Block this site

How large DOM sizes affect interactivity, and what you can do ...

May 9, 2023 — Large DOMs affect page **performance** in a few ways: ... A screenshot of a long task caused by **excessive** rendering work in the **performance** panel of A ...

How do large DOMs affect... · How do I measure DOM size?

Problems with an excessive DOM

- Impacts performance negatively

Problems with an excessive DOM

- Impacts performance negatively
- Affects style inheritance and percentage-based calculations

Affects style inheritance and percentage-based calculations

```
* > * {
  height: 100%;
}
```

Problems with an excessive DOM

- Impacts performance negatively
- Affects style inheritance and percentage-based calculations

Problems with an excessive DOM

- Impacts performance negatively
- Affects style inheritance and percentage-based calculations
- Complicates parent-child layouts

Complicates parent-child layouts

```
<div class="flex flex-row”>
  <div class=“intruder”>

    <div class="flex-1"></div>
    <div class="flex-1"></div>
    <div class="flex-1"></div>
  </div>
</div>
```

Complicates parent-child layouts

```
<div class="flex flex-row”>
  <div class=“intruder”>
    <!-- Flex relationship is broken! ->
    <div class="flex-1"></div>
    <div class="flex-1"></div>
    <div class="flex-1"></div>
  </div>
</div>
```

Problems with an excessive DOM

- Impacts performance negatively
- Affects style inheritance and percentage-based calculations
- Complicates parent-child layouts

Problems with an excessive DOM

- Impacts performance negatively
- Affects style inheritance and percentage-based calculations
- Complicates parent-child layouts
- Harms developer experience

Harms developer experience

- “Which of these 20 elements is overflowing?”
- “Why does this higher z-index not overlap this lower one?”
- “Where do I add this `flex: 1` for my element to fill up the space?”
- “Why is this `position: absolute` causing my element to go off-screen?”

Problems with an excessive DOM

- Impacts performance negatively
- Affects style inheritance and percentage-based calculations
- Complicates parent-child layouts
- Harms developer experience

```
<app-button></app-button>
```

Attribute bindings

```
@Component({
  template: `
    <!-- This <div> only exists to apply the top level class -->
    <div [class.some-class]="prop">
      <!-- intended component contents -->
    </div>
  `,
})
export class MyComponent {
  prop: boolean;
}
```

```
@Component({
  template: `
    <!-- The wrapper isn't needed anymore -->
    <!-- Intended component contents -->
  `,
  host: {
    '[class.some-class]': 'prop',
  },
})
export class MyComponent {
  prop: boolean;
}
```

```
@Component({
  host: {
    /* modifiers like [class.className] or [style.width.px] */
    '[class.some-class]': 'prop',




  },
})
```

```
@Component({
  host: {
    /* modifiers like [class.className] or [style.width.px] */
    '[class.some-class]': 'prop',
    /* attributes for testing */
    '[attr.data-test-id]': '...',




  },
})
```

```
@Component({
  host: {
    /* modifiers like [class.className] or [style.width.px] */
    '[class.some-class]': 'prop',
    /* attributes for testing */
    '[attr.data-test-id]': '...',
    /* accessibility attributes */
    '[attr.aria-label]': '...',
    '[attr.aria-hidden]': '...',



  },
})
```

```
@Component({
  host: {
    /* modifiers like [class.className] or [style.width.px] */
    '[class.some-class]': 'prop',
    /* attributes for testing */
    '[attr.data-test-id]': '...',
    /* accessibility attributes */
    '[attr.aria-label]': '...',
    '[attr.aria-hidden]': '...',
    /* CSS properties */
    '[style.--primary-color]': '...',
  },
})
```

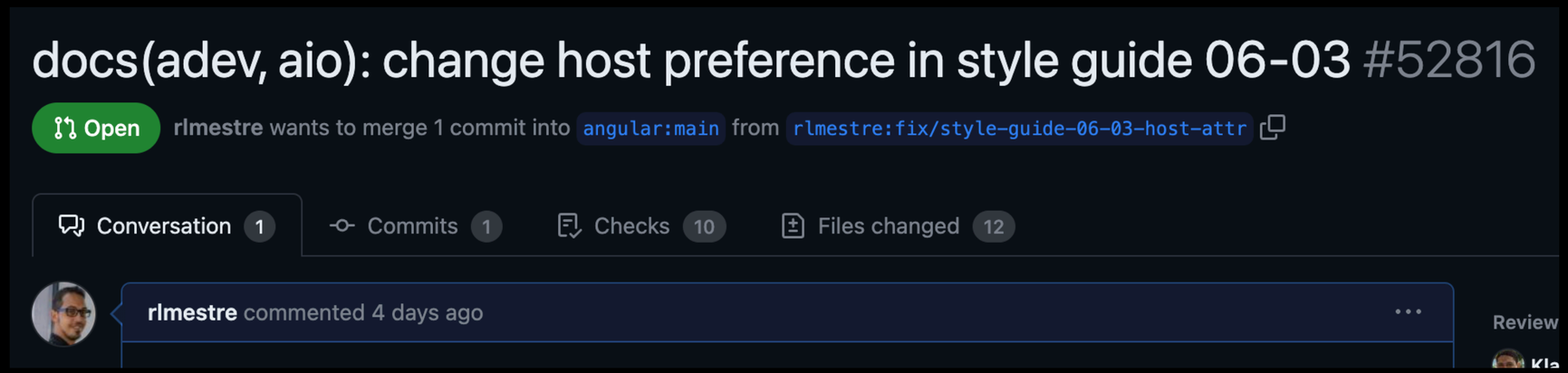
```
@Component({
  host: {




  },
})
```

```
@Component({
  host: {
    /* Ideal for Tailwind! Or any other class */
    'class': 'flex-col p-4',




  },
})
```

```
@Component({
  host: {
    /* Ideal for Tailwind! Or any other class */
    'class': 'flex-col p-4',
    /* More accessibility attributes */
    'aria-label': 'my component',
    'tabindex': '0',


  },
})
```

```
@Component({
  host: {
    /* Ideal for Tailwind! Or any other class */
    'class': 'flex-col p-4',
    /* More accessibility attributes */
    'aria-label': 'my component',
    'tabindex': '0',
    /* Roles for semantic HTML */
    'role': 'button',
  },
})
```

```
@Component({
  /* … */
})
export class MyComponent {
  prop: boolean;

  @HostBinding('class.some-class') get someClass() {
    return this.prop;
  }
}
```



```
@Component({
  host: {
    '[class.someClass]': ‘prop()’,
  },
})
export class MyComponent {
  prop = input(0);
}
```

Stylesheets

```
:host {



}
```

```
:host {
  /* Anything we put in here applies to our <host-element> */




}
```

```
:host {
  display: block;




}
```

```
:host {
  display: block;

  /* This acts as encapsulation */
  .mat-card { /* ... */ }



}
```

```
:host {
  display: block;

  /* This acts as encapsulation */
  .mat-card { /* ... */ }

  ::ng-deep {

  }
}
```

```
:host {
  display: block;

  /* This acts as encapsulation */
  .mat-card { /* ... */ }

  ::ng-deep .mat-form-field .mat-form-field-wrapper {
    padding: 0;
  }
}
```

```
:host {
  display: block;

  /* This acts as encapsulation */
  .mat-card { /* ... */ }

  ::ng-deep .mat-form-field .mat-form-field-wrapper {
    padding: 0;
  }
}

/* Applies if the element has an ancestor with this selector */
:host-context(.theme-dark) {
  color: white;
}
```

Event bindings

```
@Component({
  template: `
    <!-- This <div> only exists to capture the top level event ->
    <div (click)="onClick($event)">
      <!-- Intended component contents -->
    </div>
  `,
})
export class MyComponent {
  onClick(event: MouseEvent) {
    // ...
  }
}
```

```
@Component({
  template: `
    <!-- The wrapper component is not needed -->
    <!-- Intended component contents -->
  `,
  host: {
    '(click)': 'onClick($event)',
  },
})
export class MyComponent {
  onClick(event: MouseEvent) {
    // ...
  }
}
```

```
@Component({
  host: {
    '(click.prevent)': 'onClick($event)',
    '(window:click.prevent)': 'onClick($event)',
  },
})
export class MyComponent {
  onClick(event: MouseEvent) {
    // ...
  }
}
```

```
@Component({
  host: {
    '(click.prevent)': 'onClick($event)',
    '(window:click.prevent)': 'onClick($event)',
  },
})
export class MyComponent {
  @HostListener('click.prevent', ['$event'])
  @HostListener('window:click.prevent', ['$event'])
  onClick(event: MouseEvent) {
    // ...
  }
}
```

Directive bindings

```
@Component({
  selector: 'my-component',
  standalone: true,
  template: `<div cdkDrag><!-- ... --></div>`,
})
export class MyComponent {}
```

```ts
@Component({
  selector: 'my-component',
  standalone: true,
  template: `<div cdkDrag><!-- ... --></div>`,
})
export class MyComponent {}

@Component({
  standalone: true,
  template: `<my-component cdkDrag><!-- ... --></my-component>`,
  imports: [MyComponent],
})
export class ParentComponent {}
```

```ts
@Component({
  template: `<!-- ... -->`,
  hostDirectives: [
    CdkDrag,
  ],
})
export class MyComponent {

}
```

```ts
@Component({
  template: `<!-- ... -->`,
  hostDirectives: [
    CdkDrag,
    { directive: CdkDrag, inputs: ['cdkDragData'] },
  ],
})
export class MyComponent {

}
```

```ts
@Component({
  template: `<!-- ... -->`,
  hostDirectives: [
    CdkDrag,
    { directive: CdkDrag, inputs: ['cdkDragData: myData'] },
  ],
})
export class MyComponent {

}
```

```ts
@Component({
  template: `<!-- ... -->`,
  hostDirectives: [
    CdkDrag,
    { directive: CdkDrag, inputs: ['cdkDragData: myData'] },
  ],
})
export class MyComponent {
  constructor(cdkDrag: CdkDrag) {
    // Access it with DI
    cdkDrag.dragStart.subscribe(() => /* ... */);
  }
}
```

What if I don’t need a component?

```
@Directive({
  selector: ‘[myButton]',



})
export class MyDirective {


}
```

```
@Directive({
  selector: '[myButton]',
  host: {









  }
})
export class MyDirective {


}
```

```
@Directive({
  selector: '[myButton]',
  host: {
    '[class.some-class]': 'prop',
    '[attr.aria-label]': 'Button',
    '(click)': '$event',
    '(window:click)': '$event',
    'class': 'rounded px-10 etc',
    'role': 'button',
  }
})
export class MyDirective {
  @HostBinding('class.some-class') prop: boolean;
  @HostListener('click', ['$event']) onClick(e: MouseEvent) {}
}
```

```
@Directive({
  selector: '[nopaste]',
  host: {
    '(paste)': ‘$event.preventDefault()’,
  }
})
export class NoPasteDirective {
}
```

```
@Directive({
  selector: ‘button[reload]',
  host: {
    '(click)': ‘reload()’,
  }
})
export class ReloadStoreDirective {
  constructor(store: Store) {}

  reload() {
    this.store.dispatch(/* ... */);
  }
}
```

```
@Directive({
  selector: ‘ag-grid-angular‘,
  host: {
    '[class.some-class]': 'prop',
    '(click)': '$event',
    'class': 'rounded px-10 etc',
  }
})
export class AgGridAugmentedDirective {
  constructor(agGrid: AgGridAngular) {
    agGrid.suppressCellSelection = true;
  }
}
```

What if I do need a component?

```
@Component({
  selector: 'button[myButton]',
  template: '<ng-content />',
  host: {
    '[class.some-class]': 'prop',
    '(click)': '$event',
    'class': 'rounded px-10 etc',
  }
})
export class MyComponent {
  @HostBinding('class.some-class') prop: boolean;
  @HostListener('click', ['$event']) onClick(e: MouseEvent) {}
}
```

```
@Component({
  selector: 'a[href]',
  template: '<ng-content />',
  host: {
    'rel': ‘noopener',
    'target': ‘_blank',
  }
})
export class ExternalLinkComponent {
}
```

Structural directives and Control Flow

```
<div *ngIf="true">
  <div>Text</div>
  <button>Click</button>
</div>

<div *ngIf="{ a$ | async } as ctx">
  <div>Text</div>
  <button>Click</button>
</div>
```

```
<ng-container *ngIf="true">
  <div>Text</div>
  <button>Click</button>
</ng-container>

<ng-container *ngIf="{ a$ | async } as ctx">
  <div>Text</div>
  <button>Click</button>
</ng-container>
```

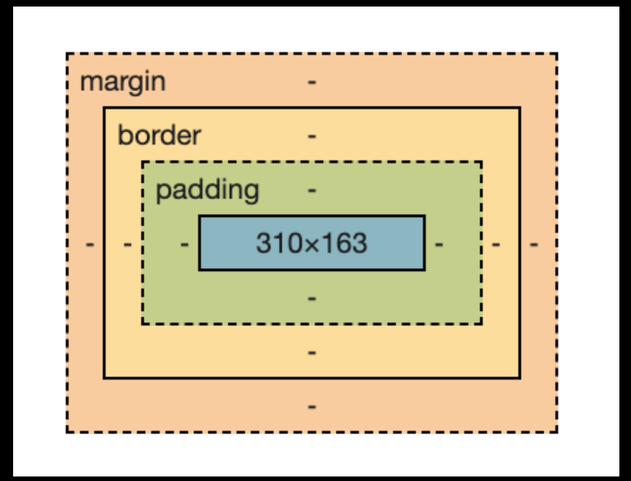
```
@if (true) {
  <div>Text</div>
  <button>Click</button>
}

@if ({ a$ | async } as ctx) {
  <div>Text</div>
  <button>Click</button>
}
```

```
@if (true) {
  <div>Text</div>
  <button>Click</button>
}

@if ({ a$ | async } as ctx) {
  <div>Text</div>
  <button>Click</button>
}

<ng-container *ngrxLet="{}">
  <button>Click</button>
</ng-container>
```

```
@if (true) {
  <div>Text</div>
  <button>Click</button>
}

@if ({ a$ | async } as ctx) {
  <div>Text</div>
  <button>Click</button>
}

<ng-container *ngrxLet="{}">
  <button>Click</button>
</ng-container>

<button *ngrxLet="{}">Click</button>
```

Last resort

```
@Component({
  selector: 'my-component',
  template: '<ng-content />',
  styles: [':host { display: contents; }'],
})
export class MyComponent {}

<div class="flex flex-row">
  <my-component> <!-- This element is ignored by the DOM! -->
    <div class="flex-1"></div>
    <div class="flex-1"></div>
    <div class="flex-1"></div>
  </my-component>
</div>
```



Wish list

```
// Ability to render a component's template directly to the DOM
@Component({
  fragment: true,
  template: `<div>Hello</div>`,
})
export class MyComponent {}

// Would render:
<div>Hello</div>
```

```
// Ability to render a component as a native element
@Component({
  as: 'button',
  template: `<div>Hello</div>`,
})
export class MyComponent {}

// Would render:
<button><div>Hello</div></button>
```

```
@Component({
  selector: ‘my-component‘,
  host: {
    '[class.some-class]': 'prop',
    '[attr.aria-label]': "'Button'",
    '(click)': '$event',
    '(window:click)': '$event',
    'class': 'rounded px-10 etc',
    'role': ‘button',
  }
})
```

```
<ng-host
  [class.some-class]="prop"
  [attr.aria-label]="'Button'"
  (click)="$event"
  (window:click)="$event"
  class="rounded px-10 etc"
  role="button"
>
  <ng-content />
</ng-host>
```

Recap

- Host bindings - classes, Tailwind, styles, attributes, ARIA, testing
- :host selector - static styles, encapsulation, ::ng-deep
- :host-context - styles based on context, ie. for themes
- Host listeners - events, including modifiers and global targets (like window)
- Host directives - API for composition and extension of directives
- Directives - logic isolation, extending existing targets
- Components with attribute selectors - directives, but with their own templates
- ng-container and control flow - structural directives, view model patterns
- `display: contents` - ignore elements in the DOM hierarchy

Thank you!