DRY in Angular Templates

Content Projection im Praxiseinsatz

Martin Grotz

Angular

- Open-source Framework
- Webtechnologien & TypeScript
- Multi-target: WebApps, MobileApps, SSR
- Komponentenorientiert
- Dependency Injection
- Automatische Change Detection

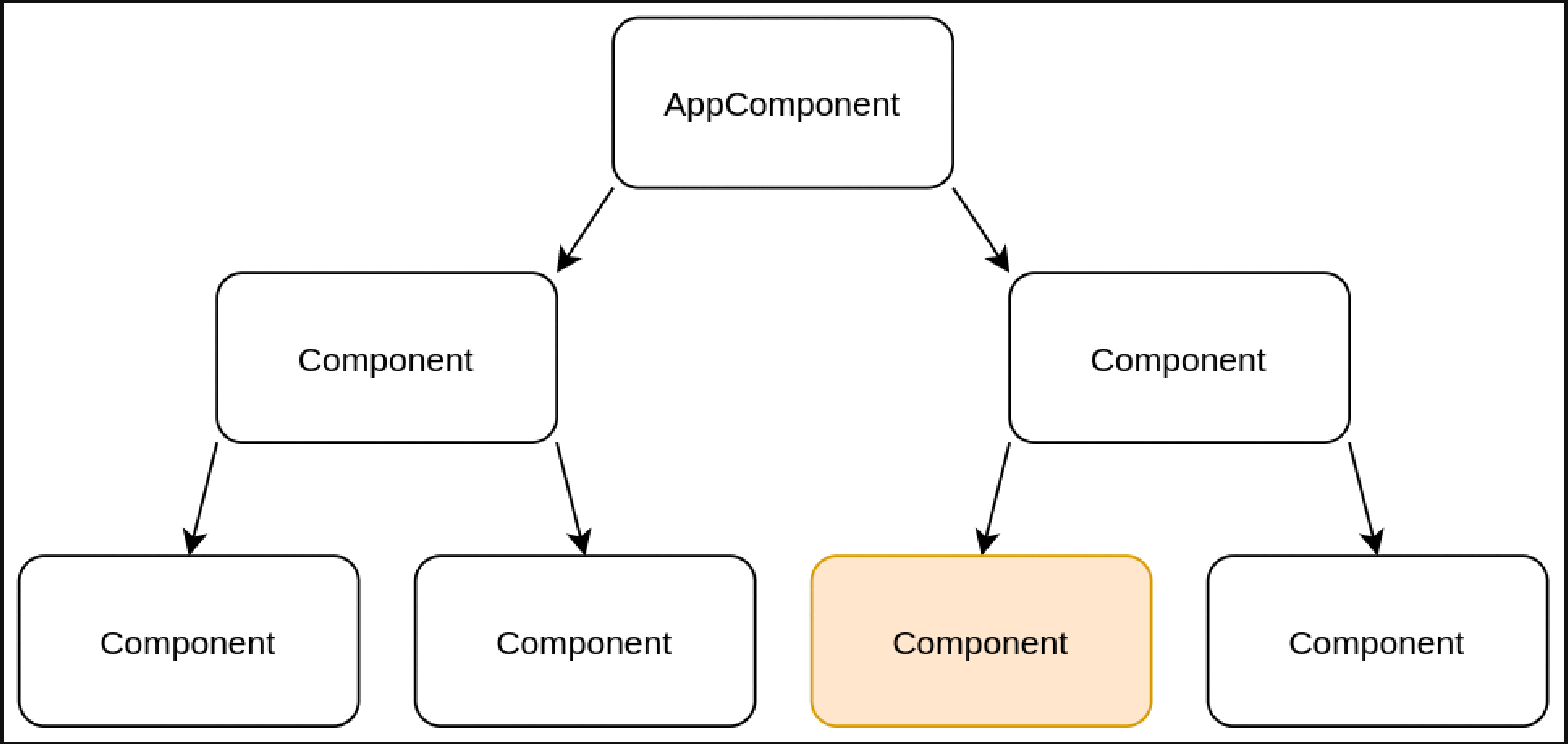
Angular Bausteine

```
@Directive({selector: '[bold]'})
export class SomeDirective {
    bold: boolean = false;
}

@Injectable()
export class SimpleService {
    public shouldBeBold = true;
}

@Component(
    {
        selector: 'app-my-component',
        template: `
        <h1>My Component</h1>
        <p>This text can be bold (or not)?</p>
        `
    }
)
export class MyComponent {}
```

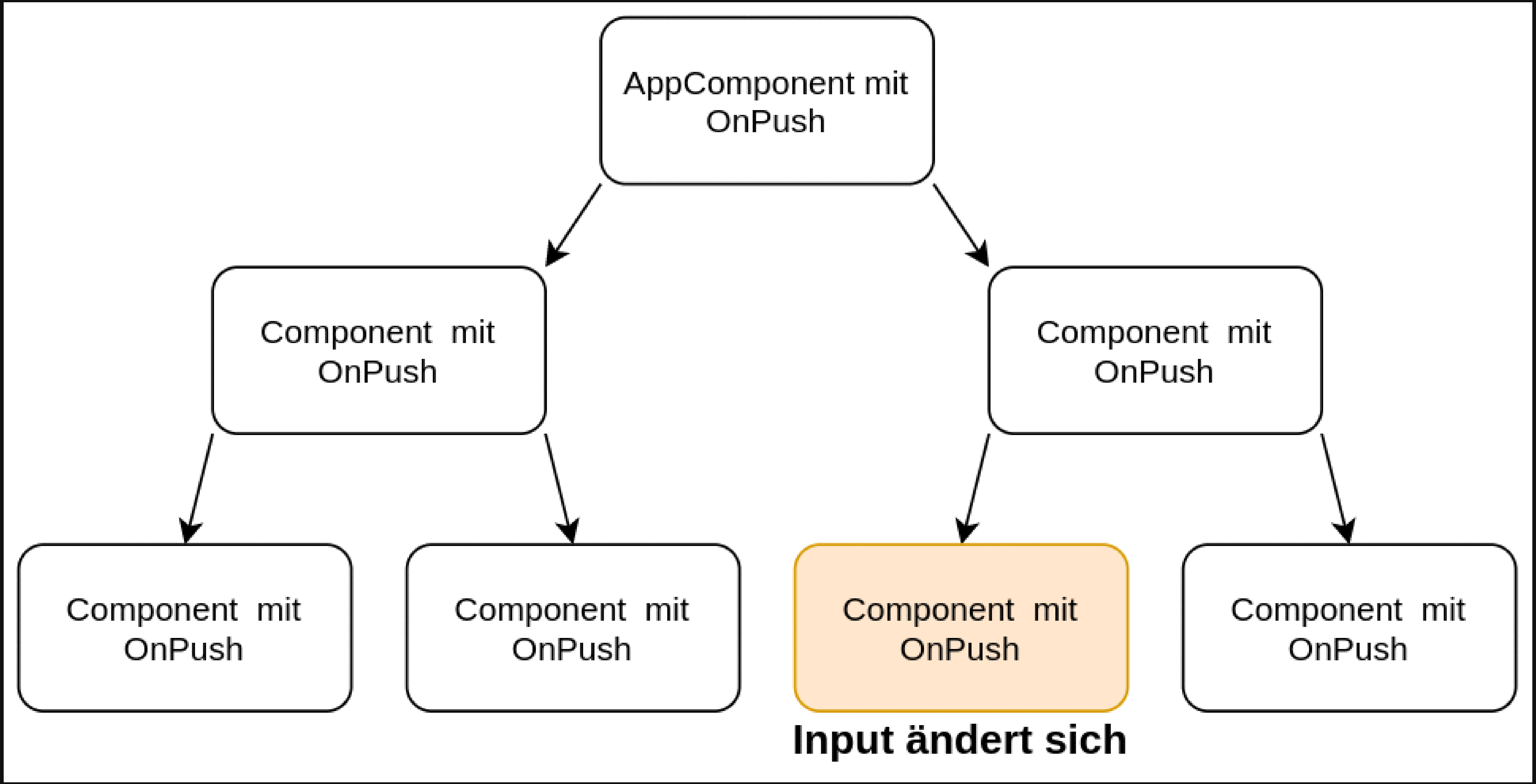
Dependency Injection & Input

```
@Directive({selector: '[bold]'})
export class SomeDirective {
    @HostBinding('class.bold')
    @Input()
    bold: boolean = false;
}

@Injectable()
export class SimpleService {
    public shouldBeBold = true;
}

@Component(
    {
        selector: 'app-my-component',
        providers: [SimpleService],
        template: `
        <style>.bold { font-weight: bold}</style>
        <h1>My Component</h1>
        <p [bold]="simpleService.shouldBeBold">This text can be bold (or not)?</p>`
    }
)
export class MyComponent {
    constructor(public readonly simpleService: SimpleService) {}
  }
```

Change Detection - Default



Change Detection - OnPush



ng-content

```
@Component(
    {
        selector: 'outer-component',
        template: `
        <h1>...</h1>
        <fancy-modal>
            <p>
                Lorem ipsum dolor sit amet.
                Duis a ornare massa.
            </p>
        </fancy-modal>
        `
    }
)
export class OuterComponent {/* ... */}
```

```
@Component(
    {
        selector: 'fancy-modal',
        template: `
        <header>...</header>
        <ng-content>

        </ng-content>
        <footer>...</footer>
        `
    }
)
export class FancyModal {/* ... */}
```

ng-content mit select

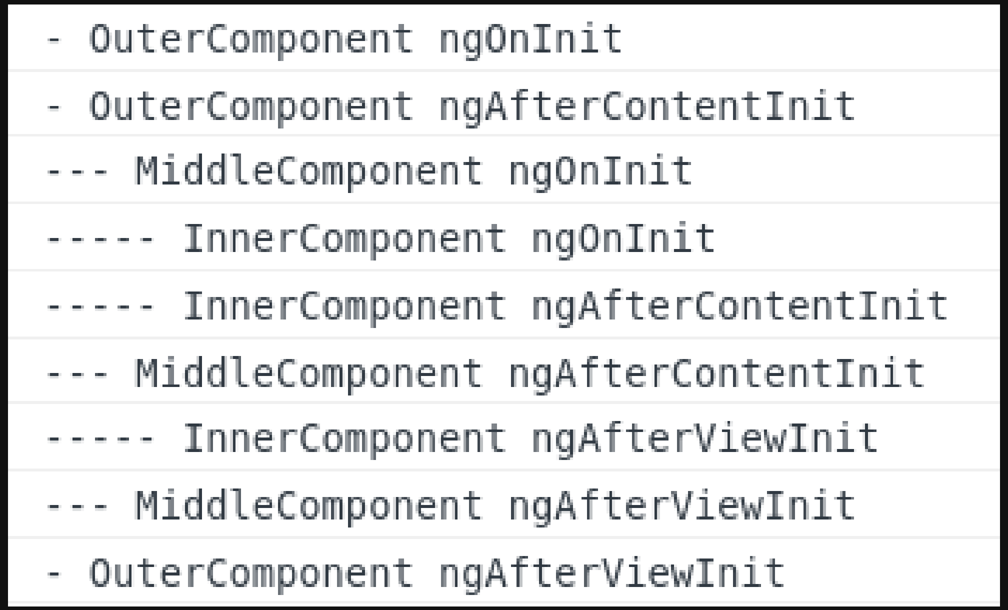
```
@Component(
    {
        /* ... */
        template: `
            <ng-content select="p"></ng-content>
            <ng-content select=".my-class"></ng-content>
            <ng-content select="[attr]"></ng-content>
            <ng-content></ng-content>
        `
    }
)
export class WithSelectors {/* ... */}
```

Angular Lifecycle Hooks

Hook	Was passiert
ngOnInit	Inputs initialisiert, Komponente initialisieren
ngAfterContentInit	Externer Content wurde projiziert und initialisiert
ngAfterViewInit	Eigenes Template und Kindkomponenten initialisiert
ngOnDestroy	Komponente wird gleich abgebaut

Lifecycle Reihenfolge

```
@Component(
    {
        selector: 'outer',
        template: `
            <middle>
                <inner></inner>
            </middle>
        `
    }
)
export class OuterComponent {/* ... */}

@Component(
    {
        selector: 'middle',
        template: `<ng-content></ng-content>`
    }
)
export class MiddleComponent {/* ... */}

@Component(
    {
        selector: 'inner',
        template: `<p>...</p>`
    }
)
export class InnerComponent {/* ... */}
```



```
- OuterComponent ngOnInit
- OuterComponent ngAfterContentInit
--- MiddleComponent ngOnInit
----- InnerComponent ngOnInit
----- InnerComponent ngAfterContentInit
--- MiddleComponent ngAfterContentInit
----- InnerComponent ngAfterViewInit
--- MiddleComponent ngAfterViewInit
- OuterComponent ngAfterViewInit
```

ContentChildren

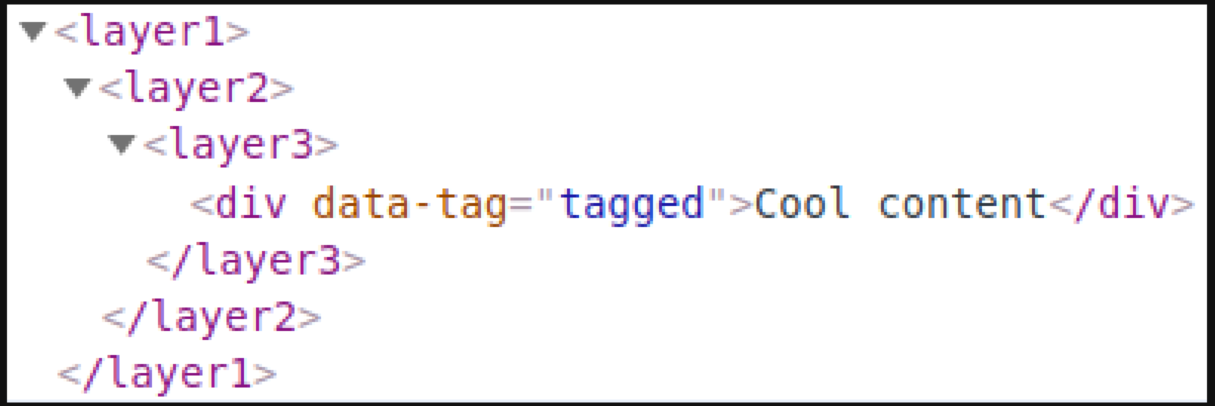
> *Use to get the QueryList of elements or directives from the content DOM. Any time a child element is added, removed, or moved, the query list will be updated, and the changes observable of the query list will emit a new value.*
>
> *Content queries are set before the ngAfterContentInit callback is called.*
>
> *Does not retrieve elements or directives that are in other components' templates, since a component's template is always a black box to its ancestors.*

```
@ContentChild(TagDirective) tagDirective: TagDirective;

@ContentChildren(TagDirective) tagDirectives: QueryList<TagDirective>;
```

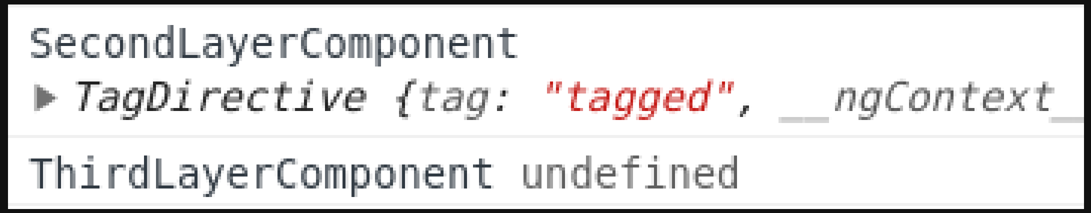
Doku:
https://angular.io/api/core/ContentChild
https://angular.io/api/core/ContentChildren

Projektion über mehrere Ebenen

```
@Component(
    {
        selector: 'layer1',
        template: `
        <layer2>
            <div [tag]="'tagged'">Cool content</div>
        </layer2>
        `
    }
)
export class FirstLayerComponent {}

@Component(
    {
        selector: 'layer2',
        template: `
        <layer3>
            <ng-content></ng-content>
        </layer3>`
    }
)
export class SecondLayerComponent {
    @ContentChild(TagDirective) tagDirective!: TagDirective;

    ngAfterContentInit(): void {
        console.log('SecondLayerComponent', this.tagDirective);
    }
}

@Component(
    {
        selector: 'layer3',
        template: `<ng-content></ng-content>`
    }
)
export class ThirdLayerComponent {
    @ContentChild(TagDirective) tagDirective!: TagDirective;

    ngAfterContentInit(): void {
        console.log('ThirdLayerComponent', this.tagDirective);
    }
}
```



```
▼<layer1>
  ▼<layer2>
    ▼<layer3>
       <div data-tag="tagged">Cool content</div>
     </layer3>
   </layer2>
 </layer1>
```



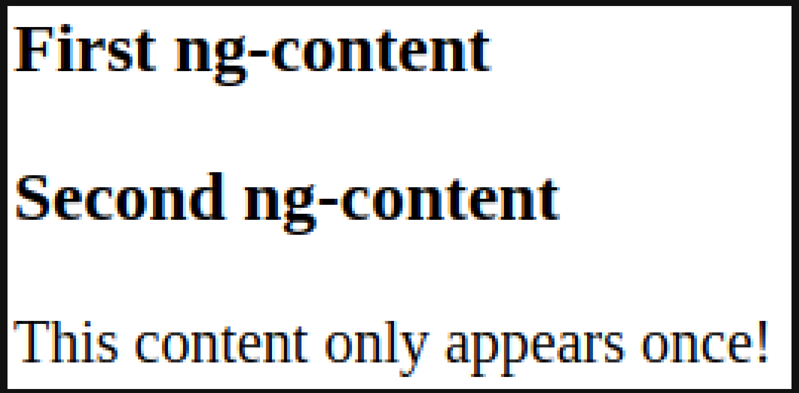
```
SecondLayerComponent
▶ TagDirective {tag: "tagged", __ngContext_
ThirdLayerComponent undefined
```

2x ng-content mit gleichem Selektor = 💥

```
@Component({
  changeDetection: ChangeDetectionStrategy.OnPush,
  selector: "app-ng-content-projecting-twice",
  template: `
    <app-content-twice>
      <p>This content only appears once!</p>
    </app-content-twice>
  `,
})
export class NgContentProjectingTwiceComponent {}
```

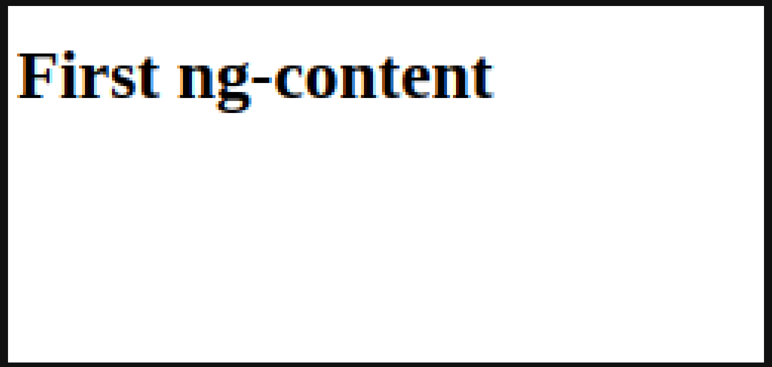
```
@Component({
  changeDetection: ChangeDetectionStrategy.OnPush,
  selector: "app-content-twice",
  template: `
    <div *ngIf="true">
      <h3>First ng-content</h3>
      <ng-content></ng-content>
    </div>
    <div *ngIf="true">
      <h3>Second ng-content</h3>
      <ng-content></ng-content>
    </div>
  `,
})
export class ContentTwiceComponent {}
```

```
@Component({
  changeDetection: ChangeDetectionStrategy.OnPush,
  selector: "app-content-twice",
  template: `
    <div *ngIf="true">
      <h3>First ng-content</h3>
      <ng-content></ng-content>
    </div>
    <div *ngIf="false">
      <h3>Second ng-content</h3>
      <ng-content></ng-content>
    </div>
  `,
})
export class ContentTwiceComponent {}
```


First ng-content

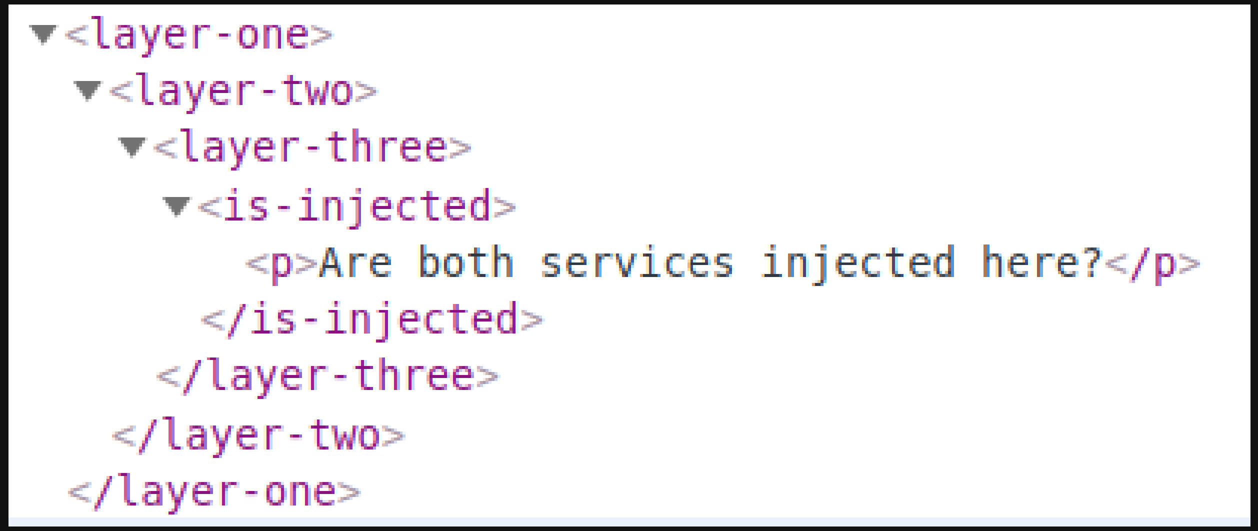
Second ng-content

This content only appears once!

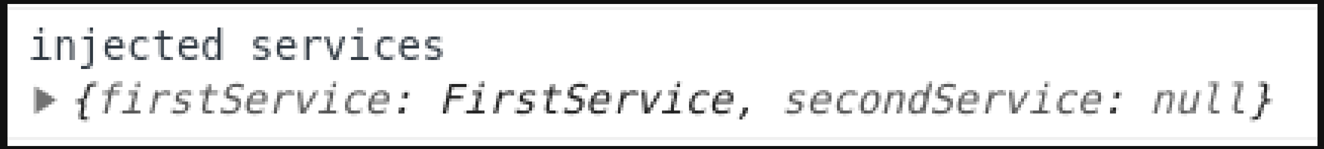

First ng-content

Injector (& Change Detection)

```
@Component(
    {
        selector: 'layer-one',
        template: `<layer-two><is-injected></is-injected></layer-two>`,
    }
)
export class LayerOneComponent {}

@Component(
    {
        selector: 'layer-two',
        template: `<layer-three><ng-content></ng-content></layer-three>`,
        providers: [FirstService]
    }
)
export class LayerTwoComponent {}

@Component(
    {
        selector: 'layer-three',
        template: `<ng-content></ng-content>`,
        providers: [SecondService]
    }
)
export class LayerThreeComponent {}

@Component(
    {
        selector: 'is-injected',
        template: `<p>Are both services injected here?</p>`
    }
)
export class IsInjectedComponent {
    constructor(
        @Optional() firstService: FirstService,
        @Optional() secondService: SecondService
    ) {
        console.log('injected services', {firstService, secondService});
    }
}
```



```
▼<layer-one>
  ▼<layer-two>
    ▼<layer-three>
      ▼<is-injected>
          <p>Are both services injected here?</p>
        </is-injected>
      </layer-three>
    </layer-two>
  </layer-one>
```

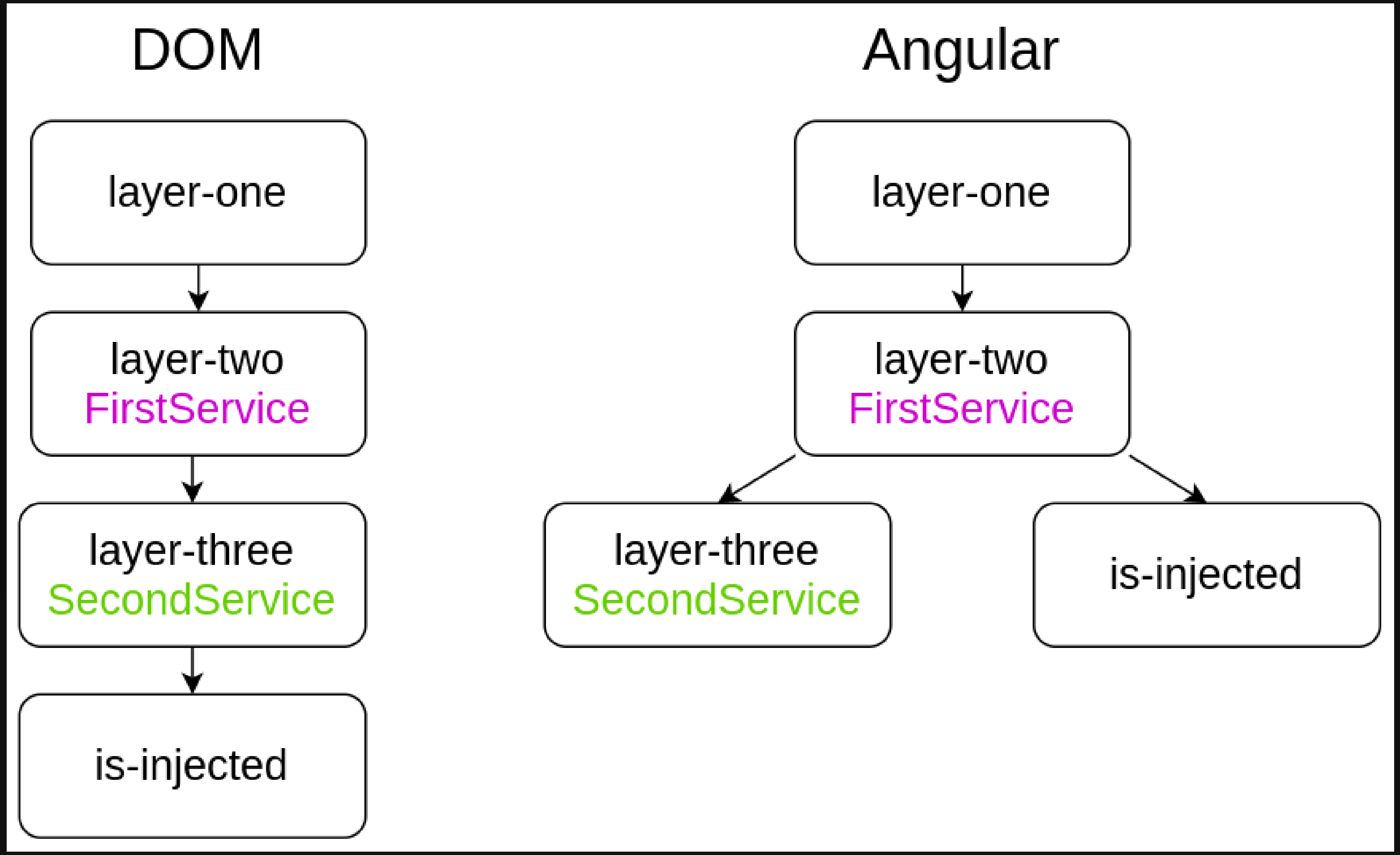


```
injected services
▶ {firstService: FirstService, secondService: null}
```

Injector (& Change Detection)

```
@Component(
    {
        selector: 'layer-one',
        template: `<layer-two><is-injected></is-injected></layer-two>`,
    }
)
export class LayerOneComponent {}

@Component(
    {
        selector: 'layer-two',
        template: `<layer-three><ng-content></ng-content></layer-three>`,
        providers: [FirstService]
    }
)
export class LayerTwoComponent {}

@Component(
    {
        selector: 'layer-three',
        template: `<ng-content></ng-content>`,
        providers: [SecondService]
    }
)
export class LayerThreeComponent {}

@Component(
    {
        selector: 'is-injected',
        template: `<p>Are both services injected here?</p>`
    }
)
export class IsInjectedComponent {
    constructor(
        @Optional() firstService: FirstService,
        @Optional() secondService: SecondService
    ) {
        console.log('injected services', {firstService, secondService});
    }
}
```

> *This is working as expected:*
> *nodes are only "projected" (moved to a different location) but everything else works relative to their source (= before projection) location.*



ng-template

*ngIf; else

```
@Component(
    {
        template: `
            <ng-container *ngIf="!!name; else noNameGiven">
                <p>Hallo, {{name}}!</p>
            </ng-container>

            <ng-template #noNameGiven>
                <p>Leider weiß ich deinen Namen noch nicht. Trotzdem: Hallo!</p>
            </ng-template>
        `
    }
)
export class NgIfElseComponent {
    @Input() name: string | undefined;
}
```

ngTemplateOutlet

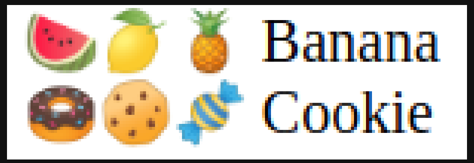
```
@Component(
    {
        template: `
            <ng-template #listItem let-item>
                <li>{{item}}</li>
            </ng-template>

            <ul>
                <ng-template *ngFor="let item of ['Apple', 'Banana', 'Cookie'];"
                    [ngTemplateOutlet]="listItem"
                    [ngTemplateOutletContext]="{$implicit: item}">
                </ng-template>
            </ul>
        `
    }
)
export class TemplateOutletComponent {}
```

Mehrere Templates

```
@Directive({ selector: "ng-template[templateType]" })
export class TemplateTypeDirective {
  @Input() templateType: string = "";

  constructor(public readonly template: TemplateRef<any>) {}
}
```

Mehrere Templates

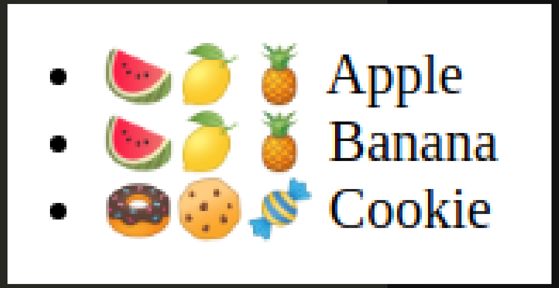
```
@Component({
    selector: "shopping-list",
    template: `
        <list-with-templates [items]="shoppingListEntries">
            <ng-template [templateType]="'fruit'" let-cartEntry>
              <li>🍉🍋🍍 {{ cartEntry.name }}</li>
            </ng-template>

            <ng-template [templateType]="'sweets'" let-cartEntry>
                <li>🍩🍪🍬 {{ cartEntry.name }}</li>
            </ng-template>
        </list-with-templates>
    `
})
export class ShoppingListComponent {
    shoppingListEntries = [
        { name: "Apple", type: "fruit" },
        { name: "Banana", type: "fruit" },
        { name: "Cookie", type: "sweets" },
        { name: "Pork", type: "meat" },
      ];
}
```



🍉🍋🍍 Banana
🍩🍪🍬 Cookie

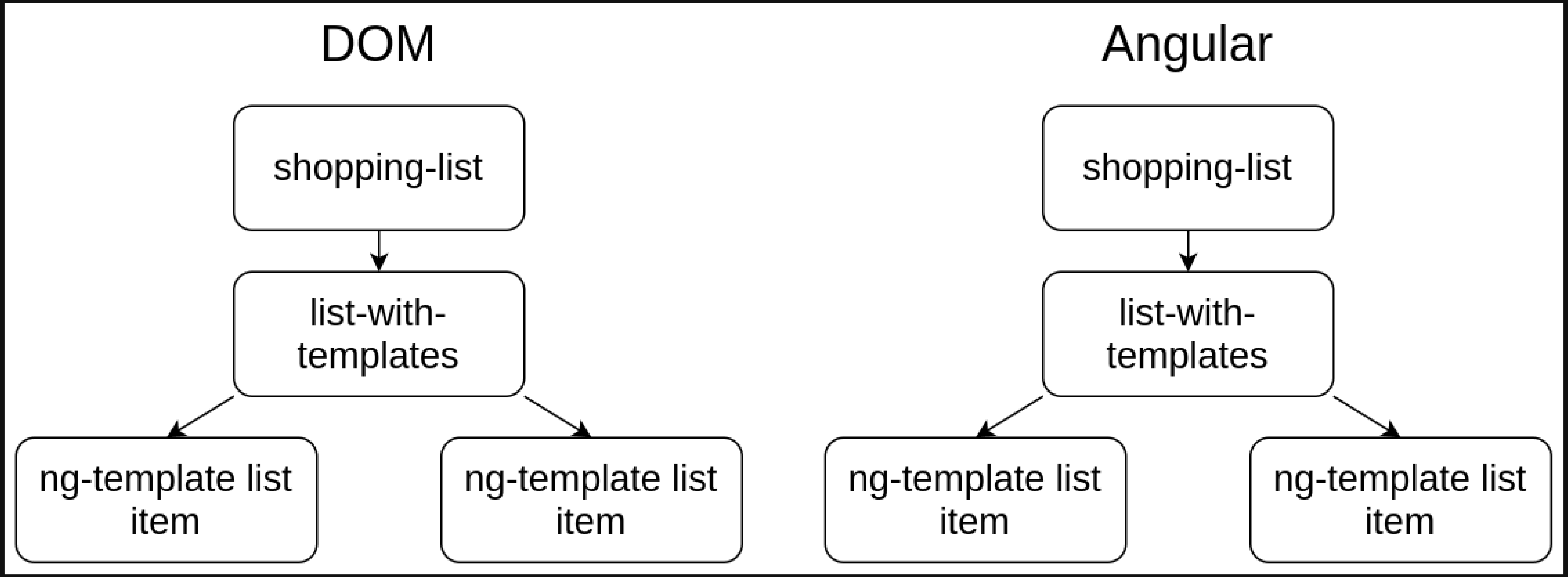
Mehrere Templates

```
@Component({
  selector: "list-with-templates",
  template: `
    <ul>
      <ng-template
          *ngFor="let item of items"
          [ngTemplateOutlet]="templates.get(item.type) || null"
          [ngTemplateOutletContext]="{ $implicit: item }"
      >
      </ng-template>
    </ul>
  `,
})
export class ListWithTemplatesComponent {
  @Input() items: any[] = [];

  templates = new Map<string, TemplateRef<any>>();

  @ContentChildren(TemplateTypeDirective) templateTypes!: QueryList<TemplateTypeDirective>;

  ngAfterContentInit(): void {
    this.templateTypes.forEach((templateTypeDirective: TemplateTypeDirective) => {
        this.templates.set(templateTypeDirective.templateType, templateTypeDirective.template);
      }
    );
  }
}
```



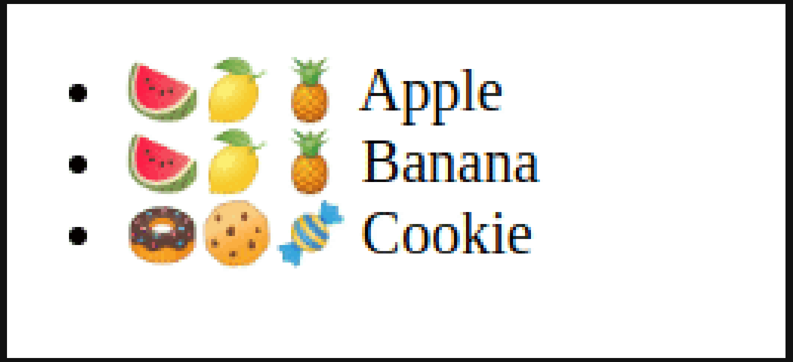
- 🍉🍋🍍 Apple
- 🍉🍋🍍 Banana
- 🍩🍪🍬 Cookie

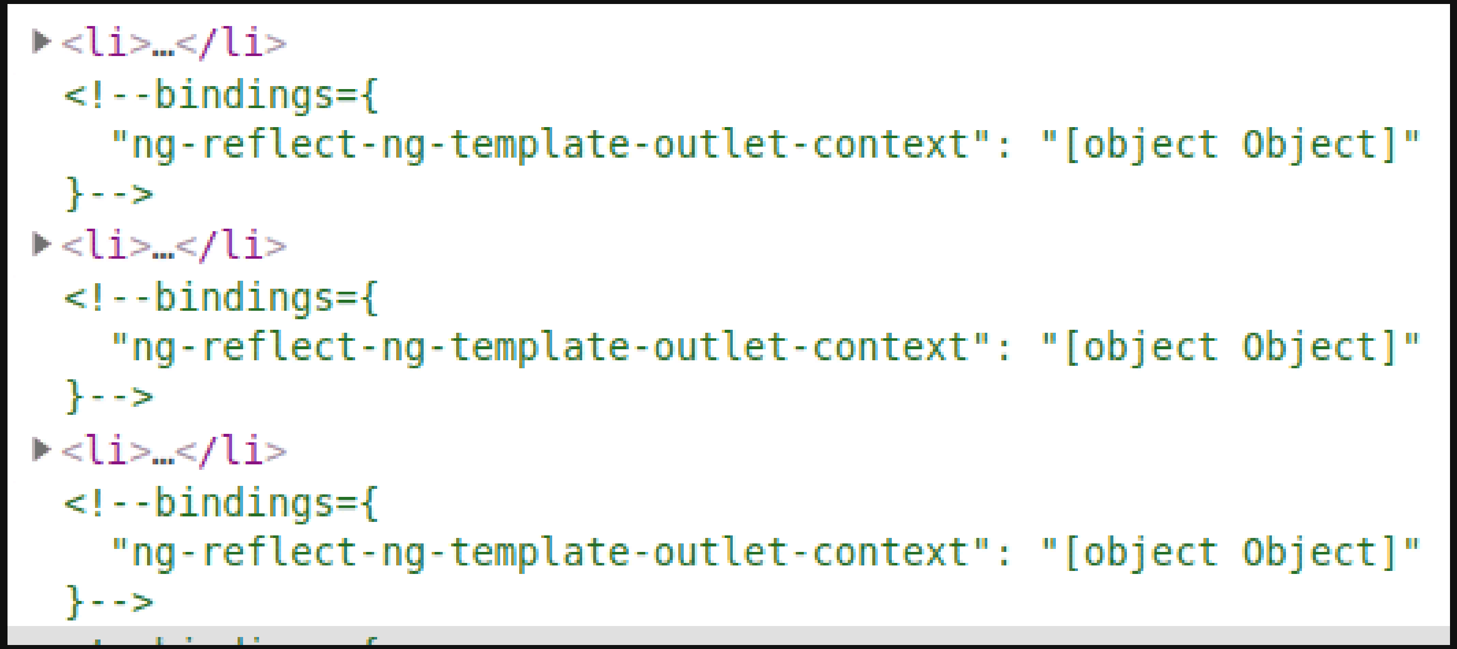
Injector Tree



Performance: trackBy

```
@Component({
    selector: "shopping-list",
    template: `
        <list-with-templates-trackBy
          [items]="shoppingListEntries">
            ...
        </list-with-templates-trackBy>
    `
})
export class ShoppingLisWithTrackByComponent {
    constructor(changeDetectorRef: ChangeDetectorRef) {
      timer(2_000, 2_000)
      .subscribe(() => {
        this.shoppingListEntries = this.shoppingListEntries
                                    .map(x => Object.assign({}, x));
        changeDetectorRef.markForCheck();
      });
    }
}

@Component({
  selector: "list-with-templates-trackBy",
  template: `
    <ul>
      <ng-template
        *ngFor="let item of items; trackBy: trackByFn"
        [ngTemplateOutlet]="templates.get(item.type) || null"
        [ngTemplateOutletContext]="{ $implicit: item }"
      >
      </ng-template>
    </ul>
  `,
})
export class ListWithTemplatesTrackByComponent {
  @Input() trackByFn: (index: number, item: any) => any = (_, item) => item;
  ...
```

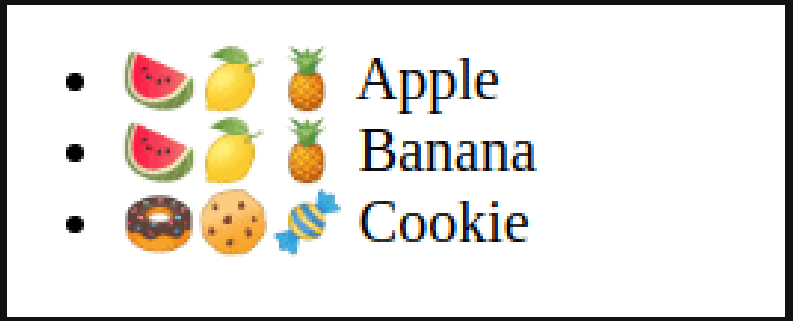

- 🍉🍋🍍 Apple
- 🍉🍋🍍 Banana
- 🍩🍪🍬 Cookie

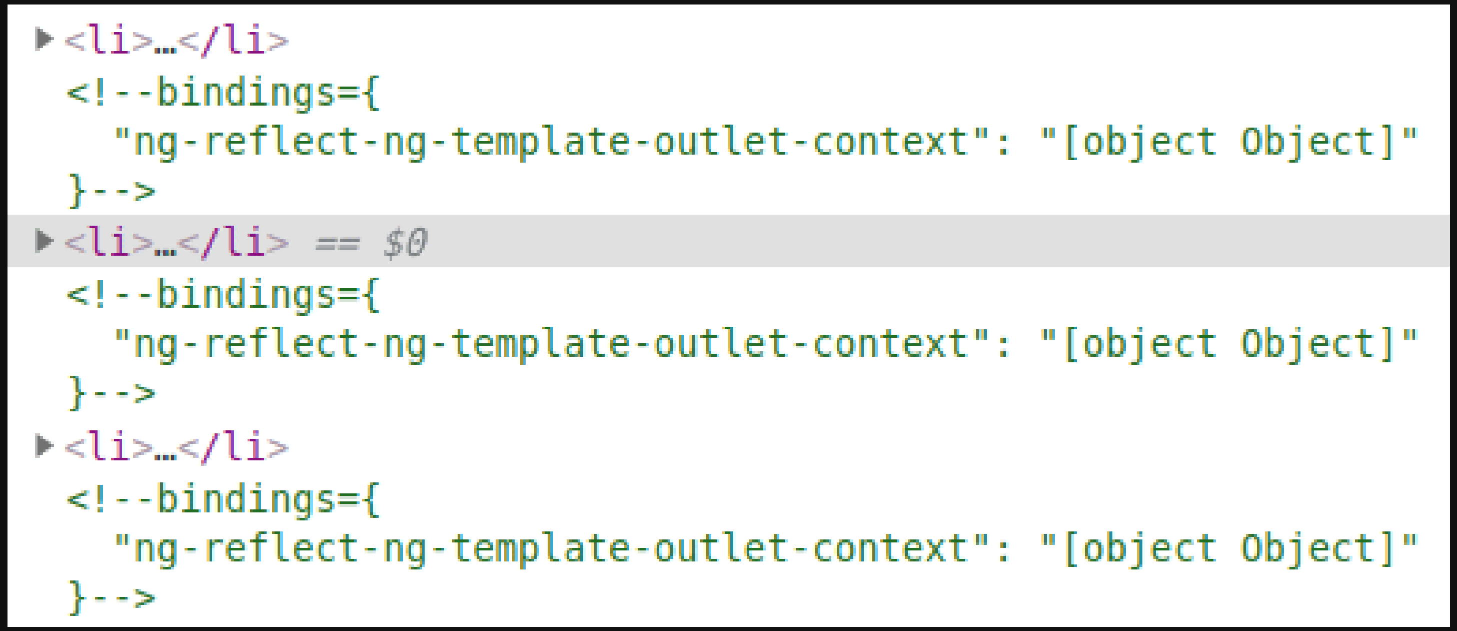


```
▶<li>…</li>
 <!--bindings={
   "ng-reflect-ng-template-outlet-context": "[object Object]"
 }-->
▶<li>…</li>
 <!--bindings={
   "ng-reflect-ng-template-outlet-context": "[object Object]"
 }-->
▶<li>…</li>
 <!--bindings={
   "ng-reflect-ng-template-outlet-context": "[object Object]"
 }-->
```

Performance: trackBy

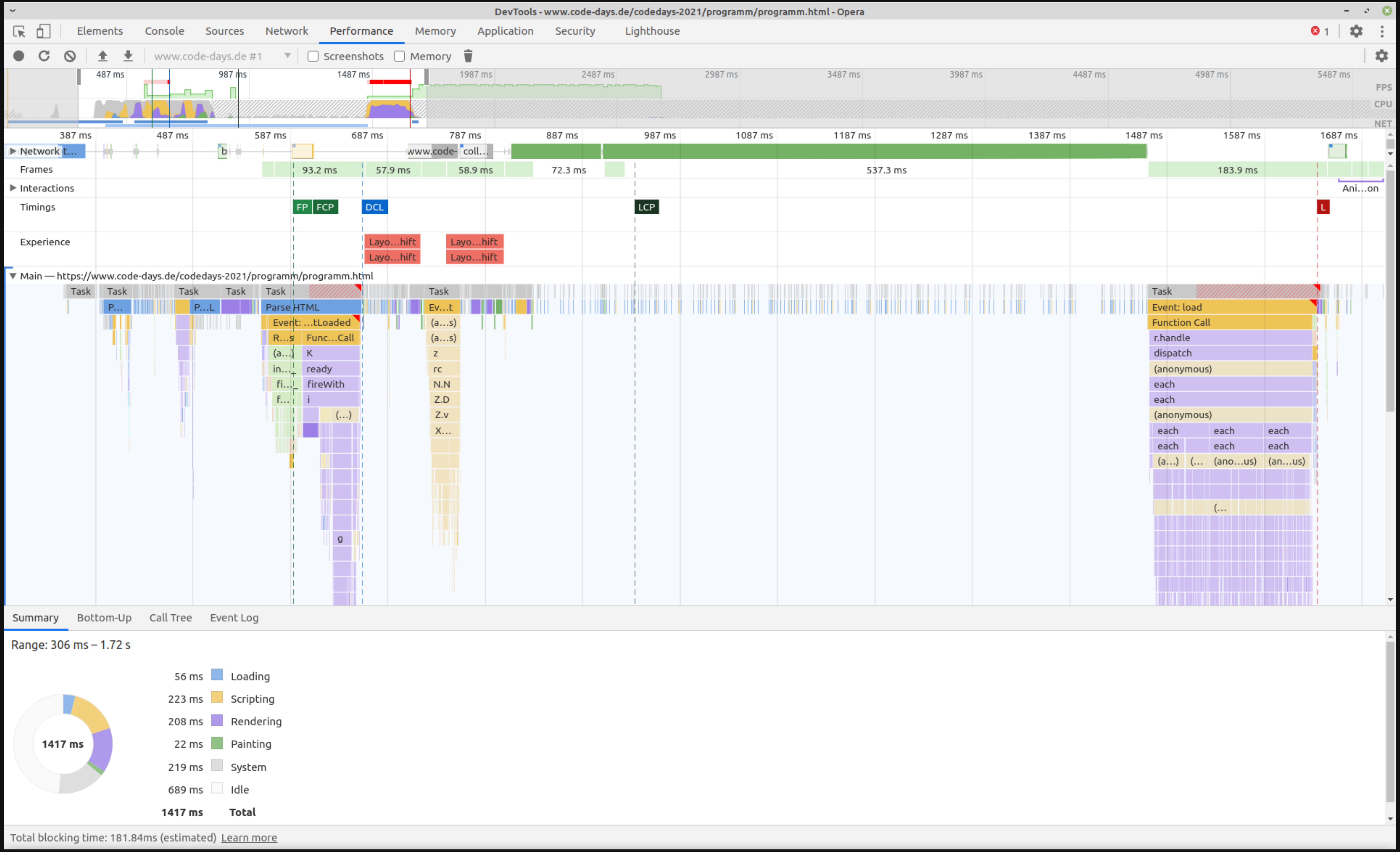
```
@Component({
    selector: "shopping-list",
    template: `
        <list-with-templates-trackBy
          [trackByFn]="trackByName"
          [items]="shoppingListEntries">
            ...
        </list-with-templates-trackBy>
    `
})
export class ShoppingLisWithTrackByComponent {
    trackByName = (index: number, item: any) => item.name;

    constructor(changeDetectorRef: ChangeDetectorRef) {
      timer(2_000, 2_000)
      .subscribe(() => {
        this.shoppingListEntries = this.shoppingListEntries
                                    .map(x => Object.assign({}, x));
        changeDetectorRef.markForCheck();
      });
    }
}

@Component({
  selector: "list-with-templates-trackBy",
  template: `
    <ul>
      <ng-template
        *ngFor="let item of items; trackBy: trackByFn"
        [ngTemplateOutlet]="templates.get(item.type) || null"
        [ngTemplateOutletContext]="{ $implicit: item }"
      >
      </ng-template>
    </ul>
  `,
})
export class ListWithTemplatesTrackByComponent {
  @Input() trackByFn: (index: number, item: any) => any = (_, item) => item;
```





```
▶<li>…</li>
 <!--bindings={
   "ng-reflect-ng-template-outlet-context": "[object Object]"
 }-->
▶<li>…</li> == $0
 <!--bindings={
   "ng-reflect-ng-template-outlet-context": "[object Object]"
 }-->
▶<li>…</li>
 <!--bindings={
   "ng-reflect-ng-template-outlet-context": "[object Object]"
 }-->
```

Performance

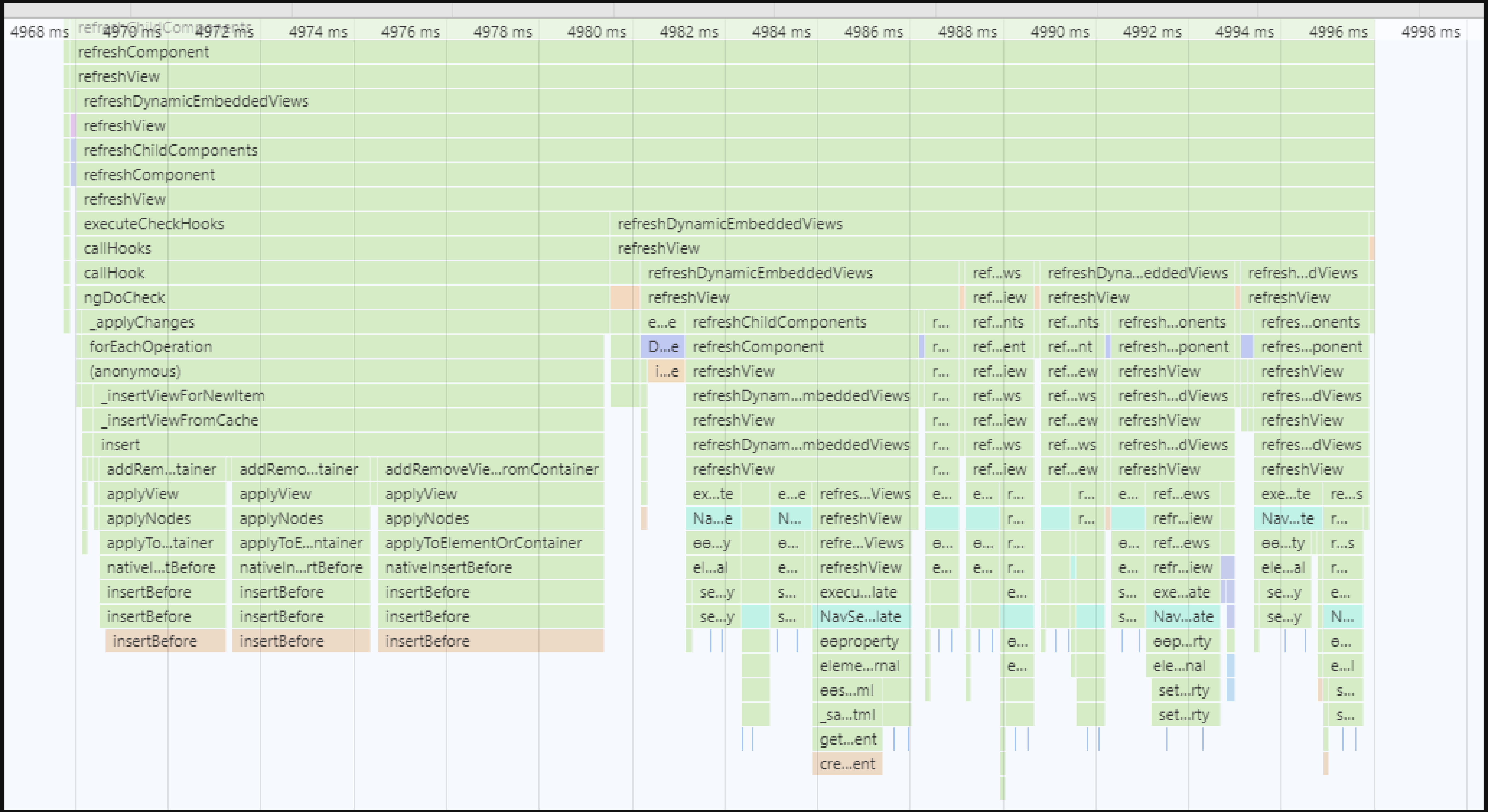


Range: 306 ms – 1.72 s

56 ms	🟦 Loading
223 ms	🟨 Scripting
208 ms	🟪 Rendering
22 ms	🟩 Painting
219 ms	⬜ System
689 ms	⬜ Idle
1417 ms	**Total**

Total blocking time: 181.84ms (estimated) Learn more

Performance



Doku-Links

- Offizielle Doku zu ng-template
- ng-content: The hidden docs
- Angular ng-template, ng-container and ngTemplateOutlet - The Complete Guide To Angular Templates

Danke

E-Mail	martin.grotz@mathema.de
Twitter	@mobilgroma
github	groma84
Slides	auf github