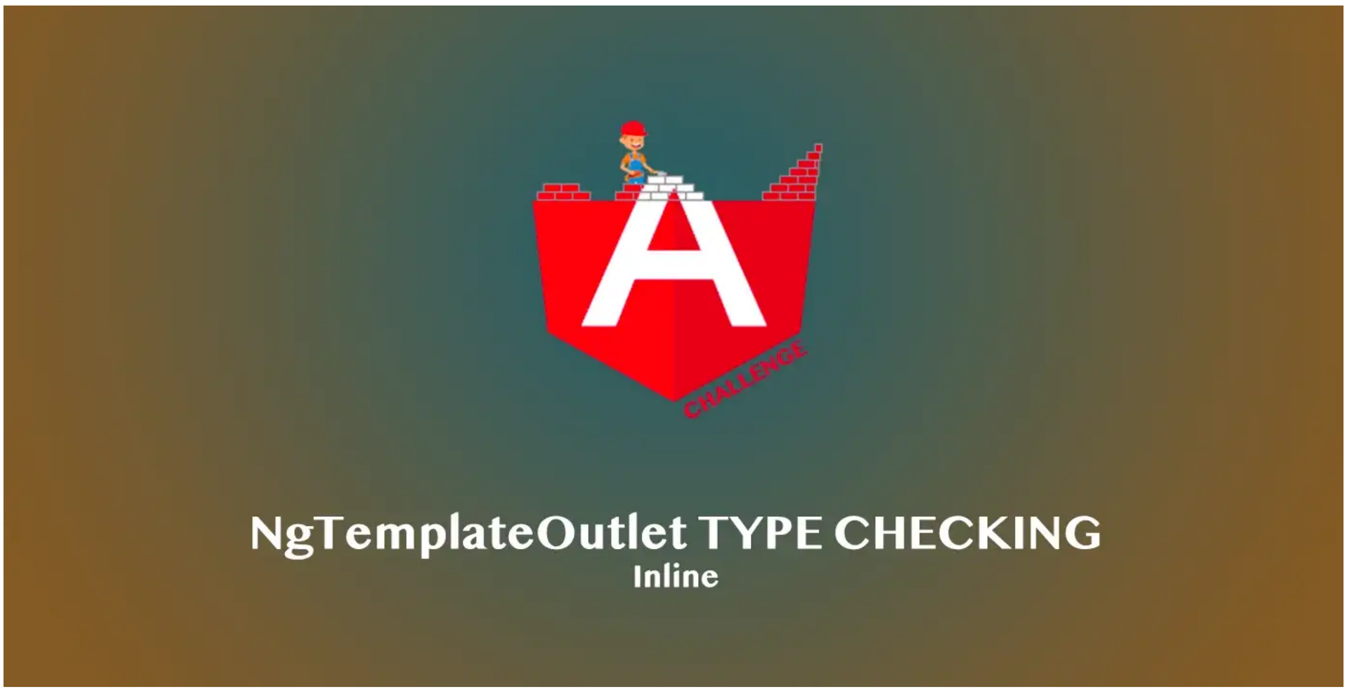
Thomas Laforge Follow

Dec 12 · 4 min read · Listen

Save

NgTemplateOutlet Typed Checking (Part 2)

Inline



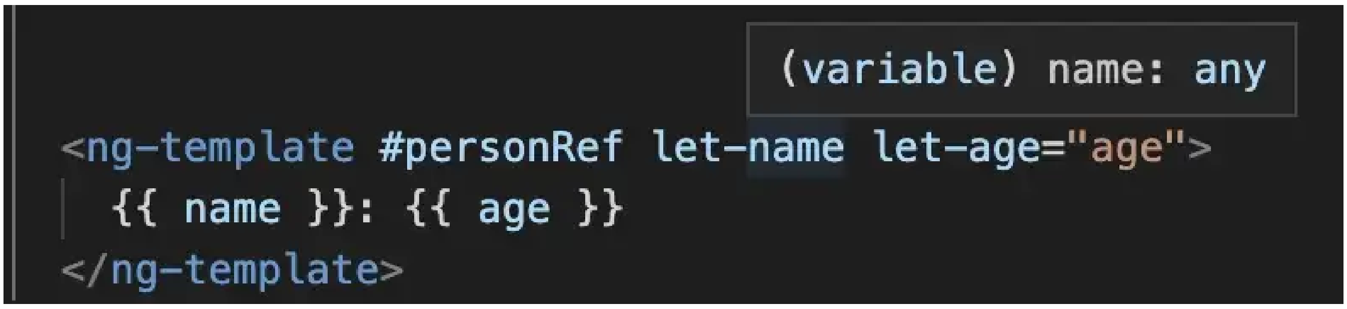
In my previous article, we saw how to strictly type-check the NgTemplateOutlet when it is used on a child component. In this article, we will see how to do this when the `TemplateRef` is located inside the same template as the `NgTemplateOutletDirective`

We define the following template where our `TemplateOutlet` directive is embedded inside the `#personRef` template.

```
<ng-container
  *ngTemplateOutlet="
```

```
      personRef;
      context: { $implicit: person.name, age: person.age }
    ">
  </ng-container>

  <ng-template #personRef let-name let-age="age">
    {{ name }}: {{ age }}
  </ng-template>
```

This template is working perfectly but we don't have any autocompletion or type inference.



```
(variable) name: any
<ng-template #personRef let-name let-age="age">
  {{ name }}: {{ age }}
</ng-template>
```

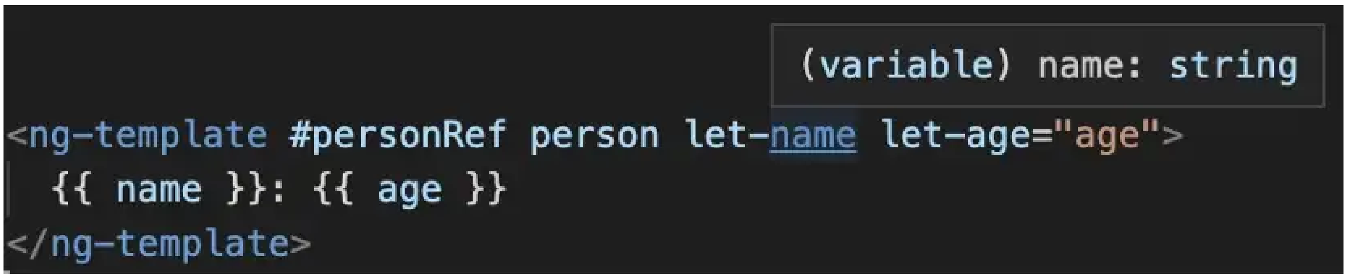
no type inference

I invite you to read my previous article on NgTemplateOutlet Type Checking and Directive Type Checking to have a better understanding over the following step.

As described in my previous article, we can create a directive with a `ngTemplateContextGuard` that defines our context `PersonContext` .

```
  interface PersonContext {
    $implicit: string;
    age: number;
  }

  @Directive({
    selector: 'ng-template[person]',
    standalone: true,
  })
  export class PersonDirective {
    static ngTemplateContextGuard(
      dir: PersonDirective,
      ctx: unknown
    ): ctx is PersonContext {
      return true;
    }
  }
```

We apply this directive on our template and that's it, we have a correct type inference.

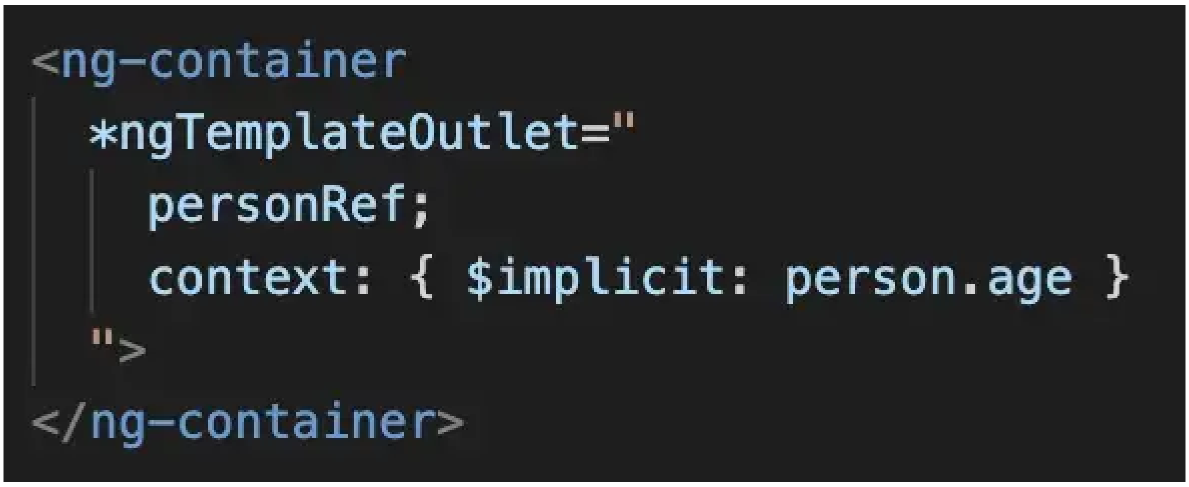


```
(variable) name: string
<ng-template #personRef person let-name let-age="age">
  {{ name }}: {{ age }}
</ng-template>
```

correct IDE type inference

However what about our directive? How can we apply strict type checking by passing our context? We can see in the image below that our compiler doesn't complain when we pass wrong property to our context.

> ***Remark:*** *as this time of writing, ngTemplateOutlet is not strictly typed and I'm using my own directive for the following part. (You can find the code here).*



```
<ng-container
  *ngTemplateOutlet="
    personRef;
    context: { $implicit: person.age }
  ">
</ng-container>
```

templateOutlet definition with wrong context and no compiler error

This part is a bit trickier because we need to get our template reference in order to get our context.

On our previous article, `@ContentChild` decorator's `read` meta property was doing this part for us. Now we have to do it ourself.

To do this, we will first inject our template reference inside our directive with the correct **context**. And to get that reference from outside, Angular exposes the `exportAs` option on the directive decorator.

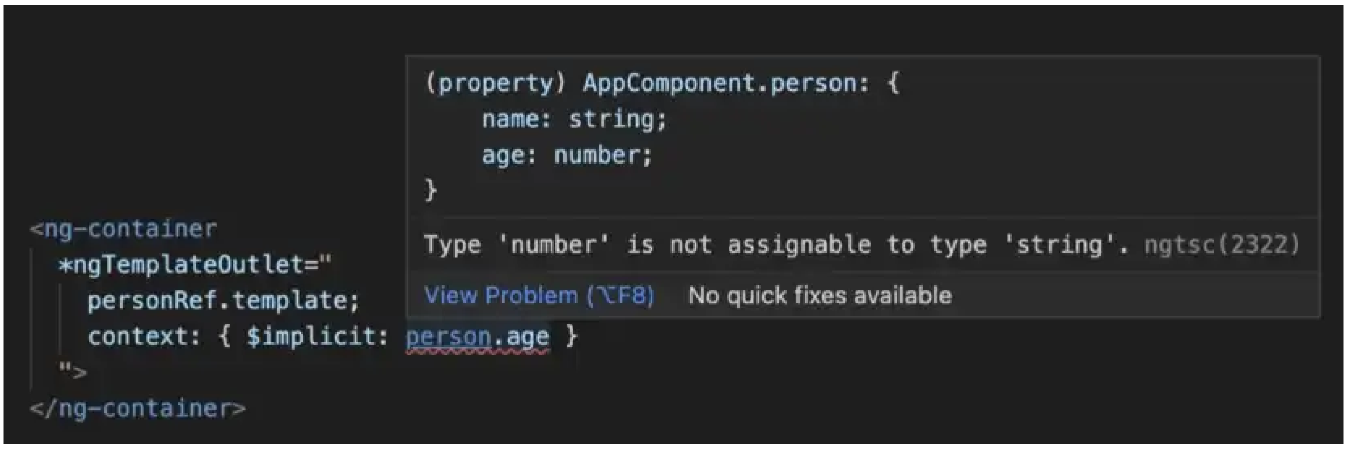
> `exportAs` *defines a name that can be used from the template.*

```
@Directive({
  selector: 'ng-template[person]',
  exportAs: 'personExportAsRef', // named this way for better clarity
  standalone: true,
})
export class PersonDirective {
  constructor(public template: TemplateRef<PersonContext>){}

  static ngTemplateContextGuard(
    dir: PersonDirective,
    ctx: unknown
  ): ctx is PersonContext {
    return true;
  }
}
```

Now we can export our directive inside our template and have access to its public internal properties.

```
<ng-container
  *ngTemplateOutlet="
    personRef.template; <!-- ref to directive TemplateRef -->
    context: { $implicit: person.name,  age: person.age }
  ">
</ng-container>

<!-- assign the directive to personRef to access internal properties -->
<ng-template #personRef="personExportAsRef" person let-name let-age="age">
  {{ name }}: {{ age }}
</ng-template>
```

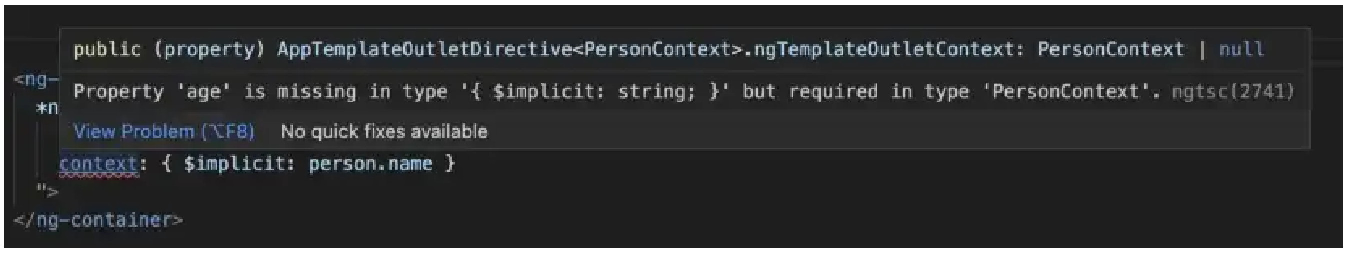
Now, the compiler is complaining if the property is of the wrong type or if one context property is missing.



```
<ng-container
  *ngTemplateOutlet="
    personRef.template;
    context: { $implicit: person.age }
  ">
</ng-container>
```

```
(property) AppComponent.person: {
    name: string;
    age: number;
}
Type 'number' is not assignable to type 'string'. ngtsc(2322)
```

compiler error on context wrong type



```
public (property) AppTemplateOutletDirective<PersonContext>.ngTemplateOutletContext: PersonContext | null
Property 'age' is missing in type '{ $implicit: string; }' but required in type 'PersonContext'. ngtsc(2741)
```

```
    context: { $implicit: person.name }
  ">
</ng-container>
```

compiler error on context missing params

. . .

NgTemplateOutlet and **Directives** should have no more secret for you. You can now create nice and secure application for your teammates.

If you haven’t read all related articles, I invite you to do it :

- Typescript Type Predicate
- Directive Type Checking
- NgTemplateOutlet Type Checking — with @ContentChild

. . .

If you found this article useful, please consider supporting my work by giving it some claps👏👏 to help it reach a wider audience. Don’t forget to share it with your teammates who might also find it useful. Your support would be greatly appreciated.

I hope you learned new Angular concept. You can find me on Medium, Twitter or Github. Don’t hesitate to ping me if you have more questions

> 👉 And if you want to accelerate your Angular learning journey, come and check out Angular challenges.

Angular Typescript Directives