ANGULAR

Stronger Type-Checking in Templates with Ivy

ngHeidelberg

Alex Rickabaugh
@synalx

ANGULAR

Stronger Type-Checking in Templates with Ivy

ngHeidelberg

Alex Rickabaugh
@synalx

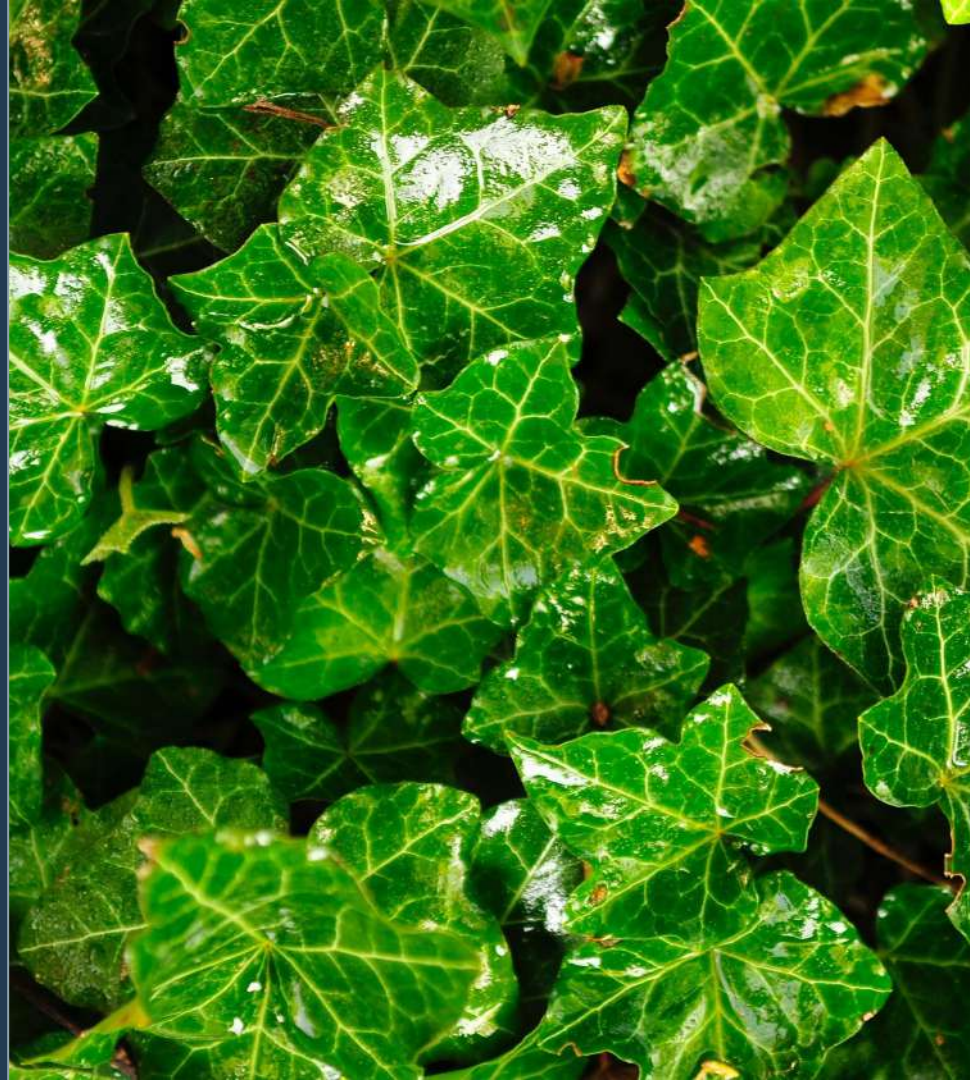
Alex Rickabaugh

Angular Framework Team

Angular Compiler

@synalx

strictTemplates



What is strictTemplates?

Strongest type-checking in templates

Similar to TypeScript `strict` flag

Why use it?

Detect bugs faster

Fewer runtime glitches

More meaningful tests

The “User” type

```
export interface User {
  name: {
    first: string;
    last: string;
  };
}
```

Color scheme

Green means complete checking

Amber means something is missing

Red means ignored

Outline

1. Comparison with previous template type-checking modes
2. Incrementally migrating your app to strictTemplates
3. How it works under the hood

Type Checking in v8

Basic mode

fullTemplateTypeCheck

Example template

```
<p>First: {{user.name.first}}</p>
<p *ngIf="showLastName">
  Last: {{user.name.last}}
</p>
```

Basic Mode

```
<p>First: {{user.name.first}}</p>
<p *ngIf="showLastName">
  Last: {{user.name.last}}
</p>
```

Checked in basic mode

```
<p>First: {{user.name.first}}</p>
<p *ngIf="showLastName">
  Last: {{user.lastName}}
</p>      Runtime glitch!
```

fullTemplateTypeCheck Mode

```
<p>First: {{user.name.first}}</p>
<p *ngIf="showLastName">
  Last: {{user.name.last}}
</p>
```

Examples

1. *ngFor loop variables
2. Bindings to @Inputs
3. $event type
4. Safe navigation

v8: ngFor

```
<div *ngFor="let user of users">
  <p>First: {{user.name.first}}</p>
  <p>Last: {{user.name.last}}</p>
</div>
```

v8: ngFor

```
<div *ngFor="let user of users">
  <p>Last: {{getLastName(user)}}</p>
</div>
```

strictTemplates: ngFor

```
<div *ngFor="let user of users">
  <p>First: {{user.name.first}}</p>
  <p>Last: {{user.name.last}}</p>
</div>
```

v8: Binding to @Inputs

```
<user-cmp
  [user]="myUser">
</user-cmp>
```

Consumer

```
<user-cmp
  [user]="'Alex'">
</user-cmp>
```

UserCmp

```
@Component({...})
export class UserCmp {
  @Input() user!: User;
}
```

v8: Binding to @Inputs

```
<user-cmp
  [user]="user$ | async">
</user-cmp>


TypeError: Cannot read property 'name' of null
```

strictTemplates: Binding to @Inputs

```
<user-cmp
  [user]="user$ | async">
</user-cmp>


ERROR in app.component.html:1:11 - error TS2322: Type
'User | null' is not assignable to type 'User'.
```

v8: $event type

```
<user-creator
  (save)="user = $event">
</user-creator>
```

strictTemplates: $event type

```
<user-creator
  (save)="user = $event">
</user-creator>
```

Safe navigation setup

```
@Component({...})
export class UserCmp {
  @Input() user?: User|undefined;

  formatName(name: string) {
    return name.trim();
  }
}
```

v8: Safe navigation

```
{{ formatName(user.name.first) }}


ERROR in src/app/user.component.html:1:15 - error TS2532: Object
is possibly 'undefined'.

1 {{ formatName(user.name.first) }}
                ~~~~~~~~~
```

v8: Safe navigation

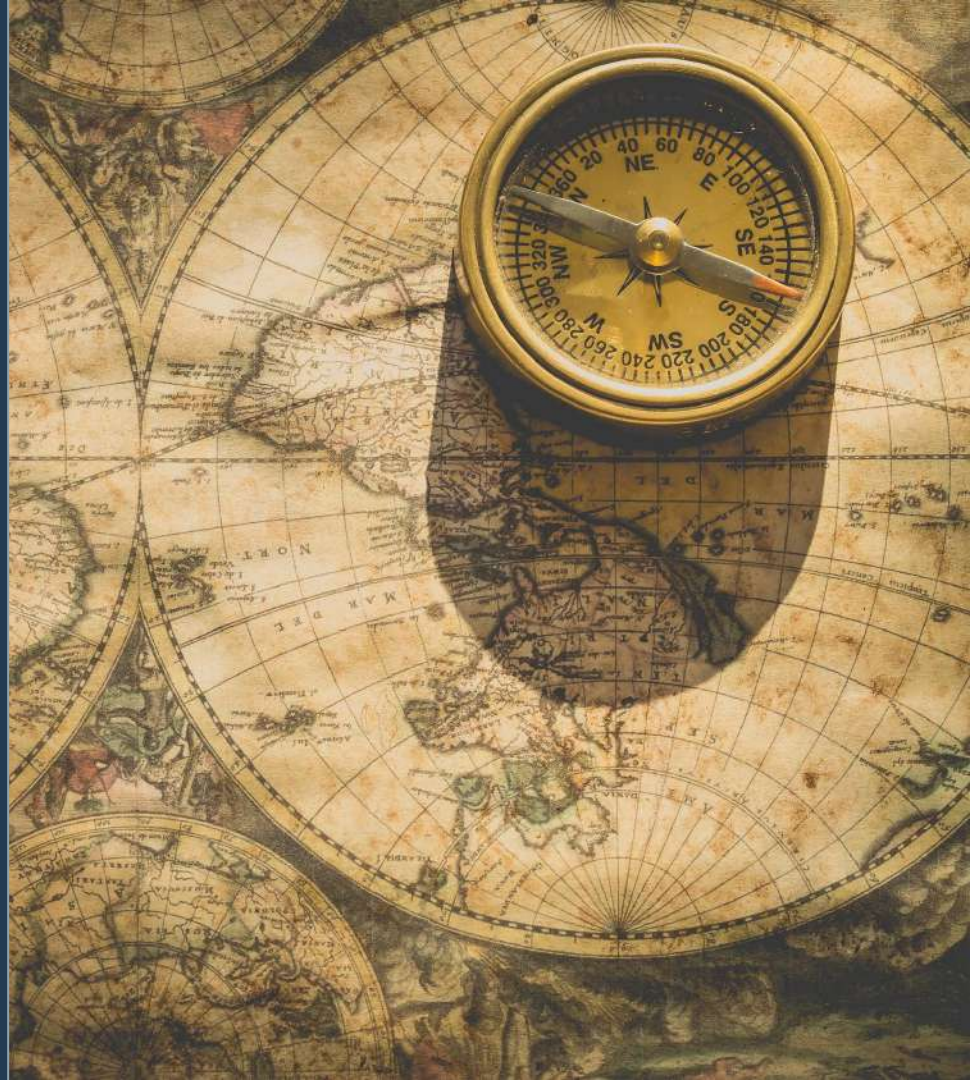
```
{{ formatName(user?.name?.first) }}


ERROR TypeError: name.trim is not a function
```

strictTemplates: Safe navigation

```
{{ formatName(user?.name?.first) }}
```

```
ERROR in src/app/app.component.html:1:15 - error TS2345: Argument
of type 'string | undefined' is not assignable to parameter of
type 'string'.
  Type 'undefined' is not assignable to type 'string'.
```

Migration



Enabling strictTemplates

1. Turn it on, see what's broken
2. Root cause analysis
3. Disable specific checks until you get a working build
4. Incrementally fix your app and re-enable checks

Why disable specific checks?

Using libraries that aren’t strict about typings

Templates have actual typing bugs that will take time to fix

Relying on previous type inference quirks

strictInputTypes: false

```
<user-cmp [user]="user">
</user-cmp>
```

Assignment of `user` to the component's `@Input` is not type-checked.

Use if: your components, or libraries, don't have accurate `@Input` types

strictNullInputTypes: false

```
<user-cmp [user]="user$ | async">
</user-cmp>
```

`null` can be assigned to an `@Input` which aren't specified as nullable

Use if: using libraries not compiled with `strictNullChecks`

strictAttributeTypes: false

```
<custom-button disabled>
</custom-button>
```

Don’t type-check any attribute-style bindings to an `@Input`.

Use if: using components/libraries which aren’t typed for attribute bindings

strictSafeNavigationTypes: false

```
{{ formatName(user?.name?.first) }}
```

Safe navigation operations produce an any type

Use if: you happened to be relying on this behavior previously

strictOutputEventTypes: false

```
<user-creator (save)="saveUser($event)">
</user-creator>
```

`$event` is left as the any type in an `@Output` binding

Use if: you happened to be relying on this behavior previously

strictDomEventTypes: false

```
<input
  (change)="onChange($event.target.value)">
</input>
```

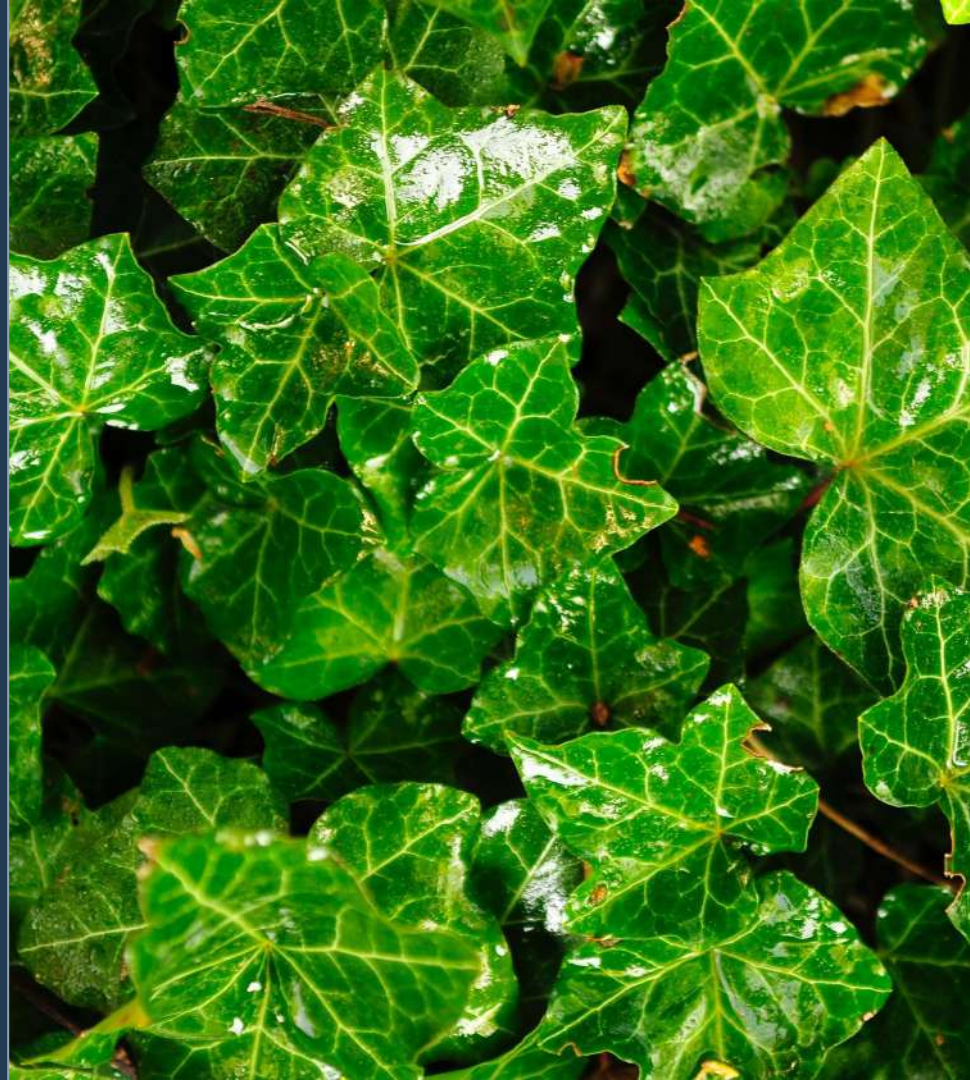
Use if: you happened to be relying on this behavior previously, or if your application uses `$event.target` extensively

And others

`strictDomLocalRefTypes`
`strictContextGenerics`
`strictLiteralTypes`

Read about these in our docs at
angular.io/guide/template-typecheck

How it works



Example Template

```
<div *ngFor=”let user of users”>
  Name: {{user.name.first}}
</div>
```

Template -> TypeScript

Generated Type-checking Code

```
const _ctor1: <T = any, U extends i3.NgIterable<T> = NgIterable<T>>(init: Pick<i2.NgForOf<T,
U>, "ngForOf" | "ngForTrackBy" | "ngForTemplate">) => i2.NgForOf<T, U> = (null!);

function _tcb2(ctx: i1.AppComponent) { if (true) {
    var _t1 = _ctor1({ "ngForOf": ((ctx).users /*25,30*/) /*5,31*/, "ngForTrackBy": (null as
any), "ngForTemplate": (null as any) }) /*0,32*/;
    var _t2: any = (null!);
    if (i2.NgForOf.ngTemplateContextGuard(_t1, _t2) /*0,32*/) {
        var _t3 = _t2.$implicit /*5,31*/;
        var _t4 = document.createElement("div") /*0,32*/;
        "" + ((_t3 /*37,41*/).name /*37,46*/).first /*37,52*/;
    }
} }
```

Generated Type-checking Code

```
const _ctor1: <T = any, U extends i3.NgIterable<T> =
    NgIterable<T>>(init: Pick<i2.NgForOf<T, U>, "ngForOf" |
"ngForTrackBy" | "ngForTemplate">) => i2.NgForOf<T, U> = (null!);
```

Generated Type-checking Code

```
function _tcb2(ctx: i1.AppComponent) { if (true) {
    var _t1 = _ctor1({ "ngForOf": ((ctx).users /*25,30*/)
/*5,31*/, "ngForTrackBy": (null as any), "ngForTemplate": (null
as any) }) /*0,32*/;
  …
} }
```

Generated Type-checking Code

```
function _tcb2(ctx: i1.AppComponent) { if (true) {
  …
    var _t2: any = (null!);
    if (i2.NgForOf.ngTemplateContextGuard(_t1, _t2) /*0,32*/) {
      …
    }
} }
```

Generated Type-checking Code

```
function _tcb2(ctx: i1.AppComponent) { if (true) {
    …
    if (…) {
        var _t3 = _t2.$implicit /*5,31*/;
        var _t4 = document.createElement("div") /*0,32*/;
        "" + ((_t3 /*37,41*/).name /*37,46*/).first /*37,52*/;
    }
} }
```

Summary

Use strictTemplates (and strict)!

Don’t be afraid to disable checks in order to migrate incrementally

Thank You!

Slides | bit.ly/strictTemplates

Alex Rickabaugh
@synalx