API > @angular/common

NgForOf DIRECTIVE

A structural directive that renders a template for each item in a collection. The directive is placed on an element, which becomes the parent of the cloned templates.

See more...

See also

- Structural Directives

Exported from

- `CommonModule`

Selectors

`[ngFor][ngForOf]`

Properties

Property	Description
`@Input()` `ngForOf: U & NgIterable<T>`	***Write-Only*** The value of the iterable expression, which can be used as a template input variable.

Property	Description

`@Input()`
`ngForTrackBy:`
`TrackByFunction<T>`
	Specifies a custom `TrackByFunction` to compute the identity of items in an iterable.
	
	If a custom `TrackByFunction` is not provided, `NgForOf` will use the item's object identity as the key.
	
	`NgForOf` uses the computed key to associate items in an iterable with DOM elements it produces for these items.
	
	A custom `TrackByFunction` is useful to provide good user experience in cases when items in an iterable rendered using `NgForOf` have a natural identifier (for example, custom ID or a primary key), and this iterable

Property	Description
	could be updated with new object instances that still represent the same underlying entity (for example, when data is re-fetched from the server, and the iterable is recreated and re-rendered, but most of the data is still the same). See also: • `TrackByFunction`
`@Input() ngForTemplate: TemplateRef<NgForOfContext<T, U>>`	***Write-Only*** A reference to the template that is stamped out for each item in the iterable. See also: • template reference variable

Description

The `ngForOf` directive is generally used in the shorthand form `*ngFor`. In this form, the template to be rendered for each iteration is the content of an anchor element containing the directive.

The following example shows the shorthand syntax with some options, contained in an `<li>` element.

```
<li *ngFor="let item of items; index as i; trackBy:
trackByFn">...</li>
```

The shorthand form expands into a long form that uses the `ngForOf` selector on an `<ng-template>` element. The content of the `<ng-template>` element is the `<li>` element that held the short-form directive.

Here is the expanded version of the short-form example.

```
<ng-template ngFor let-item [ngForOf]="items" let-
i="index" [ngForTrackBy]="trackByFn">
  <li>...</li>
</ng-template>
```

Angular automatically expands the shorthand syntax as it compiles the template. The context for each embedded view is logically merged to the current component context according to its lexical position.

When using the shorthand syntax, Angular allows only one structural directive on an element. If you want to iterate conditionally, for example, put the `*ngIf` on a container element that wraps the `*ngFor` element.

For futher discussion, see Structural Directives.

Local variables

`NgForOf` provides exported values that can be aliased to local variables.

For example:

```
<li *ngFor="let user of users; index as i; first as
isFirst">
  {{i}}/{{users.length}}. {{user}} <span
*ngIf="isFirst">default</span>
</li>
```

The following exported values can be aliased to local variables:

- `$implicit: T` : The value of the individual items in the iterable (`ngForOf`).
- `ngForOf: NgIterable<T>` : The value of the iterable expression. Useful when the expression is more complex then a property access, for example when using the async pipe (`userStreams | async`).
- `index: number` : The index of the current item in the iterable.
- `count: number` : The length of the iterable.
- `first: boolean` : True when the item is the first item in the iterable.
- `last: boolean` : True when the item is the last item in the iterable.
- `even: boolean` : True when the item has an even index in the iterable.
- `odd: boolean` : True when the item has an odd index in the iterable.

Change propagation

When the contents of the iterator changes, `NgForOf` makes the corresponding changes to the DOM:

- When an item is added, a new instance of the template is added to the DOM.
- When an item is removed, its template instance is removed from the DOM.
- When items are reordered, their respective templates are reordered in the DOM.

Angular uses object identity to track insertions and deletions within the iterator and reproduce those changes in the DOM. This has important implications for animations and any stateful controls that are present, such as `<input>` elements that accept user input. Inserted rows can be animated in, deleted rows can be animated out, and unchanged rows retain any unsaved state such as user input. For more on animations, see Transitions and Triggers.

The identities of elements in the iterator can change while the data does not. This can happen, for example, if the iterator is produced from an RPC to the server, and that RPC is re-run. Even if the data hasn't changed, the second response produces objects with different identities, and Angular must tear down the entire DOM and rebuild it (as if all old elements were deleted and all new elements inserted).

To avoid this expensive operation, you can customize the default tracking algorithm. by supplying the `trackBy` option to `NgForOf`. `trackBy` takes a function that has two arguments: `index` and `item`. If `trackBy` is given, Angular tracks changes by the return value of the function.

Static methods

ngTemplateContextGuard()

Asserts the correct type of the context for the template that `NgForOf` will render.

```
static ngTemplateContextGuard<T, U extends
NgIterable<T>>(dir: NgForOf<T, U>, ctx: any): ctx is
NgForOfContext<T, U>
```

Parameters

dir	`NgForOf<T, U>`
ctx	`any`

Returns

`ctx is NgForOfContext<T, U>`

The presence of this method is a signal to the Ivy template type-check compiler that the `NgForOf` structural directive renders its template with a specific context type.