A special element that can hold structural directives without adding new elements to the DOM.

The `<ng-container>` allows us to use structural directives without any extra element, making sure that the only DOM changes being applied are those dictated by the directives themselves.

This not only increases performance (even so slightly) since the browser ends up rendering less elements but can also be a valuable asset in having cleaner DOMs and styles alike.

It can for example enable us to use structural directives without breaking styling dependent on a precise DOM structure (as for example the ones we get when using flex containers, margins, the child combinator selector, etc...).

@usageNotes

With `*NgIf`s

One common use case of `<ng-container>` is alongside the `*ngIf` structural directive. By using the special element we can produce very clean templates easy to understand and work with.

For example, we may want to have a number of elements shown conditionally but they do not need to be all under the same root element. That can be easily done by wrapping them in such a block:

```
<ng-container *ngIf="condition">
  ...
</ng-container>
```

This can also be augmented with the an else statement alongside an `<ng-template>` as:

```
<ng-container *ngIf="condition; else templateA">
  ...
</ng-container>
<ng-template #templateA>
  ...
</ng-template>
```

Combination of multiple structural directives

Multiple structural directives cannot be used on the same element; if you need to take advantage of more than one structural directive, it is advised to use an `<ng-container>` per structural directive.

The most common scenario is with `*ngIf` and `*ngFor`. For example, let's imagine that we have a list of items but each item needs to be displayed only if a certain condition is true. We could be tempted to try something like:

```
<ul>
  <li *ngFor="let item of items" *ngIf="item.isValid">
    {{ item.name }}
  </li>
</ul>
```

As we said that would not work, what we can do is to simply move one of the structural directives to an `<ng-container>` element, which would then wrap the other one, like so:

```
<ul>
  <ng-container *ngFor="let item of items">
    <li *ngIf="item.isValid">
      {{ item.name }}
    </li>
  </ng-container>
</ul>
```

This would work as intended without introducing any new unnecessary elements in the DOM.

Use alongside ngTemplateOutlet

The `NgTemplateOutlet` directive can be applied to any element but most of the time it's applied to `<ng-container>` ones. By combining the two, we get a very clear and easy to follow HTML and DOM structure in which no extra elements are necessary and template views are instantiated where requested.

For example, imagine a situation in which we have a large HTML, in which a small portion needs to be repeated in different places. A simple solution is to define an `<ng-template>` containing our repeating HTML and render that where necessary by using `<ng-container>` alongside an `NgTemplateOutlet`.

Like so:

```
<!-- ... -->

<ng-container *ngTemplateOutlet="tmpl; context: {$implicit: 'Hello'}">
</ng-container>

<!-- ... -->

<ng-container *ngTemplateOutlet="tmpl; context: {$implicit: 'World'}">
</ng-container>

<!-- ... -->

<ng-template #tmpl let-text>
  <h1>{{ text }}</h1>
</ng-template>
```

For more information regarding `NgTemplateOutlet`, see the `NgTemplateOutlet` s api documentation page.