**Q=I18n?**

Ans=We need to install Angular CLI tool called ng-xi18n that will extract this text and place it into an [XLIFF](https://en.wikipedia.org/wiki/XLIFF) or [XMB](http://cldr.unicode.org/development/development-process/design-proposals/xmb) translation file, depending on your preference.

<https://scotch.io/tutorials/simple-language-translation-in-angular-2-part-1>

There are two ways to do this: You can use the **JiT** (Just in Time) Compiler or **AoT** (Ahead-of-Time) Compiler. Regardless of the approach, you will need to provide the Angular compiler with

* the translation file(s)
* the translation file format
* the Locale Id (autogenerated by ng-xi18n and can be found in the translation file)

# Importing the Translation Files with the AoT Compiler

To Internationalize with the AoT (Ahead of time) compiler, you will have to:

* pre-build a seperate application package for each language
* determine which language the user needs
* serve the appropriate application package

To pre-build a seperate application, you will have to ensure that you have the tools required to setup AoT. Refer to the [AoT cookbook](https://angular.io/docs/ts/latest/cookbook/aot-compiler.html) for details on how to do this.

Once your ready, use the ngc compile command providing the compiler with the following 3 options:

* --i18nFile: the path to the translation file
* --locale: the name of the locale
* --i18nFormat: the format of the localization file

For example, the French language command would look something like this:

./node\_modules/.bin/ngc --i18nFile=./locale/messages.fr.xlf --locale=fr --i18nFormat=xlf

# Importing translation files into your application with the JiT Compiler

The JiT (Just-in-time) compiler compiles the application dynamically, as the application loads. To do this, we will need to rely on 3 providers that tell the JiT compiler how to translate the template texts for a particular language:

* TRANSLATIONS is a string containing the content of the translation file.
* TRANSLATIONS\_FORMAT is the format of the file.
* LOCALE\_ID is the locale of the target language.

Here's how to boostrap the app with the translation providers for French. We're assuming the translation file is messages.fr.xlf.

app/index.ts:

import { NgModule, TRANSLATIONS, TRANSLATIONS\_FORMAT, LOCALE\_ID } from'@angular/core';

import { BrowserModule } from'@angular/platform-browser';

import { platformBrowserDynamic } from'@angular/platform-browser-dynamic';

import { Hello } from'./app.component.ts';

// Using SystemJs' text plugin

import translations from'./messages.fr.xlf!text';

const localeId = 'fr';

@NgModule({

imports: [

BrowserModule

],

declarations: [

Hello

],

bootstrap: [ Hello ]

})

exportclassAppModule{

}

platformBrowserDynamic().bootstrapModule(AppModule, {

providers: [

{ provide: TRANSLATIONS, useValue: translations },

{ provide: TRANSLATIONS\_FORMAT, useValue: 'xlf' },

{ provide: LOCALE\_ID, useValue: localeId }

]

});

Q=create custom directive ?

Ans=

@Directive({

selector: `[appConfirm]`

})

exportclass ConfirmDirective {

@HostListener('click', ['$event'])

confirmFirst(event: Event) {

returnwindow.confirm('Are you sure you want to do this?');

}

}

Q=@input and @Output?

### Ans=@Input

@Input decorator binds a property within one component (child component) to receive a value from another component (parent component). This is one way communication from parent to child. The component property should be annotated with @Input decorator to act as input property. A component can receive a value from another component using component property binding. Now we will see how to use **@Input**. It can be annotated at any type of property such as number, string, array or user defined class.

Find the use of @Input with string data type.

@Input()

ctMsg : string;

Now find array data type with @Input decorator. Here we are aliasing the property name. In the component property binding, alias name ctArray will be used.

@Input('ctArray')

myctArray : Array<string>

Now find @Input decorator with a property of user defined class type.

@Input('stdLeader')

myStdLeader : Student;

### @Output

@Output decorator binds a property of a component to send data from one component (child component) to calling component (parent component). This is one way communication from child to parent component. @Output binds a property of the type of angular EventEmitter class. This property name becomes custom event name for calling component. @Output decorator can also alias the property name as @Output(alias) and now this alias name will be used in custom event binding in calling component.   
Find the @Output decorator using aliasing.

@Output('addStudentEvent')

addStdEvent = new EventEmitter<Student>();

In the above code snippet addStudentEvent will become custom event name. Now find @Output decorator without aliasing.

@Output()

sendMsgEvent = new EventEmitter<string>();

### Component Property Binding using @Input

Find the steps for component property binding using @Input decorator step by step.   
1. In the parent component, first create a property. Here we are creating a property of our class Student type.

stdLeaderObj = new Student('Narendra', 'Modi');

2. Create a custom element in parent component that is a selector of one of our child component. Here we will perform component property binding.

<child-two [stdLeader] = "stdLeaderObj"></child-two>

3. Use @Input decorator to declare child component property as an input property that will receive value from parent using component property binding. Here we are using aliasing for property name.

@Input('stdLeader')

myStdLeader : Student;

4. Now we are ready to fetch values from input component property in our child component.

{{myStdLeader.fname +' '+ myStdLeader.lname}}

### Custom Event Binding using @Output and EventEmitter

Here we will discuss custom event binding using @Output decorator step by step.   
1. Create text box using element property binding in child component. input event is fired when there is any change in text box. $event.target.value fetches the current value of text box entered by user.

<div>

First Number :<input (input)="num1=$event.target.value" /><br/>

Second Number:<input (input)="num2=$event.target.value" /><br/>

<br/><button (click)="addNumber()">Add Number</button>

</div>

2. Find the method created in child component that will be fired when click event is invoked on click of button from above (step-1) code snippet. emit() is the method of EventEmitter class that emits event payload.

addNumber() {

this.addNumEvent.emit(parseInt(this.num1) + parseInt(this.num2));

}

3. In the child component, create an instance of EventEmitter annotated by @Output decorator. This instance will work as custom event name. Here we are using aliasing for custom event name.

@Output('addNumberEvent')

addNumEvent = new EventEmitter<number>();

4. Now we are performing custom event binding. The custom event addNumberEvent will be invoked in parent component when emit() method is invoked from child component. The event payload is accessed by $event object.

<child-two (addNumberEvent) = "printSum($event)" ></child-two>

5. The event addNumberEvent will call printSum() method.

printSum(res) {

this.sum = res;

}

<https://lishman.io/angular-2-component-input-output>

<https://www.concretepage.com/angular-2/angular-2-input-and-output-example>

# **@ContentChild and @ContentChildren Example**

They are used to fetch first or all elements from content DOM. @ContentChild gives first element matching the selector from the content DOM. @ContentChildren gives all elements of content DOM as QueryList. Contents queried by @ContentChild and @ContentChildren are set before ngAfterContentInit() is called. If we do any change in content DOM for the matching selector, that will be observed by @ContentChild and @ContentChildren and we will get updated value. As a selector for @ContentChild and @ContentChildren, we can pass directive, component or local template variable. By default @ContentChildren only selects direct children of content DOM and not all descendants. @ContentChildren has a metadata descendants and setting its value true, we can fetch all descendant elements.

|  |
| --- |
| @Component({ |
|  | selector: 'tabs', |
|  | template: ` |
|  | <ng-content></ng-content> |
|  | `, |
|  | }) |
|  | exportclassTabsComponent { |
|  | @ContentChildren(TabComponent) tabs:QueryList<TabComponent> |
|  |  |
|  | ngAfterContentInit() { |
|  | this.tabs.forEach(tabInstance=>console.log(tabInstance)) |
|  | } |
|  | } |

<https://netbasal.com/understanding-viewchildren-contentchildren-and-querylist-in-angular-896b0c689f6e>

<https://www.concretepage.com/angular-2/angular-2-4-contentchild-and-contentchildren-example>

### QueryList —

The return type of ViewChildren is QueryList. QueryList is just a fancy name for an object that stores a list of items. What is special about this object is when the state of the application changes Angular will automatically update the object items for you.

QueryList implements an iterable interface, therefore, it can be used in Angular templates with the ngFor directive. ( you can read more about this topic [here](https://netbasal.com/angular-2-ngfor-array-with-unique-values-6b15478d6484#.letjx3svz) )

#### QueryList API —

Getters —

* first — get the first item
* last — get the last item
* length — get the items length

Methods —

[map](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map)(), [filter](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/filter)() , [find](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/find)(), [reduce](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/reduce)(), [forEach](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/forEach)(), [some](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/some)().

* toArray() — returns the items as javascript array
* changes() — Changes can be observed by subscribing to the changes Observable. 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.

#### Remember —

The QueryList is initialized only before the ngAfterViewInit lifecycle hook, therefore, is available only **from** this point.

### ViewChildren vs ContentChildren —

ViewChildren **don’t** include elements that exist within the ng-content tag.

ContentChildren includes **only** elements that exists within the ng-contenttag.

Q=@viewChild and @ViewChildren?

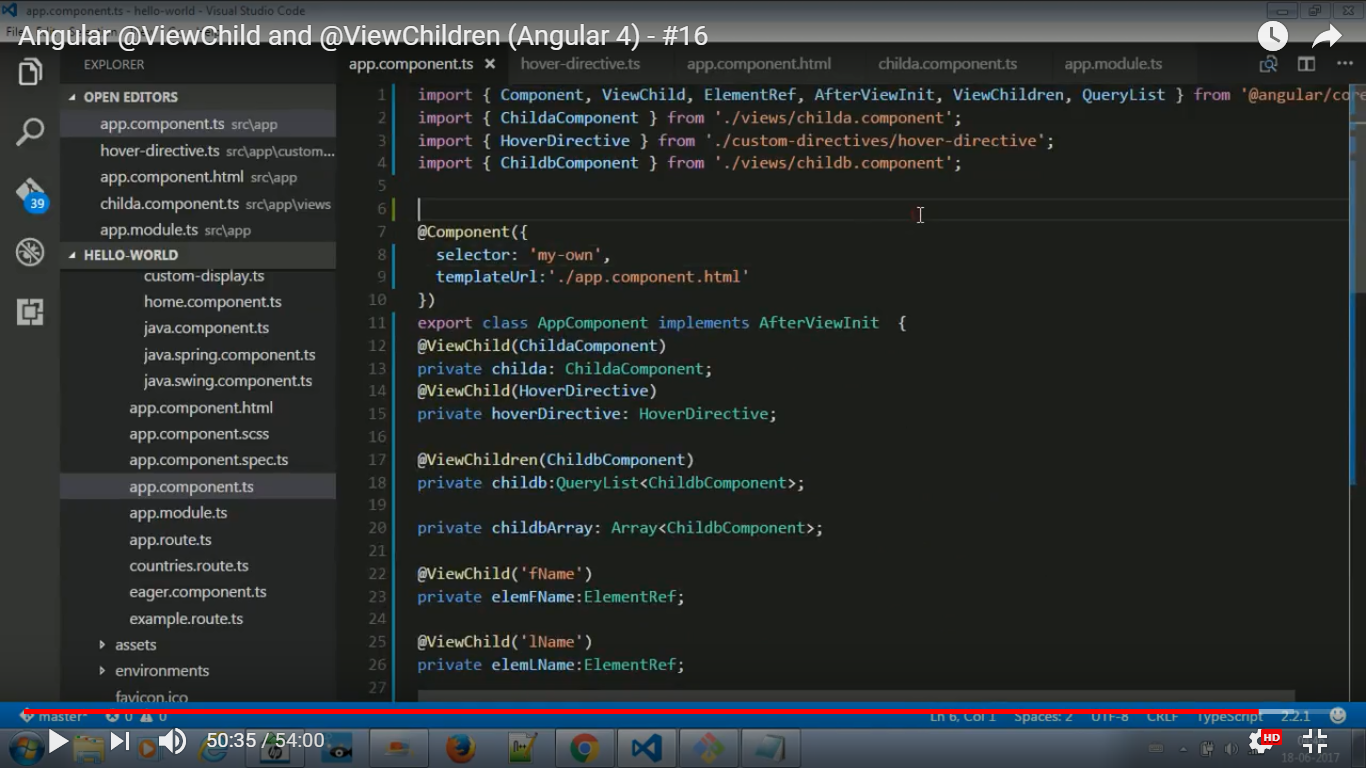
Ans= viewvhild and children is also same as @input and @output

But when we don’t want to change any thing In child and want the value of child the we need @vewChild.

From this we get the refrence of the child and then we can change any thing in child.

And if we have more than one child in DOM then we must use @VirewChildren from this we got the QueryList of all childs. It is array of child we can get access from index.

<https://www.youtube.com/watch?v=In8Mcy7vse8>



Q=what is ElementRef?

Ans= ElementRef is used for refrencing the DOM element.

# Q=**Angular 2 - Directives**

A **directive** is a custom HTML element that is used to extend the power of HTML.Directive are behavior of dom element.s

## Types of Directives

Angular 2 categorizes directives into 3 parts:

1. Directives with templates known as **Components**
2. Directives that creates and destroys DOM elements known as **Structural Directives**
3. Directives that manipulate DOM by changing behavior and appearance known as **Attribute Directives**

### Attribute Directives

Attribute directives, as the name goes, are applied as attributes to elements. They are used to manipulate the DOM in all kinds of different ways except creating or destroying them. I like to call them DOM-friendly directives.

ngstyle

ngClass

hidden

### Structural Directives

Structural directives are not DOM-friendly in the sense that they create, destroy, or re-create DOM elements based on certain conditions.

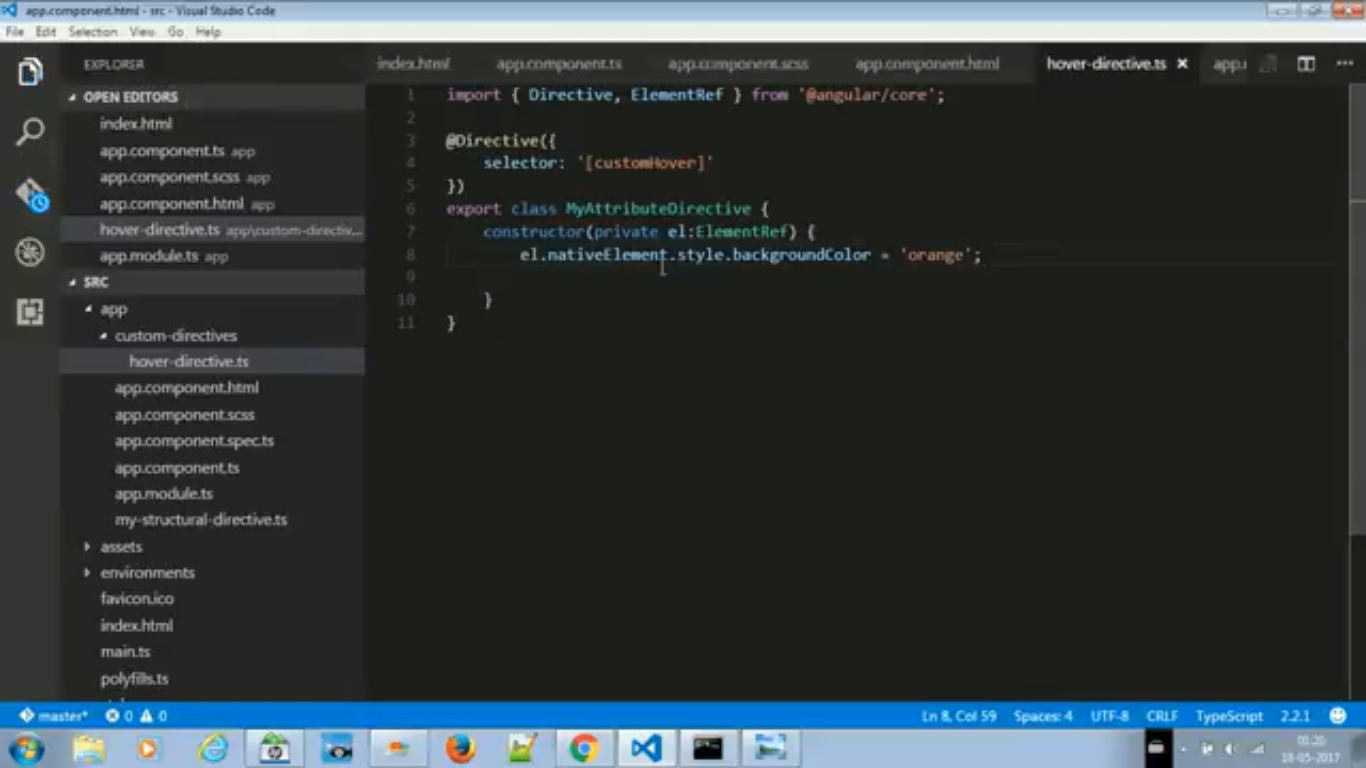
ngFor

ngIF

ngSwitch

<https://www.codementor.io/christiannwamba/build-custom-directives-in-angular-2-jlqrk7dpw>

### Custom Directives



*// ./app/shared/underline.directive.ts*

**import** { Directive, HostListener, Renderer, ElementRef } **from**'@angular/core';

@Directive({

selector: '[myUnderline]'

})

**exportclassUnderlineDirective**{

**constructor**(

private renderer: Renderer,

private el: ElementRef

){}

*// Event listeners for element hosting*

*// the directive*

@HostListener('mouseenter') onMouseEnter() {

**this**.hover(true);

}

@HostListener('mouseleave') onMouseLeave() {

**this**.hover(false);

}

*// Event method to be called on mouse enter and on mouse leave*

hover(shouldUnderline: boolean){

**if**(shouldUnderline){

*// Mouse enter this.renderer.setElementStyle(this.el.nativeElement, 'text-decoration', 'underline');*

} **else** {

*// Mouse leave this.renderer.setElementStyle(this.el.nativeElement, 'text-decoration', 'none');*

}

}

}

In the example above, we chose not to perform any action in the constructor or a lifecycle method. We chose to write a method called hover which is decorated with Host Listeners. The method is called by the host listeners and the host listeners are event listeners attached on the element hosting the directive.

***Host Listeners****are event listeners attached to any element that hosts (the directive is placed on) the directive.*

We can attach the directive to our template like so:

// ./app/app.component.html

<p><span myUnderline>Hover to underline</span></p>

# **The Ahead-of-Time (AOT) Compiler**

The Angular Ahead-of-Time (AOT) compiler converts your Angular HTML and TypeScript code into efficient JavaScript code during the build phase before the browser downloads and runs that code.

Angular offers two ways to compile your application:

1. *Just-in-Time* (JIT), which compiles your app in the browser at runtime
2. *Ahead-of-Time* (AOT), which compiles your app at build time.

JIT compilation is the default when you run the *build-only* or the *build-and-serve-locally* CLI commands:

**ng build**

**ng serve**

For AOT compilation, append the --aot flags to the *build-only* or the *build-and-serve-locally* CLI commands:

**ng build --aot**

**ng serve --aot**

The --prod meta-flag compiles with AOT by default.

### What is shadow DOM? How is it helping Angular 2 to perform better?

Shadow DOM is a part of the HTML spec which allows developers to encapsulate their HTML markup, CSS styles and JavaScript. Shadow DOM, along with a few other technologies, gives developers the ability to build their own 1st class tags, web components and APIs just like the <audio> tag. Collectively, these new tags and APIs are referred to as Web Components. Shadow DOM provides better separation of concern along with lesser conflict of styles and scripts with other HTML DOM elements.

Since shadow DOM are static in nature, it’s a good candidate to be cached as it is not accessible to developer. The cached DOM would be rendered faster in the browser providing better performance. Moreover, shadow DOM can be managed comparatively well while detecting the change in angular 2 application and re-paint of view can be managed efficiently.

<https://developer.mozilla.org/en-US/docs/Web/Web_Components/Shadow_DOM>

<https://glazkov.com/2011/01/14/what-the-heck-is-shadow-dom/>

<https://code.tutsplus.com/tutorials/intro-to-shadow-dom--net-34966>

### How would you optimize the angular 2 application for better performance?

Well, optimization depends on the type and size of application and many other factors. But in general, I would consider the following points while optimizing the angular 2 app:

1. Consider AOT compilation.
2. Make sure the application is bundled, uglified, and tree shaking is done.
3. Make sure the application doesn’t have un-necessary import statements.
4. Make sure that any 3rd party library, which is not used, is removed from the application.
5. Have all dependencies and dev-dependencies are clearly separated.
6. I would consider lazy loading instead of fully bundled app if the app size is more.

**Further Reading:**

<https://medium.com/@areai51/the-4-stages-of-perf-tuning-for-your-angular2-app-922ce5c1b294#.pw4m2srmr>

<https://www.lucidchart.com/techblog/2016/05/04/angular-2-best-practices-change-detector-performance/>

### What are the security threats should we be aware of in angular 2 application?

Just like any other client side or web application, angular 2 application should also follow some of the basic guidelines to mitigate the security risks. Some of them are:

1. Avoid using/injecting dynamic Html content to your component.
2. If using external Html, that is coming from database or somewhere outside the application, sanitize it.
3. Try not to put external urls in the application unless it is trusted. Avoid url re-direction unless it is trusted.
4. Consider using AOT compilation or offline compilation.
5. Try to prevent XSRF attack by restricting the api and use of the app for known or secure environment/browsers.

Q=difference between ngif and \*ngif?

Ans=ngIf is the directive. Because it's a structural directive (template-based), you need to use the \* prefix to use it into templates.

\*ngIf corresponds to the shortcut for the following syntax ("syntactic sugar"):

<template [ngIf]="condition">

<p>

Our heroes are true!

</p>

</template>

Equivalent to:

<p \*ngIf="condition">

Our heroes are true!

</p>

Q=difference between stateful and stateless component?

Q=interpolation in angular?

Q=encaptulation in angular?

Q=what is Shadow DOM?

Q= what is @Optional?

Q=how to resolve collision in angular 2 dependency injection?

Q=Opaque Token?