How can I select an element in a component template?

Asked 3 years, 11 months ago Active 2 months ago Viewed 444k times

447

153

Does anybody know how to get hold of an element defined in a component template? Polymer makes it really easy with the `$` and `$$`.

I was just wondering how to go about it in Angular.

Take the example from the tutorial:

```
import {Component} from '@angular/core'

@Component({
    selector:'display'
    template:`
     <input #myname(input)="updateName(myname.value)"/>
     <p>My name : {{myName}}</p>
    `

})
export class DisplayComponent {
    myName: string = "Aman";
    updateName(input: String) {
        this.myName = input;
    }
}
```

How do I catch hold of a reference of the `p` or `input` element from within the class definition?

angular typescript angular-components

edited Nov 16 '18 at 7:49
laike9m
9,206 10 70 96

asked Sep 21 '15 at 10:34
Aman Gupta
2,585 4 12 20

815

Instead of injecting `ElementRef` and using `querySelector` or similar from there, a declarative way can be used instead to access elements in the view directly:

```
<input #myname>
```

```
@ViewChild('myname') input;
```

element

```
ngAfterViewInit() {
  console.log(this.input.nativeElement.value);
}
```

StackBlitz example

- @ViewChild() supports directive or component type as parameter, or the name (string) of a template variable.
- @ViewChildren() also supports a list of names as comma separated list (currently no spaces allowed `@ViewChildren('var1,var2,var3')`).
- @ContentChild() and @ContentChildren() do the same but in the light DOM (`<ng-content>` projected elements).

descendants

`@ContentChildren()` is the only one that allows to also query for descendants

```
@ContentChildren(SomeTypeOrVarName, {descendants: true}) someField;
```

~~`{descendants: true}` should be the default but is not in 2.0.0 final and it's considered a bug~~
This was fixed in 2.0.1

read

If there are a component and directives the `read` parameter allows to specify which instance should be returned.

```
@ViewChild('myname', { read: ViewContainerRef }) target;
```

subscribe changes

Even though view children are only set when `ngAfterViewInit()` is called and content children are only set when `ngAfterContentInit()` is called, if you want to subscribe to changes of the query result, it should be done in `ngOnInit()`

https://github.com/angular/angular/issues/9689#issuecomment-229247134

```
@ViewChildren(SomeType) viewChildren;
@ContentChildren(SomeType) contentChildren;

ngOnInit() {
  this.viewChildren.changes.subscribe(changes => console.log(changes));
  this.contentChildren.changes.subscribe(changes => console.log(changes));
}
```

direct DOM access

can only query DOM elements, but not components or directive instances:

```
export class MyComponent {
  constructor(private elRef:ElementRef) {}
  ngAfterViewInit() {
    var div = this.elRef.nativeElement.querySelector('div');
    console.log(div);
  }

  // for transcluded content
  ngAfterContentInit() {
    var div = this.elRef.nativeElement.querySelector('div');
    console.log(div);
  }
}
```

get arbitrary projected content

See Access transcluded content

12 The angular teams advised against using ElementRef, this is the better solution. – Honorable Chow Mar 30 '16 at 14:32

6 Actually `input` also is an `ElementRef`, but you get the reference to the element you actually want, instead of querying it from the host `ElementRef`. – Günter Zöchbauer Mar 30 '16 at 14:35

26 Actually using `ElementRef` is just fine. Also using `ElementRef.nativeElement` with `Renderer` is fine. What **is discouraged** is accessing properties of `ElementRef.nativeElement.xxx` directly. – Günter Zöchbauer Jun 3 '16 at 12:33

2 @Natanael I don't know if or where this is explicitly documented but it is mentioned regularly in issues or other discussions (also from Angular team members) that direct DOM access should be avoided. Accessing the DOM directly (which is what accessing properties and methods of `ElementRef.nativeElement)` is, prevents you from using Angulars server side rendering and WebWorker feature (I don't know if it also breaks the upcoming offline template compiler - but I guess not). – Günter Zöchbauer Jun 14 '16 at 10:30

9 As mentioned above in the *read* section, if you want to get the nativeElement for an element with ViewChild, you have to do the following: `@ViewChild('myObj', { read: ElementRef }) myObj: ElementRef;` – jsgoupil Aug 18 '16 at 23:02

191

You can get a handle to the DOM element via `ElementRef` by injecting it into your component's constructor:

```
constructor(myElement: ElementRef) { ... }
```

Docs: https://angular.io/docs/ts/latest/api/core/index/ElementRef-class.html

edited Aug 3 '16 at 16:24
Khaled Al-Ansari
2,820 1 18 24

answered Sep 21 '15 at 11:19
Brocco
44.3k 8 58 72

1 @Brocco can you update your answer? I'd like to see a current solution since `ElementRef` is gone. – Jefftopia Nov 24 '15 at 2:07

23 `ElementRef` is available (again?). – Günter Zöchbauer Feb 4 '16 at 19:15

9 link Use this API as the **last resort** when direct access to DOM is needed. Use templating and data-binding provided by Angular instead. Alternatively you take a look at Renderer which provides API that can safely be used even when direct access to native elements is not supported. Relying on direct DOM access creates tight coupling between your application and rendering layers which will make it impossible to separate the two and deploy your application into a web worker. – sandeep talabathula Apr 26 '17 at 10:40

@sandeeptalabathula What is a better option for finding an element to attach a floating date picker component from a third-party library to? I'm aware that this wasn't the original question, but you make it out that finding elements in the DOM is bad in all scenarios... – John Jul 24 '17 at 5:52

46

```
import { Component, ElementRef, OnInit } from '@angular/core';

@Component({
  selector:'display',
  template:`
   <input (input)="updateName($event.target.value)">
   <p> My name : {{ myName }}</p>
  `
})
class DisplayComponent implements OnInit {
  constructor(public element: ElementRef) {
    this.element.nativeElement // <- your direct element reference
  }
  ngOnInit() {
    var el = this.element.nativeElement;
    console.log(el);
  }
  updateName(value) {
    // ...
  }
}
```

Example updated to work with the latest version

For more details on native element, here

edited May 24 '17 at 14:48

answered Sep 22 '15 at 6:17
gdi2290
776 4 7

18

Angular 4+: Use `renderer.selectRootElement` with a CSS selector to access the element.

I've got a form that initially displays an email input. After the email is entered, the form will be expanded to allow them to continue adding information relating to their project. However, if they are **not** an existing client, the form will include an address section above the project information section.

I tried the solutions with no success. However, Update 3 in this answer gave me half of the eventual solution. The other half came from MatteoNY's response in this thread. The result is this:

```
import { NgZone, Renderer } from '@angular/core';

constructor(private ngZone: NgZone, private renderer: Renderer) {}

setFocus(selector: string): void {
    this.ngZone.runOutsideAngular(() => {
        setTimeout(() => {
            this.renderer.selectRootElement(selector).focus();
        }, 0);
    });
}

submitEmail(email: string): void {
    // Verify existence of customer
    ...
    if (this.newCustomer) {
        this.setFocus('#firstname');
    } else {
        this.setFocus('#description');
    }
}
```

Since the only thing I'm doing is setting the focus on an element, I don't need to concern myself with change detection, so I can actually run the call to `renderer.selectRootElement` outside of Angular. Because I need to give the new sections time to render, the element section is wrapped in a timeout to allow the rendering threads time to catch up before the element selection is attempted. Once all that is setup, I can simply call the element using basic CSS selectors.

I know this example dealt primarily with the focus event, but it's hard for me that this couldn't be used in other contexts.

edited May 23 '17 at 12:02
Community ♦
1 1

answered May 19 '17 at 18:21
Neil T.
3,140 1 19 30

The class Renderer is DEPRECATED since Angular 4.3.0. angular.io/api/core/Renderer – Jamie Jul 17 '17 at 10:33

3 One can use Renderer2 angular.io/api/core/Renderer2 – theFreedomBanana Aug 16 '17 at 12:40

12

Create an `init` directive.

```
import {
    Directive,
    EventEmitter,
    Output,
    OnInit,
    ElementRef
} from '@angular/core';

@Directive({
    selector: '[init]'
})
export class InitDirective implements OnInit {
    constructor(private ref: ElementRef) {}

    @Output() init: EventEmitter<ElementRef> = new EventEmitter<ElementRef>();

    ngOnInit() {
        this.init.emit(this.ref);
    }
}
```

Export your component with a name such as `myComponent`

```
@Component({
    selector: 'wm-my-component',
    templateUrl: 'my-component.component.html',
    styleUrls: ['my-component.component.css'],
    exportAs: 'myComponent'
})
export class MyComponent { ... }
```

Use this template to get the `ElementRef` AND `MyComponent` instance

```
<div [ngSwitch]="type">
    <wm-my-component
           #myComponent="myComponent"
           *ngSwitchCase="Type.MyType"
           (init)="init($event, myComponent)">
    </wm-my-component>
```

```
init(myComponentRef: ElementRef, myComponent: MyComponent) {
}
```

answered Dec 29 '16 at 0:21
jsgoupil
2,217 1 25 38

10

import the `ViewChild` decorator from `@angular/core` , like so:

HTML Code:

```
<form #f="ngForm">
  ...
  ...
</form>
```

TS Code:

```
import { ViewChild } from '@angular/core';

class TemplateFormComponent {

  @ViewChild('f') myForm: any;
    .
    .
    .
}
```

now you can use 'myForm' object to access any element within it in the class.

Source

edited Jul 1 at 9:01
Prashant Pimpale
5,510 4 12 43

answered Nov 2 '17 at 11:01
Hany
493 8 14

2 Dont use any, the type is ElementRef – Johannes Dec 19 '17 at 14:25

8

```
 */
import {Component,ViewChild} from '@angular/core' /*Import View Child*/

@Component({
    selector:'display'
    template:`

     <input #myname (input) = "updateName(myname.value)"/>
     <p> My name : {{myName}}</p>

    `
})
export class DisplayComponent{
  @ViewChild('myname')inputTxt:ElementRef; /*create a view child*/

   myName: string;

    updateName: Function;
    constructor(){

        this.myName = "Aman";
        this.updateName = function(input: String){

            this.inputTxt.nativeElement.value=this.myName;

            /*assign to it the value*/
        };
    }
}
```

edited Oct 31 '17 at 8:22
user2025187
2,803 3 10 25

answered Jan 16 '17 at 14:18
Eng.Gabr
347 4 14

8 Please provide some explanation to this code. Simply code dumping without explanation is highly discouraged. – rayryeng Jan 16 '17 at 14:38

3

I would like to add that if you are using `ElementRef` , as recommended by all answers, then you will immediately encounter the problem that `ElementRef` has an awful type declaration that looks like

```
export declare class ElementRef {
  nativeElement: any;
}
```

this is stupid in a browser environment where nativeElement is an `HTMLElement` .

To workaround this you can use the following technique

```
import {Inject, ElementRef as ErrorProneElementRef} from '@angular/core';

interface ElementRef {
  nativeElement: HTMLElement;
}

@Component({...}) export class MyComponent {
  constructor(@Inject(ErrorProneElementRef) readonly elementRef: ElementRef) { }
}
```

answered Feb 11 '17 at 17:10
Aluan Haddad
14.5k 2 33 56

1 This explains a problem I was having. This doesn't work because it'll say `item` needs to be an ElementRef, even though you're setting it to another ElementRef: `let item:ElementRef, item2:ElementRef; item = item2; // no can do.` . Very confusing. But this is fine: `let item:ElementRef, item2:ElementRef; item = item2.nativeElement` because of the implementation you pointed out. – oooyaya Mar 5 '17 at 3:24

1 Actually your first example `let item: ElementRef, item2: ElementRef; item = item2` fails because of definite assignment analysis. Your second fails for the same reasons but both succeed if `item2` is initialized for the reasons discussed (or as a useful quick check for assignability we can use `declare let` here). Regardless, truly a shame to see `any` on a public API like this. – Aluan Haddad Mar 5 '17 at 23:29

1

```
event.source._elementRef.nativeElement.nextElementSibling
```

answered Oct 12 '17 at 12:49

Apoorv
598 1 7 15

1

Selecting target element from the list. It is easy to select particular element from the list of same elements.

component code:

```
export class AppComponent {
  title = 'app';

  listEvents = [
    {'name':'item1', 'class': ''}, {'name':'item2', 'class': ''},
    {'name':'item3', 'class': ''}, {'name':'item4', 'class': ''}
  ];

  selectElement(item: string, value: number) {
    console.log("item="+item+" value="+value);
    if(this.listEvents[value].class == "") {
      this.listEvents[value].class='selected';
    } else {
      this.listEvents[value].class= '';
    }
  }
}
```

html code:

```
<ul *ngFor="let event of listEvents; let i = index">
   <li  (click)="selectElement(event.name, i)" [class]="event.class">
  {{ event.name }}
</li>
```

css code:

```
    background:blue;
  }
```

edited Mar 2 '18 at 19:39
Stphane
2,862 1 23 34

answered Mar 2 '18 at 19:18
Sai Goud
41 3

protected by Günter Zöchbauer Jul 11 '18 at 12:25

Thank you for your interest in this question. Because it has attracted low-quality or spam answers that had to be removed, posting an answer now requires 10 reputation on this site (the association bonus does not count).

Would you like to answer one of these unanswered questions instead?