Ionic 2 - Angular 2 - Event 를 통한 컴포넌트 끼리 통신, Service 와 컴포넌트 통신

요약

컴포넌트 또는 Service 간의 통신하는 방법을 설명한다.

- Parent 에서 Child 로 통신하는 것은 비교적 쉽다.
- Child 에서 Parent 로 통신 할 때에는 본 장에서 설명하는 @OUTPUT() 를 통해서 Custom 이벤트를 전송하는 것이 좋다.
    - Ionic 을 사용하는 경우에는 Ionic 에서 별도의 event 방식을 제공하지만 그래도 가장 좋은 방법은 @Output() 을 통한 Custom event 전송인 것 같다.
- Parent 에서 Child 를 reference 로 바로 액세스 하는 방법.

참고 문서

Angular 2 공식 문서 :
https://angular.io/docs/ts/latest/cookbook/component-communication.html

인터넷 문서 : http://mean.expert/2016/05/21/angular-2-component-communication/

EventEmitter 에 대한 설명

Angular 의
EventEmitter 는 Observable 을 extends 하여 event 를
컴포넌트 또는 디렉티브로 이벤트를 보내고자 할 때 @Output() 과 함께 사용된다.
.emit() 을 호출하면 그 값을 .next() 로 전달하여 실제로 이벤트 호출이 발생한다.

공식문서: Observable in Angular 에서는
마치, 서비스에서는 EventEmitter 를 사용 할 수 없는 것처럼 설명을 하는데, 실제로 해 보니 서비스에서도 잘 된다.

실제로 개발자 사이에서 이것은 잘못된 것이며 이것을 사용하지 말기를 요청하는 issue 및 질문/답변 글을 작성하고 있다.

참고: https://github.com/mgechev/codelyzer/issues/457
참고:
https://stackoverflow.com/questions/36076700/what-is-the-proper-use-of-an-eventemitter#answer-36076701

따라서 서비스에서는 사용하지 않는다.

단, 아래와 같이 사용은 가능하다.

하지만 서비스에서는 EventEmitter 대신 Subject 와 같은 것을 사용한다.

참고: Angular 에서 Subject 사용하기

Custom event 을 만들고 fire 하기 위한 클래스이다.

- EventEmitter 와 Input, Output 을 통해서 컴포넌트간에 event 를 주고 받는다.

Inputs and Outputs

- Input 은 property 에 지정하는 것이며, 데이터를 입력(수산, 지정) 받을 수 있다.
- Input 의 property 는
    - @Input() isFavorite; 와 같이, 멤버 변수 앞에 @Input() decorator 를 둘 수 있는데, 이렇게 하면 자동으로 외부에서 볼 수 (public 멤버 변수가) 있게 된다.
- Output property 를 통해서 custom event 를 해당 Component 가 부모 Component 에게 메세지를 보낼 수 있다.

Input property 로 만들기 위해서는 input decorator 를 사용하면 된다.

- Decorator 는 import 으로 불러들이고 함수 처럼 사용하면 된다. (주의: 앞에 @ 을 붙이고 끝에 ; 을 붙이지 않는다.)

```
import {Component, Input, Output, EventEmitter} from '@angular/core';
@Component({
    selector: 'favorite',
    template: `
    <i [style.backgroundColor]="isFavorite ? 'blue' : 'gray'" (click)="onClick()">
        Favorite
    </i>
    `
})
export class FavoriteComponent {
    @Input() isFavorite: boolean; // @Input() 은 값을 받는다는 뜻.
    @Output() colorChange = new EventEmitter(); // @Output() 은 custom 이벤트를 만들고 emit() 을 통해 fire 할 것이라는 뜻.
    onClick() {
        this.isFavorite = !this.isFavorite;
        this.colorChange.emit({ newValue: this.isFavorite }); // onclick() 이 호출되고, isFavorite 가 변경되면, event 를 fire 한다.
    }
}
```

- Input property 를 Component decorator 에 집어 넣을 수 있는데, 이렇게 하는 경우, 변수명을 바꾸면 Component decorator 에도 바꾸어야하기 때문에 이중 일이 된다.

```
@Component({
   …
   Inputs: [‘isFavorite’]
   …
})
```

- @Input() 을 사용하는 경우 장점은 aliasing 을 할 수 있다는 것이다.
- @input(‘is-favorite’) isFavorite = false; 와 같이 하는 경우, 변수명이 바뀌어도 aliasing 을 사용하기 때문에 상관이 없다.
- @Component({ … inputs: [‘isFavorite:is-favorite’] … }) 와 같이 aliasing 을 할 수 있다.

부모 Component 예제)

```
import { Component } from '@angular/core';
import { FavoriteComponent } from "./f.component";
@Component({
  selector: 'my-app',
  template: `
  <favorite [isFavorite]="post.isFavorite"
(colorChange)="onFavoriteChange($event)"></favorite>
  `,
  directives: [FavoriteComponent]
})
export class AppComponent {
  post = {
    isFavorite: true
  }
  onFavoriteChange( $event ) {
    console.log($event);
  }
}
```

- 위 예제에서 자식 Component 가 fire 한 event - colorChange 를 받아서 핸들러로 연결해서 처리를 하고 있다.

서비스에서 EventEmitter 사용하기

매우 간단하다. 그냥 EventEmitter() 로 .emit() 하고, 그 변수를 subscribe 하면 된다.

예제) Service 에서 EventEmitter 를 사용하면 안된다.

```
export class Language {
  load: EventEmitter<any> = new EventEmitter();
  const re = ajax 데이터
  this.load.emit(re); // 서비스에서 그냥 emit() 을 하면 된다.
@Output() 필요 없다.
}
export class AppService {
  constructor( language: Language ) {
       language.load.subscribe( ln => {
           console.log('language load ln: ', ln);
       });
  }
```

단순 변수 또는 정보 공유

Service 를 만들고 provider 에 넣지 않고, 그냥 static 으로 사용하면 된다.

참고: Service 와 Providers 요약

간편한 Ionic 의 Event 방식 - Ionic 2 Events API

Ionic 2 의 Events API 는 매우 편리하다.

- Angular 2 에서 제공하는 Event Emit 방식은 조금 복잡하다.

Ionic 2 의 Event 장점은

- 컴포넌트간의 Dependency 가 없다는 것이다.
    - 즉, 아무 어디서든 (어떤 컴포넌트에서든) events.subscribe() 를 하고,
    - 어디서든 events.publish() 를 하면 된다.
    - 어떤 컴포넌트를 import 해야 하는 등의 걱정을 할 필요가 없다.

예를 들어 실재로 이런 경우가 발생했다.

- login.ts 에서 app-header.ts 를 import 하는데,
- ap-header.ts 에서 다시 login.ts 를 import 해서
- 재귀적으로 서로 import 하는 것이다.

이와 같은 경우 unknown directive “undefined” 등의 에러가 발생하는데,

재귀 import 할 필요 없이 event 로 하면 된다.

아래와 같이 그냥 막 하면 막 된다.

예제) 이벤트를 받는 쪽

```
import { Events } from 'ionic-angular';

constructor( private events: Events ) {

    events.subscribe('app', ( data ) => {
        console.log( data );
    })

}
```

예제) 이벤트를 보내는 쪽

```
import { Events } from 'ionic-angular';
```

```
constructor( private events: Events ) { .. }

this.events.publish('app', { hello:'How are you?', 'event-data': 'OK'} ); // 언제, 어디서든 ...
```

중요: 주의 할 점은

- subscribe(‘message’, callback((data) => …)) 에서

  data 는 객체이다. 따라서 필요하면 첫번째 항목만 사용한다.

- events.subscribe(‘event-message’, callback(…)) 에서 callback 안 쪽의 영역은 anonymous function scope 영역이라 해당 클래스를 액세스 하지 못한다.

이 때에는 아래와 같이 한다.

```
static _this: MyApp;
constructor( private events: Events ) {
    MyApp._this = this;
    events.subscribe('app', this.onEvents );
}
onEvents( events: any ) {
    let e = events[0];
    let a = MyApp._this;
```

- 현재 컴포넌트가 여러군데에서 사용하는데, 각 컴포넌트 객체가 가지는 상태가 서로 다른 경우, 하나의 컴포넌트 객체가 동일한 컴포넌트의 다른 객체로 이벤트를 전송할 수 있다.

  이 같은 경우 이벤트를 전송하는 컴포넌트와 수신하는 컴포넌트가 동일하게 된다.

Ionic 2 의 Events 에서 주의 할 점

- 한 번 Events.subscribe() 한 것은 페이지가 소멸되어도 (페이지가 stack 에서 사라져도) Events.subscribe() 할 때 등록한 handler 는 closure 와 비슷하게 앱이 종료되지 않는한 계속 살아 있다.
  즉, subscribe() 를 한번 했으면 두번 하지 않거나 반드시 unsubscribe() 를 해 주어야 하는 것이다.

- 그리고 ionic 2 문서의 설명에서 unsubscribe('topic', handler) 와 같이되어져 있는데, 두번째 항목은 handler 가 아닌 그냥 null 을 입력을 해야한다.
  2016년 11월 21일 현재, 공식 문서에 오류가 있다.

예제)

```
import { Component } from '@angular/core';
import { Events } from 'ionic-angular';
@Component({
    selector: 'child-cat',
    templateUrl: 'child-cat.html'
})
export class ChildCat {
    no: number = 0;
    messageHandler = this.message;
    constructor( private events: Events ) {}
    onClickNo() {
        this.no ++;
        if ( this.no % 2 ) {
            this.subscribe();
        }
        else {
            this.unsubscribe();
        }
    }
    message( re? ) {
            console.log( 'message:', re);
    }
    subscribe() {
        console.log('subscribe()');
        this.events.subscribe('hi', re => { this.messageHandler(re); } );
    }
    unsubscribe() {
        let re = this.events.unsubscribe('hi', null);
        if ( re ) console.log("success on unsubscribe()");
        else console.log("failed on unsubscribe()");
    }
    onFireEvent() {
        console.log('onFireEvent()');
        this.events.publish('hi', 'event fired...');
    }
}
```

Angular Eventemitter 방식을 통한 컴포넌트 간의 통신

참고)

- 이 방식은 비교적 번거롭다.
- 만약 Ionic 2 를 사용한다면, Ionic 2 Events API 를 사용한다.

참고: Dependency Injection 요약

컴포넌트 끼리 통신을 해야하는데, @Input() 과 @Output 이 안될 때가 있다.

@Input() 은 Property binding 을 할 때 사용하는 것으로

특정한 조건이 발생 할 때, 서로 연결되어 있지 않는 컴포넌트에서 특정 작업을 하고자 하는 경우, Event 를 통해 통신 할 수 있다.

참고: 공부를 한 것은 아닌데, 그냥 막 쓰니까 된다.

이벤트 전송은 아래와 같이 한다.

예제) 버튼을 클릭하면 onClickLanguage() 가 호출되는데, 이 때 메세지를 전송하는 것이다.

```
import { Subject }    from 'rxjs/Subject';

  static change = new Subject<string>();

  onClickLanguage( ln: string ) {
    this.db.set( 'language', ln );
    SettingPage.change.next('language-change');
  }
```

위와 같이 static 으로 하면 아래와 같이 받을 수 있다.

어느 컴포넌트에서든 상관없이 SettingPage 를 import 하고 SettingPage.change.subscribe() 를 통해서 이벤트를 받을 수 있다.

위에서 한번 event 를 발생하지만, 여러 곳에서 subscribe 하여 데이터를 수신 할 수 있다.

예제) app.ts 에서 subscribe

```
import { SettingPage } from './pages/setting/setting';
    SettingPage.change.subscribe( (x) => console.log("Subscription in app.ts : " + x) );
```

예제) login.ts 에서 subscribe

```
import { SettingPage } from '../setting/setting';
    SettingPage.change.subscribe( (x) => console.log("Subscription in LoginPage : " + x) );
```

아래와 같이 단순히 문자열 이벤트를 부모 클래스로 전송 할 수 있다.

예제) 서비스 클래스

```
import { Observable } from 'rxjs/Observable';
import { Subject }    from 'rxjs/Subject';

export class Language {
  ready = new Subject<string>();

  constructor( private translate: TranslateService, private db: Database ) {
    translate.setDefaultLang('en');
    this.db.get('language', (x) => {
      if ( x ) translate.use(x);
      else translate.use('en');
      this.ready.next('language-set');
    })
```

위에서는 this.db.get 을 통해서 database 액세를 하는데, 속도가 느려 Promise 를 통해서 값을 읽는다.

그리고 특정 액션을 취하는데, 그 액션 다음에 부모 클래스에서 어떤 동작이 일어나야 한다.

위에서는 service 에서 '언어'를 선택하면, 부모 클래스에서 그 언어에 맞는 문장을 보여주는 것이다.

즉, service 에서 특정 작업을 해야지만 부모 클래스에서 추가 작업을 할 수 있는데, service 에서 준비가 되었다는 표시를 event-emitting 을 통해서 한다.

```
ready = new Subject<string>();
```

와 같이 하고,

```
this.ready.next('...');
```

와 같이 하면 해당 문자열이 부모에게 전달된다.

부모 클래스에서는 아래와 같이 문장을 받으면 된다.

예제)

```
constructor(private language: Language) {
  this.language.ready.subscribe( (x) => console.log("app.ts : " + x) );
```

위에서 염려 할 것은,

- 자식 클래스(service)가 동작이 느릴 것이라고 판단했는데,
- 막상 자식클래스가 동작이 빨라서,
- 부모 클래스에서 subscribe() 하기도 전에
    - 자식 클래스(service)에서 작업이 끝나 버리고,
    - 부모 클래스가 subscribe() 하기 전에 자식 클래스에서 Subject().next() 로 이벤트를 전송해 버리면,
- 부모 클래스에서 자식 클래스의 event 를 받지 못하는 경우가 발생할 수 있다.

따라서 확실히 하기 위해서, 자식 (service) 클래스에서 이벤트를 전송 할 때, 아래와 같이 시간차를 둔다.

이렇게 하면 한 층 더 안전하게, 자식 클래스 작업이 끝난 다음에 부모 클래스에서 호출이 된다.

예제)

```
setTimeout( () => {
        this.ready.next('language-set');
    }, 200 );
```

Event Emit 통신의 문제점

Event emitting 을 통해서 작업하는 것이 쉽고 편리하지만, 매우 복잡해 진다.

따라서 Ionic 2 를 사용한다면, Ionic 2 Event API 를 사용한다.

특히 event 가 앱 시작 후 나중에 발생하고,

한번 event subscribe() 한 것은 반드시 unsubscribe() 를 해 주어야 한다.

특히, view 를 새로 열때마다 새로운 객체를 생성하고 dependency injection 을 하고 constructor() 를 호출 하는데, view 의 객체가 무한적으로 생성되는데, view 에서 subscribe() 했으면 반드시 unsubscribe() 를 해 주어야 한다.

참고 : Dependency Injection 에 잘 설명을 해 놓았다.

Custom Event - 이벤트 직접 제작 - @Output - 중요

- 매우 중요 : 암기 할 것
  - Child Component 에서
    - Output 과 EventEmitter 를 @angular/core 로 부터 import 하고
    - @Output('name') 이벤트변수명 = new EventEmitter() 와 같이 하고
    - 필요한 때에 this.변수명.emit(전달 할 값) 으로 호출하며 된다.
  - Parent Component 에서는 DOM element 에 event 명을 지정하고, 이벤트 핸들러를 지정한다.
    - 예) <search-box (이벤트변수명)="이벤트 핸들링 함수()" ...>

- 참고. @Output() 은 컴포넌트로 전달하는 경우, 즉, template output property 로 사용하는 경우에만 필요하다.

- 주의 해야 할 점
  - 능숙하지 않을 때, 문법 에러가 간간히 발생한다. 자주 발생하는 에러는
    - 이벤트 핸들링 함수 끝에 괄호 "()" 를 붙이지 않는 것이다.

예를 들어, 사용자가 입력 창에서 검색어를 입력하고 엔터 키를 누르면 검색 하고자 할 때, 엔터키 이벤트를 받아야 하는데, Angular 2 에는 (keydown.enter) 라는 이벤트가 있다.

하지만 문제는 이 입력 창을 가지고 있는 컴포넌트가 자식 컴포넌트이고, 검색 루틴은 부모 컴포넌트에 있다는 것이다.

Angular 2 에서는 부모와 자식이 통신을 하는 수단으로 @Input, @Output 을 사용하고 있다. 물론 이 외에도 다른 방법이 있지만 가장 많이 쓰이는 것이 바로 @Input, @Output 이다.

- 자식 컴포넌트에서 입력창에 검색어를 입력하고 엔터키를 눌렀을 때,

- 부모 컴포넌트에서 "search" 라는 이벤트를 받아 들이도록 해 보자.

예제) 부모 컴포넌트에서는 아래와 같이 이벤트를 받는다.

```
<search-box (search)="onSearch( $event )"></search-box>
```

참고: 위에서 $event 변수명은 고정이다. 이 $event 변수에는 해당 이벤트에서 전달하는 값이 들어가 있다. 하지만 onSearch() 함수에서 받을 때에는 변수명을 마음데로 지정해도 된다.

예제) 자식 컴포넌트에서 이벤트를 부모 컴포넌트로 전송

```
import { Component, Output, EventEmitter } from '@angular/core';
@Component({
   selector: 'search-box',
   template: `
      <input #box (keydown.enter)="onEnter( box.value )" placeholder="Input
Keyword">
   `
})
export class SearchBox {
   @Output() search = new EventEmitter();
   onEnter( value ) {
      this.search.emit( value );
   }
}
```

위 에서 <search-box (search)="....”> 의 “search” 가 바로 이벤트 명이 된다.
이것은 자식 컴포넌트의 search property 를 @Output 으로 decorate 했기 때문에 search 가 이벤트 명으로 사용 가능 한 것이다.

예제) 부모 컴포넌트 클래스

```
import { SearchBox } from '../../templates/search';
@Component({
  directives: [ SearchBox ],
});

  onSearch( $eventData ) {
    console.log($eventData);
  }
```

참고: 위 에서 event 를 받을 때, template expression 에서는 $event 가 고정이지만, 함수에서는 변수를 마음데로 해도 된다.

다음은 간추린 예제이다.

예제) Child Component 에서 해야 할 것.

```
import { Component, Output, EventEmitter } from '@angular/core';
@Output() cancel : EventEmitter<RegisterTemplate> = new
EventEmitter<RegisterTemplate>();
this.cancel.emit(this);
```

예제) 부모 Component 에서 해야 할 것

```
<xapi-register #Register (cancel)="onCancel($event)"></xapi-register>
```

Input or output?

원문)

Input properties usually receive data values. *Output* properties expose event producers, such as `EventEmitter` objects.

The terms *input* and *output* reflect the perspective of the target directive.

`HeroDetailComponent.hero` is an **input** property from the perspective of `HeroDetailComponent` because data flows *into* that property from a template binding expression.

`HeroDetailComponent.deleteRequest` is an **output**property from the perspective of `HeroDetailComponent` because events stream *out* of that property and toward the handler in a template binding statement.

Aliasing input/output properties

원문)

Sometimes we want the public name of an input/output property to be different from the internal name.

This is frequently the case with attribute directives. Directive consumers expect to bind to the name of the directive. For example, when we apply a directive with a `myClick` selector to a `<div>` tag, we expect to bind to an event property that is also called `myClick`.

```
<div (myClick)="clickMessage=$event">click with myClick</div>
```

However, the directive name is often a poor choice for the name of a property within the directive class. The directive name rarely describes what the property does. The`myClick` directive name is not a good name for a property that emits click messages.

Fortunately, we can have a public name for the property that meets conventional expectations, while using a different name internally. In the example immediately above, we are actually binding *through the* `myClick` *alias*to the directive's own `clicks` property.

We can specify the alias for the property name by passing it into the input/output decorator like this:

```
@Output('myClick') clicks = new EventEmitter<string>(); //  @Output(alias) propertyName = ...
```

We can also alias property names in the `inputs` and`outputs` arrays. We write a colon-delimited (`:`) string with the directive property name on the *left* and the public alias on the *right*:

```
@Directive({
  outputs: ['clicks:myClick']  // propertyName:alias
})
```

Parent 에서 Child 를 직접 참조

ViewChild 를 통해서 Parent 에서 직접 Child 의 속성을 참조 할 수 있다.

변수 값 지정이나 메소드 호출 등이 가능하다.

중요 : @ViewChild 사용법 요약 - Ionic 2 의 Root Component 에서 @ViewChild 참고

참고 : Angular 2 공식 문서 Cookbook 의 Component iteration 을 참고한다.
웹 사이트 : http://mean.expert/2016/05/21/angular-2-component-communication/ 의 View Template Reference 와 View Child 를 살펴본다.

아래의 예제 처럼 특정 (컴포넌트) DOM element 를 "#변수"와 같이 참조 할 수 있는데,

이 참조 변수를 @ViewChild 로 하여 해당 Child Component 의 변수와 메소드를 직접 사용할 수 있다.

다만, constructor() 에서는 사용하지 못하고 view 가 완전히 rendering 이 된 다음에 사용 가능하다.

예제)

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

```
import { NavController } from 'ionic-angular';
import { AppHeader } from '../../xapi/template/app-header';
@Component({
  template: `
    <ion-header>
      <xapi-header #header></xapi-header>
    </ion-header>
    <ion-content>
      <h2>Content</h2>
    </ion-content>
  `,
  directives: [ AppHeader ]
})
export class HomePage {
  @ViewChild('header') h: AppHeader;
  constructor(public navCtrl: NavController) {
  }
  ngAfterViewInit() {
    this.h.appTitle = "Title ABC";
    this.h.func();
  }
}
```

Child 에서 Parent 를 참조 - 자식이 부모 함수 호출

- Angular 에서 지원하지는 않지만, 충분히 가능하다..

예제) 부모 클래스

```
get self(): WritePage {
    return this;
}
hookPostCreate() {
    let content = this.tiny.editor.getContent();
    this.postCreateComponent.post_content = content;
}
```

- 부모 클래스를 template 에서 this 로 참조 할 수 없으므로 self 라는 getter 를 만들어서
  자식 클래스로 넘겨준다.

예제) 부모 클래스 template

```
<post-create-component #postCreateComponent
[parent] = 'self'
(success)="onPostCreatSuccess($event)"
[category]="'adwriter'"
[caption]="'광고 글 쓰기'"></post-create-component>
```

예제) 자식 클래스

- 필요 할 때, 부모 클래스를 참조하면 된다.
- 부모 클래스에서 다시 자식 클래스를 ViewChild 로 정보 변경을 할 수 있다.

```
@Input() parent;
if ( this.parent && this.parent.hookPostCreate ) {
   this.parent.hookPostCreate();
}
```

예제) 부모 클래스

- 아래의 예제는 글쓰기 폼에서 기존의 content textarea 를 숨기고 tinyMce 를 보여주고 글 쓰기 버튼을 클릭하면 tinyMce 내용을 원래 content textarea 에 넣어주는 것이다.
- 부모에서 자식(글쓰기 폼)을 로드하고,
- (자식 클래스에서 글을 써야하는데,) 부모에서 글을 쓰고,
- 자식 클래스에서 글 쓰기 버튼을 클릭하면,
- 부모 클래스의 TinyMCE 의 글 내용을 자식 클래스로 전송하고
- 자식 클래스가 글을 쓴다.

```
@ViewChild('postCreateComponent') postCreateComponent: PostCreateComponent;
hookPostCreate() {
   let content = this.tiny.editor.getContent();
   this.postCreateComponent.post_content = content;
}
```