<http://xgrommx.github.io/rx-book/index.html>

// https://medium.com/@tarik.nzl/creating-a-canvas-component-with-free-hand-drawing-with-rxjs-and-angular-61279f577415

// typescript observable subscribe example

// https://xgrommx.github.io/rx-book/content/getting\_started\_with\_rxjs/creating\_and\_querying\_observable\_sequences/creating\_and\_subscribing\_to\_simple\_observable\_sequences.html

//https://wonism.github.io/rxjs-5/

//https://angular-2-training-book.rangle.io/handout/observables/using\_observables.html

//https://github.com/Reactive-Extensions/RxJS/blob/master/doc/gettingstarted/creating.md

//https://github.com/Reactive-Extensions/RxJS/tree/master/doc/api/core/operators

이벤트처리

|  |
| --- |
| Observable.*fromEvent*(**this**.canvas, 'mousedown').subscribe((event: MouseEvent)=>{  **if**(**this**.manager)**this**.manager.mousedown(event); }); Observable.*fromEvent*(**this**.canvas, 'mouseup').subscribe((event: MouseEvent)=>{  **if**(**this**.manager)**this**.manager.mouseup(event); }); Observable.*fromEvent*(**this**.canvas, 'mousemove').subscribe((event: MouseEvent)=>{  **if**(**this**.manager)**this**.manager.mousemove(event); }); Observable.*fromEvent*(**this**.canvas, 'keydown').subscribe((event: KeyboardEvent)=>{  **if**(**this**.manager)**this**.manager.keydown(event); }); Observable.*fromEvent*(**this**.canvas, 'keyup').subscribe((event: MouseEvent)=>{  **if**(**this**.manager)**this**.manager.keyup(event); }); |

zip으로 여러 Observable 합치기

|  |
| --- |
| **const** droneImage = **new** Image() **as** HTMLImageElement;  droneImage.src = "assets/image/drone.png"; // can also be a remote URL e.g. http://  **let** dronObservable: Observable<Event> = Observable.*fromEvent*(droneImage,'load');  dronObservable.subscribe((e: Event)=>{ctxBuffer.drawImage(droneImage,x - droneImage.width/2, y - droneImage.height/2);});  //  **const** groundImage = **new** Image() **as** HTMLImageElement;  groundImage.src = "assets/image/ground.png"; // can also be a remote URL e.g. http://  **let** groundObservable: Observable<Event> = Observable.*fromEvent*(groundImage,'load');  groundObservable.subscribe((e: Event)=>{ctxBuffer.drawImage(groundImage,x - groundImage.width/2, **this**.canvas.height - groundImage.height);});  //  **const** cloudImage = **new** Image() **as** HTMLImageElement;  cloudImage.src = "assets/image/cloud.png"; // can also be a remote URL e.g. http://  **let** cloudObservable: Observable<Event> = Observable.*fromEvent*(groundImage,'load');  cloudObservable.subscribe((e: Event)=>{ctxBuffer.drawImage(cloudImage,x - cloudImage.width/2, 0);});  //  **let** source = Observable.zip(  dronObservable,  groundObservable,  cloudObservable,  **function** (s1, s2, s3) {  **return** s1 + ':' + s2 + ':' + s3;  }  );  //  **let** subscription = source.subscribe(x=>{  // ctxBuffer.clearRect(0, 0, this.canvas.width, this.canvas.height);  context.clearRect(0, 0, **this**.canvas.width, **this**.canvas.height);  context.drawImage(cnvsBuffer,0,0);  console.log('Next: ' + x);  },  err=>{  console.log('Error: ' + err);  },  ()=> {  context.drawImage(cnvsBuffer,0,0);  });  context.drawImage(cnvsBuffer,0,0); }  http://reactivex.io/documentation/operators/zip.html |

from filter map

|  |
| --- |
| ngOnInit() {  **this**.i18nService.subscribe(i18Data => **this**.changeMenuPr(i18Data.lang), err=>console.error(err));   **this**.userService.getLoginInfo().subscribe(user => {  // this.changeDetector.detach();  **let** targetAuths: Array<AuthMenu>;  Observable.from(user.auths)  .filter(auth=>'Y'!=auth.hddnYn && 'Y'==auth.useYn && 'GET'==auth.crudTypeCd)  .map(auth=>{  **let** authMenu:AuthMenu = **new** AuthMenu();  authMenu.urlSeq = auth.urlSeq ;  authMenu.menuNm = auth.menuNm ;  authMenu.menuNmEn = auth.menuNmEn ;  authMenu.menuLvl = auth.menuLvl ;  authMenu.url = auth.url ;  authMenu.menuIcon = auth.menuIcon ;  authMenu.prntUrlSeq = auth.prntUrlSeq;  **return** authMenu;  }).toArray().subscribe(auths=>targetAuths=auths);   Observable.from(targetAuths)  .filter(auth=>auth.menuLvl==1)  .map(auth=>{  Observable.from(targetAuths)  .filter(subAuth=>subAuth.menuLvl==2 && auth.urlSeq==subAuth.prntUrlSeq)  .map(subAuth=>subAuth **as** AuthMenu)  .toArray()  .subscribe(subAuths=>auth.auths=subAuths)  **return** auth;  })  .toArray()  .subscribe(auths => **this**.auths = auths, console.error);   console.log(**this**.auths);  }) |

<http://xgrommx.github.io/rx-book/content/observable/observable_methods/fromevent.html>

|  |
| --- |
| emitter이용 data전송  var eventEmitter = new EventEmitter();  var source = Rx.Observable.fromEvent(  eventEmitter,  'data',  function (args) {  return { foo: args[0], bar: args[1] };  });  var subscription = source.subscribe(  function (x) {  console.log('Next: foo -' x.foo + ', bar -' + x.bar);  },  function (err) {  console.log('Error: ' + err);  },  function () {  console.log('Completed');  });  eventEmitter.emit('data', 'baz', 'quux');  // => Next: foo - baz, bar - quux |

careate

|  |
| --- |
| **const** windObservable = Observable.*create*((subscriber: Subscriber<**any**>) => {  console.log('wwwwwwwwwww'); }); |

subject를 이용하여

|  |
| --- |
| **this**.\_webSocket = **new** WebSocket('ws://localhost:8999'); **const** observable = Observable.*create*(  (obs: Observer<MessageEvent>) => {  **this**.\_webSocket.onmessage = obs.next.bind(obs);  **this**.\_webSocket.onerror = obs.error.bind(obs);  **this**.\_webSocket.onclose = obs.complete.bind(obs);  **return this**.\_webSocket.close.bind(**this**.\_webSocket);  }); **const** observer = {  next: (data: **any**) => {  **if** (**this**.\_webSocket.readyState === WebSocket.OPEN) {  **this**.\_webSocket.send(JSON.stringify(data));  }  },  error : (error: **any**) => {  console.log('error websocket');  }, }; **this**.\_webScoketSubject = Subject.*create*(observer **as** Observer<**any**>, observable).map((response: MessageEvent): Telegram<**any**> => {  **const** data = JSON.parse(response.data) **as** Telegram<**any**>;  **return** data; }); |
| DroneStageManager.*getInstance*().webScoketSubject.subscribe((telegram) => {  console.log('telegram ' + telegram); }); |
| context.fillText('(시작하기)(' + WebSocket.CLOSED + ', ' + WebSocket.OPEN + ', ' + WebSocket.CLOSING + ', ' + WebSocket.CONNECTING + ')' + DroneStageManager.*getInstance*().webSocket.readyState, x, y + 30); |

subject이해 , bind

|  |
| --- |
| **import** 'rxjs/operator/map'; **import** 'rxjs/add/operator/map'; **import** {Observable} **from** 'rxjs/Observable'; **import** {Observer} **from** 'rxjs/Observer'; **import** {Subject} **from** 'rxjs/Subject';  **class** A {  **public** subject: Subject<**any**>;   **public** obs: Observer<**any**>;   **constructor**() {  **const** observable = Observable.*create*((obs: Observer<**any**>) => {  **this**.obs = obs;  **this**.next = obs.next.bind(obs);  **this**.onerror = obs.error.bind(obs);  **this**.onclose = obs.complete.bind(obs);  // this.\_webSocket.onclose = obs.complete.bind(obs);  });   **const** observer = {  next: (data: **any**) => {  console.log('observer next ' + data);  **this**.obs.next(data);  },  error : (error: **any**) => {  console.log('observer error ' + error);  },  complete : () => {  console.log('observer complete');  },  };   **this**.subject = Subject.*create*(observer **as** Observer<**any**>, observable);  // this.subject = Subject.create(observer as Observer<any>, observable).map((response: MessageEvent): any => {  // console.log('in subject ' + response);  // // const data = response.data;  // // return data;  // });  }   main(): **string** {  console.log('welcome');  **return** '---';  }  next(name: **string**): **string** {  console.log('next' + name);  **return** 'next' + name;  }  onerror(): **string** {  console.log('onerror');  **return** 'onerror';  }  onclose(): **string** {  console.log('onclose');  **return** 'onclose';  } }  **const** a = **new** A();  **const** subScription = a.subject.subscribe((it) => { console.log('\*\*' + it); }); a.next('aaaa22a');  a.subject.next('nenene');  subScription.unsubscribe(); subScription.unsubscribe(); a.subject.complete(); a.subject.unsubscribe();  \*\*aaaa22a  observer next nenene  \*\*nenene  observer complete |

---------

1. Observable를 통해서 만들어서 모듈 관찰하는건 Observer 이다
2. Observer 인터페이스를 구현한체는 Subscriber이다.
3. Observable.carete하면 Observable 리턴된다. Observable을 구현체는 Subject이다.
4. 거기에 subscribe를 하여 데이터를 받는다. 추가하면 Subscription 를 리턴한다.

|  |
| --- |
| **export class** Subscriber<T> **extends** Subscription **implements** Observer<T> {  ------  **export interface** Observer<T> {  closed?: **boolean**;  next: (value: T) => **void**;  error: (err: **any**) => **void**;  complete: () => **void**; }  **export declare class** Subject<T> **extends** Observable<T> **implements** ISubscription { |

|  |
| --- |
| **import** {Observable} **from** 'rxjs/Observable'; **import** {Observer} **from** 'rxjs/Observer'; **import** 'rxjs/operator/map'; **import** {Subject} **from** 'rxjs/Subject';  **let** subject: Subject<**any**>;  **let** obs: Observer<**any**>;  **const** observable = Observable.*create*((obss: Observer<**any**>) => {  obs = obss; });  **const** observer = {  next: (data: **any**) => {  console.log('observer next ' + data);  obs.next(data);  },  error : (error: **any**) => {  console.log('observer error ' + error);  },  complete : () => {  console.log('observer complete');  }, };  subject = Subject.*create*(observer **as** Observer<**any**>, observable); // this.subject = Subject.create(observer as Observer<any>, observable).map((response: MessageEvent): any => { // console.log('in subject ' + response); // // const data = response.data; // // return data; // });    **const** subScription = subject.subscribe((it) => { console.log('\*\*' + it); }); // a.next('aaaa22a');  subject.next('nenene'); |
| // import 'rxjs/add/operator/map'; **import** {Observable} **from** 'rxjs/Observable'; **import** {Observer} **from** 'rxjs/Observer'; **import** 'rxjs/operator/map'; **import** {Subject} **from** 'rxjs/Subject'; **import** {Subscriber} **from** 'rxjs/Subscriber';  **let** fullSubscriber; **const** o: Observable = Observable.*create*( (obs: Subscriber<**any**>) => {  fullSubscriber = obs;  obs.next('555');  });  **const** s: Subscriber = o.subscribe( (it) => {  console.log(it + ' aaa'); }); s.next('sss');   **const** b: Subscription = o.subscribe( (it) => {  console.log(it + ' bbb'); });  console.log(fullSubscriber === s) // console.log(s === b) // console.log(fullSubscriber === b)   fullSubscriber.next('ffff') |

Subject

BehaviorSubject

|  |
| --- |
| // import 'rxjs/add/operator/map'; **import** 'rxjs/add/operator/filter'; **import** {BehaviorSubject} **from** 'rxjs/BehaviorSubject'; **import** {Observable} **from** 'rxjs/Observable'; **import** {interval} **from** 'rxjs/observable/interval'; **import** {Observer} **from** 'rxjs/Observer'; **import** 'rxjs/operator/map'; **import** {Subject} **from** 'rxjs/Subject';  **var** subject = **new** BehaviorSubject(42); // var subject = new BehaviorSubject(42)  **var** subscription = subject.subscribe(  **function** (x) {  console.log('Next1: ' + x.toString());  },  **function** (err) {  console.log('Error1: ' + err);  },  **function** () {  console.log('Completed1');  });    **var** subscription2 = subject.subscribe(  **function** (x) {  console.log('Next2: ' + x.toString());  },  **function** (err) {  console.log('Error2: ' + err);  },  **function** () {  console.log('Completed2');  }); // subject.filter  **var** subscription3 = subject.filter( (it) => it > 50).subscribe(  **function** (x) {  console.log('Next3: ' + x.toString());  },  **function** (err) {  console.log('Error3: ' + err);  },  **function** () {  console.log('Completed3');  });  // => Next: 42  subscription2.unsubscribe(); subject.next(56); // => Next: 56  subject.complete(); // => Completed  Next1: 42  Next2: 42  Next1: 56  Next3: 56  Completed1  Completed3 |

ReactiveX

Zip : <http://reactivex.io/documentation/operators/zip.html>

Merge : <http://reactivex.io/documentation/operators/merge.html>

Join : <http://reactivex.io/documentation/ko/operators/join.html>

|  |
| --- |
| **import** 'rxjs/add/observable/interval';  **import** 'rxjs/add/observable/combineLatest';  **const** xs = Observable.*interval*(100)  .map((x) => 'first' + x);  **const** ys = Observable.*interval*(100)  .map((x) => 'second' + x); Observable.combineLatest(xs, ys).subscribe( (it) => {  console.log(it); }); |

switchMap : <http://reactivex.io/documentation/operators/flatmap.html>

<http://www.syntaxsuccess.com/viewarticle/combining-multiple-rxjs-streams-in-angular-2.0>

<http://xgrommx.github.io/rx-book/content/observable/observable_instance_methods/join.html>

Observable결과쪽에서

또다른 Observable 결과를 이어서 넘겨주고싶을때

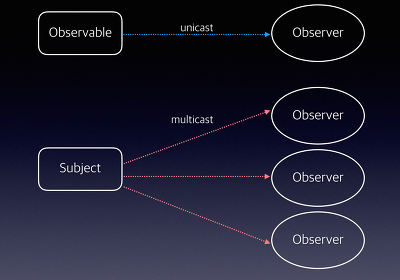
mergeMap을 사용한다

|  |
| --- |
| **return this**.http.post<HttpResponse<**any**>>(  **this**.apiUrl + `/${apiCode}`,  body,  {headers: **new** HttpHeaders().set(**this**.API\_HEADER\_NAME, JSON.stringify(head)), observe: 'response'} ).**mergeMap**((it: HttpResponse<**any**>) => {  **const** responseHead = Object.assign(**new** ResponseHead(), JSON.parse(it.headers.get(**this**.API\_HEADER\_NAME)) **as** ResponseHead);  console.log(responseHead.result\_code);  // if (ApiHeadResultCode.CODE\_SUCCESS !== responseHead.result\_code) {  **if** (ApiHeadResultCode.*CODE\_SUCCESS* === responseHead.result\_code) {  // let a = new ({ new (): T; })();  // let a = new () => T;  // let aa = new testType();  // console.log(aa);  // this.storage.set(apiCode, it.body).then(ii => {  // console.log(ii);  // });  // return from(this.storage.set(apiCode, it.body));  // return fromPromise(this.storage.set(apiCode, it.body));  return this.storage.get(apiCode);//promise  } **else** {  // it.status = 200;  // throw new Error('Valid token not returned');  **throw** responseHead;  }  // return Object.assign(new T(), it.body); }).catch((err: RequestHead | HttpErrorResponse) => {  console.log('eee ' + err);   // if (err instanceof HttpResponse) {  // // err.  // }  **if** (err **instanceof** HttpErrorResponse) {  **return this**.storage.get(apiCode);  } **else** {  **throw** err;  }  // throw new Error('Valid token notsssssss returned');  // return ['asd'];  // return this.storage.get(apiCode); }); |

# RxJs Subject 종류 알아보기

***Subject : Observable + Observer***

Subject는 observable과 observer의 역할을 모두 할 수 있는 bridge/proxy Observable이라 생각하면 됩니다. 그렇기 때문에 Observable이나 Subject 모두 Subscribe를 할 수 있습니다. 다만 subscribe의 차이가 있다면 Subject는 multicast방식이기 때문에 여러개의 observer를 subscribe할 수 있습니다. 단순 observable은 unicast방식이기 때문에 observer하나만을 subscribe할 수 있습니다. 아래 그림을 보시면 이해가 될 것입니다.



### Kinds of Subject

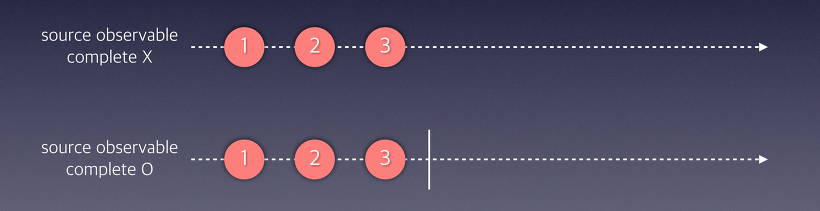
Subject의 종류에는 3가지가 있습니다.

* + **ReplaySubject** : 버퍼 사이즈에 따라 받을 수 있는 갯수가 다름. observable의 complete or error에 상관없음.
  + **AsyncSubject** : 무조건 complete되고 난 뒤 source observable의 마지막 데이터를 받는다.
  + **BehaviorSubject** : source observable의 가장 마지막 데이터 소스를 받는다. 단, complete 또는 error되면 받을 수 없다.

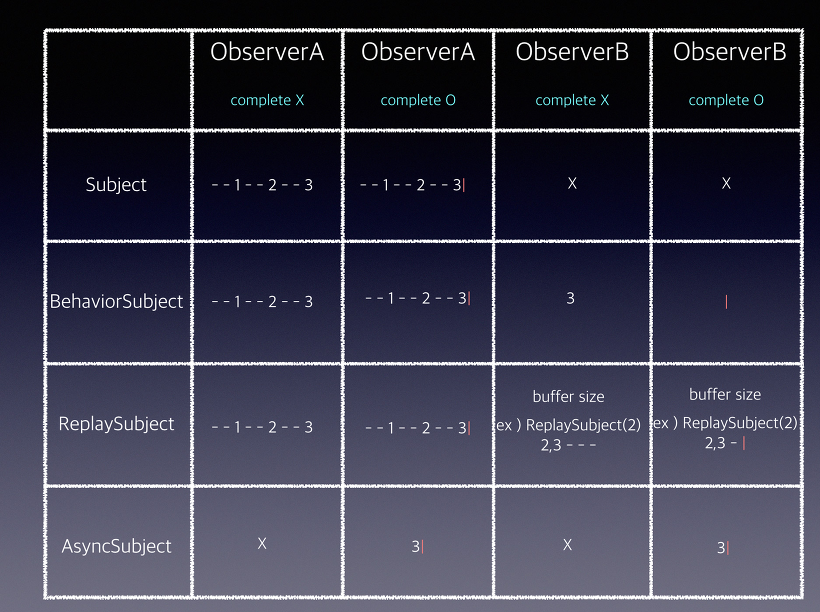
위 3개가 subject의 종류 입니다. 기본적으로 subject는 observable + observer의 역할을 동시에 가지고 있다고 위에서 설명하였습니다. subject의 종류를 나누는 기준은 subject를 통해 전해지는 데이터를 어떤식으로 받을 것 인지에 대한 기준으로 나뉘게 됩니다. 그럼 아래 그림을 통해 상황별 어떻게 데이터를 받게 되는지 알아보도록 하겠습니다.

**source code**

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | // var subject = new Rx.Subject();  // var subject = new Rx.BehaviorSubject(0);  // var subject = new Rx.ReplaySubject(2);  var subject = new Rx.AsyncSubject();    var observerA = {    next: function (x) { console.log('A next ' + x); },    error: function (err) { console.log('A error ' + err); },    complete: function () { console.log('A done'); },  };    subject.subscribe(observerA);    var observerB = {    next: function (x) { console.log('B next ' + x); },    error: function (err) { console.log('B error ' + err); },    complete: function () { console.log('B done'); },  };    setTimeout(function () {    subject.subscribe(observerB);  }, 2000);    subject.next(1);  subject.next(2);  subject.next(3);  // subject.complete();    [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |



위와같은 source observable이 있다고 가정하겠습니다. 위에것은 source observable이 complete되지 않은 상태이며 아래 스트림은 3까지 데이터를 발생시킨 뒤 complete() 메서드를통해서 종료됨을 나타내고 있습니다. 이제 이 source observable에 2개의 observer를 등록시켜 steam을 share하여 데이터를 받아보도록 하겠습니다. 2개의 observerA, observerB가 있는대 observerA는 subscribe을 바로 하고 observerB는 setTimeout을 통해 2초 뒤 데이터를 받는 상황으로 되어있는 코드를 확인하고 아래 도표를 보면 되겠습니다.



간단한듯하면서도 매우 중요한 subject는 기능별 확인을하고 상황에 맞게 잘 사용하면 좋을 것 같습니다. 예를들어 BehaviorSubject 같은 경우는 채팅창에 새로운 메시지가 온경우 제일 마지막 메시지를 꺼내서 보내주는 상황에 잘 어울리니 이 BehaviorSubject를 사용하면 좋을 것 입니다.

출처: <http://arnoldyoo.tistory.com/18> [Arnold's life]

# 주제

주제(subject)는 옵저버나 Observable처럼 행동하는 ReactiveX의 일부 구현체에서 사용 가능한 일종의 교각 혹은 프록시라고 볼 수 있는데, 그 이유는 주제는 옵저버이기 때문에 하나 이상의 Observable을 구독할 수 있으며 동시에 Observable이기도 하기 때문에 항목들을 하나 하나 거치면서 재배출하고 관찰하며 새로운 항목들을 배출할 수도 있다.

하나의 주제는 하나의 Observable을 구독하면서, (Observable이 "차가운" Observable인 경우 즉, 옵저버가 구독을 시작하기 전까지 항목들의 배출을 지연시키는 Observable일 경우)Observable이 항목들을 배출시키도록 동작시킨다. 그 결과로 인해 원래는 "차가운" Observable이었던 주제를 "뜨거운" Observable로 만들기도 한다.

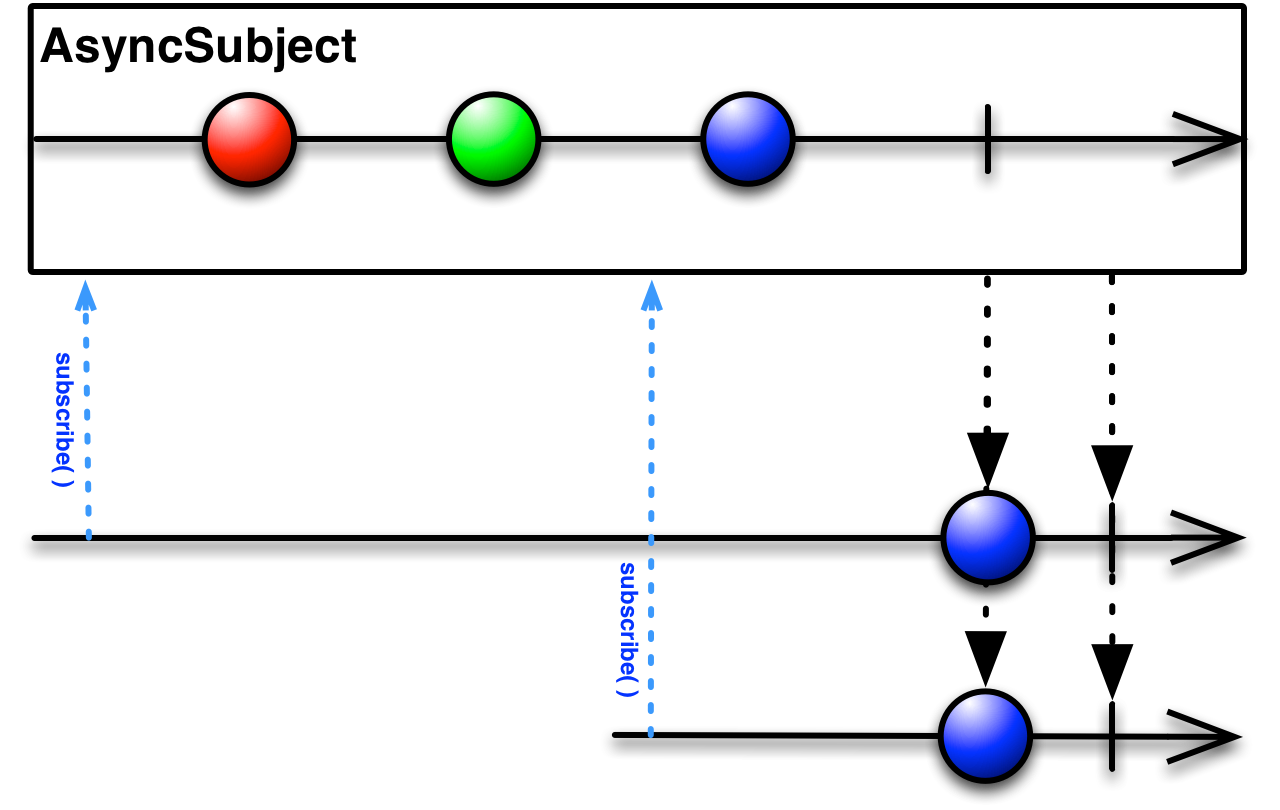
#### 참고

* [To Use or Not to Use Subject](http://davesexton.com/blog/post/To-Use-Subject-Or-Not-To-Use-Subject.aspx) from Dave Sexton’s blog
* [Introduction to Rx: Subject](http://www.introtorx.com/Content/v1.0.10621.0/02_KeyTypes.html#Subject)
* [101 Rx Samples: ISubject<T> and ISubject<T1,T2>](http://rxwiki.wikidot.com/101samples#toc44)
* [Advanced RxJava: Subject](http://akarnokd.blogspot.hu/2015/06/subjects-part-1.html) by Dávid Karnok
* [Using Subjects](http://xgrommx.github.io/rx-book/content/getting_started_with_rxjs/subjects.html) by Dennis Stoyanov

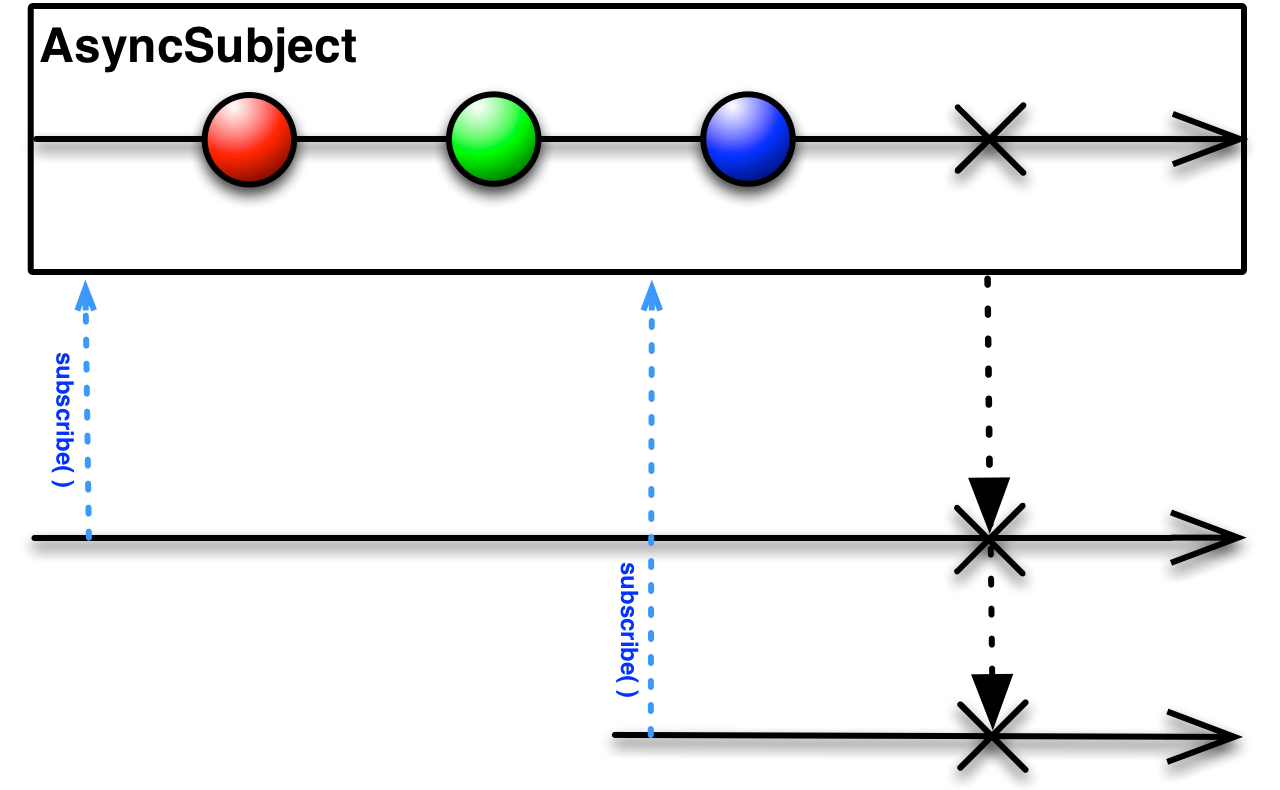
## **주제의 종류**

모두 4 종류의 Subject가 존재하며, 각각의 Subject는 특정 상황에 맞도록 설계되었다. 그렇기 때문에 모든 상황에서 아무 주제(Subject)를 임의대로 사용할 수 없으며 일부 구현체는 주제를 다른 이름으로 부르기도 한다(예를 들어, RxScala는 "주제(Subject)"를 "발행주제(PublishSubject)"로 부른다).

### AysncSubject



AsyncSubject는 소스 Observable로부터 배출된 마지막 값(만)을 배출하고 소스 Observalbe의 동작이 완료된 후에야 동작한다. (만약, 소스 Observable이 아무 값도 배출하지 않으면 AsyncSubject 역시 아무 값도 배출하지 않는다.)

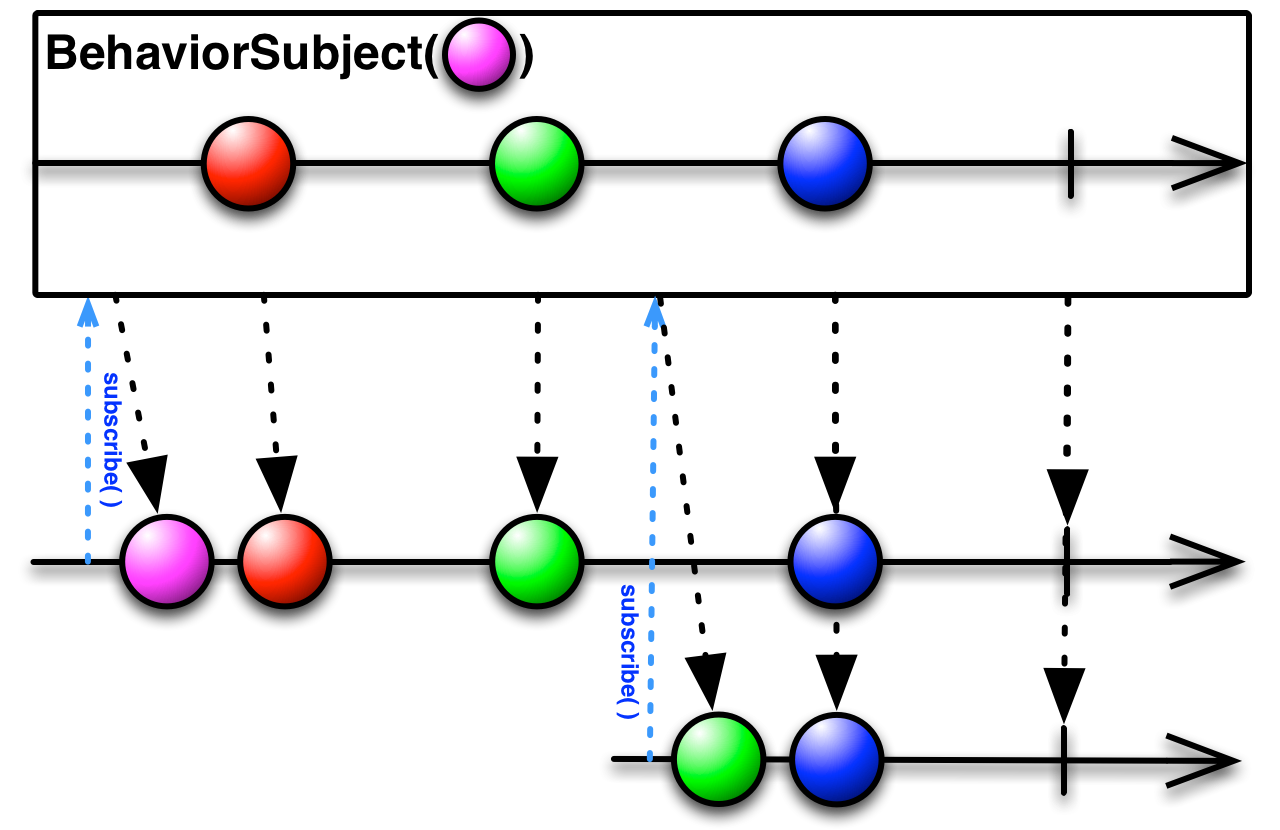


또한 AsyncSubject는 맨 마지막 값을 뒤 이어 오는 옵저버에 전달하는데, 만약 소스 Observable이 오류로 인해 종료될 경우 AsyncSubject는 아무 항목도 배출하지 않고 발생한 오류를 그대로 전달한다.

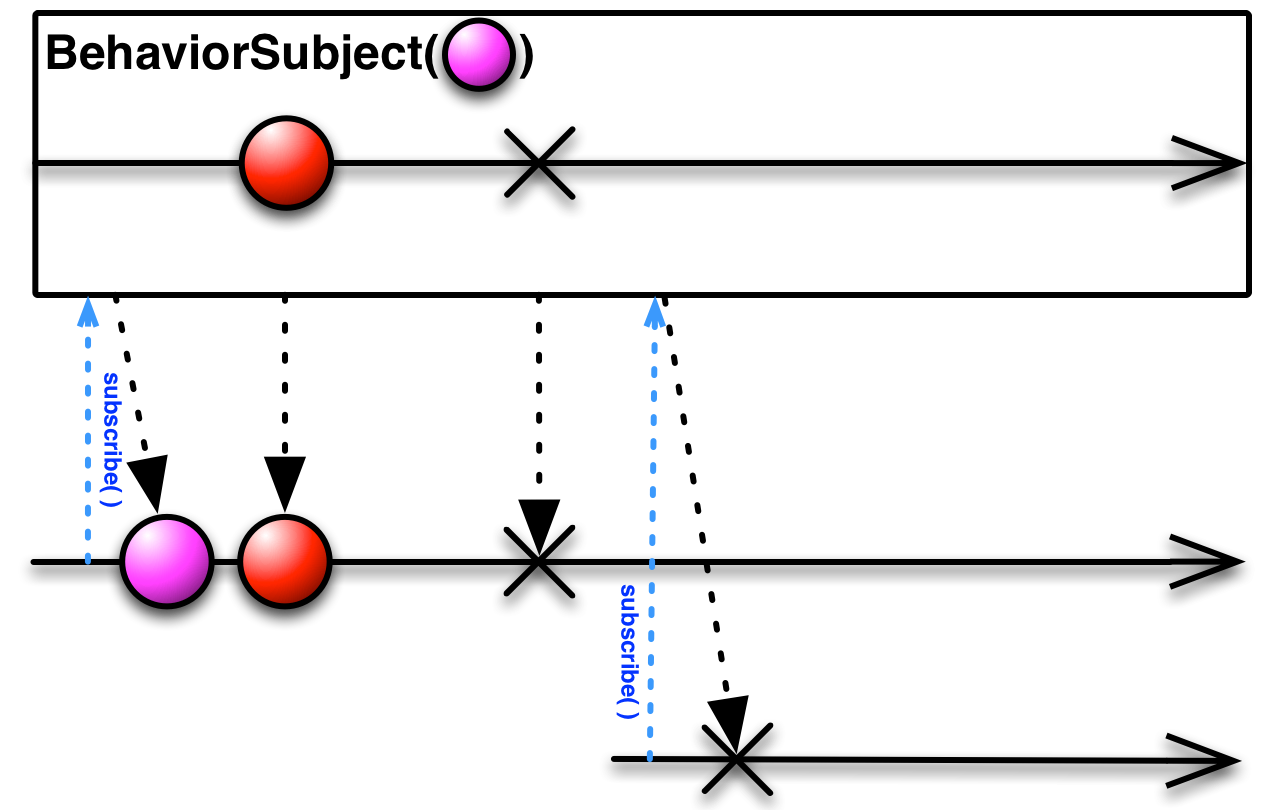
#### 참고

* [Introduction to Rx: AsyncSubject](http://www.introtorx.com/Content/v1.0.10621.0/02_KeyTypes.html#AsyncSubject)

### BehaviorSubject



옵저버가 BehaviorSubject를 구독하기 시작하면, 옵저버는 소스 Observable이 가장 최근에 발행한 항목(또는 아직 아무 값도 발행되지 않았다면 맨 처음 값이나 기본 값)의 발행을 시작하며 그 이후 소스 Observable(들)에 의해 발행된 항목들을 계속 발행한다.

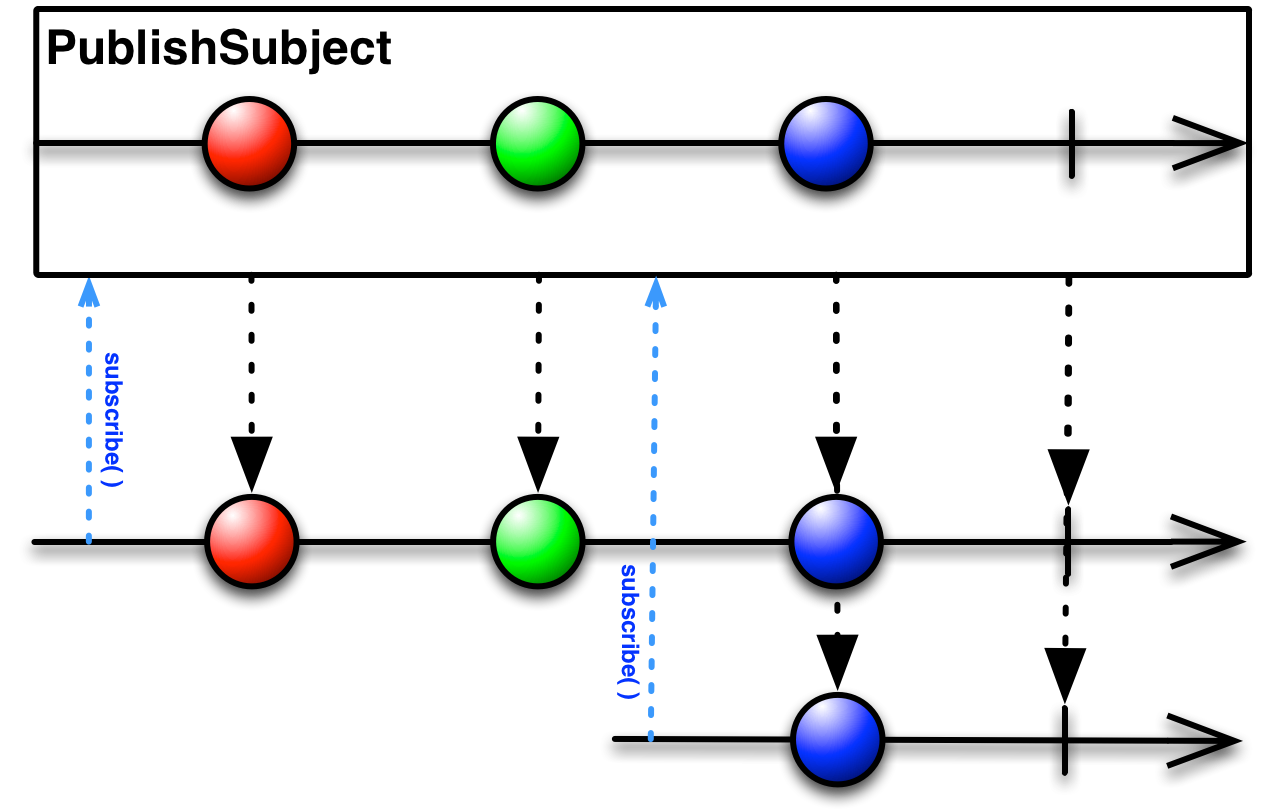


만약, 소스 Observable이 오류 때문에 종료되면 BehaviorSubject는 아무런 항목도 배출하지 않고 소스 Observable에서 발생한 오류를 그대로 전달한다.

#### 참고

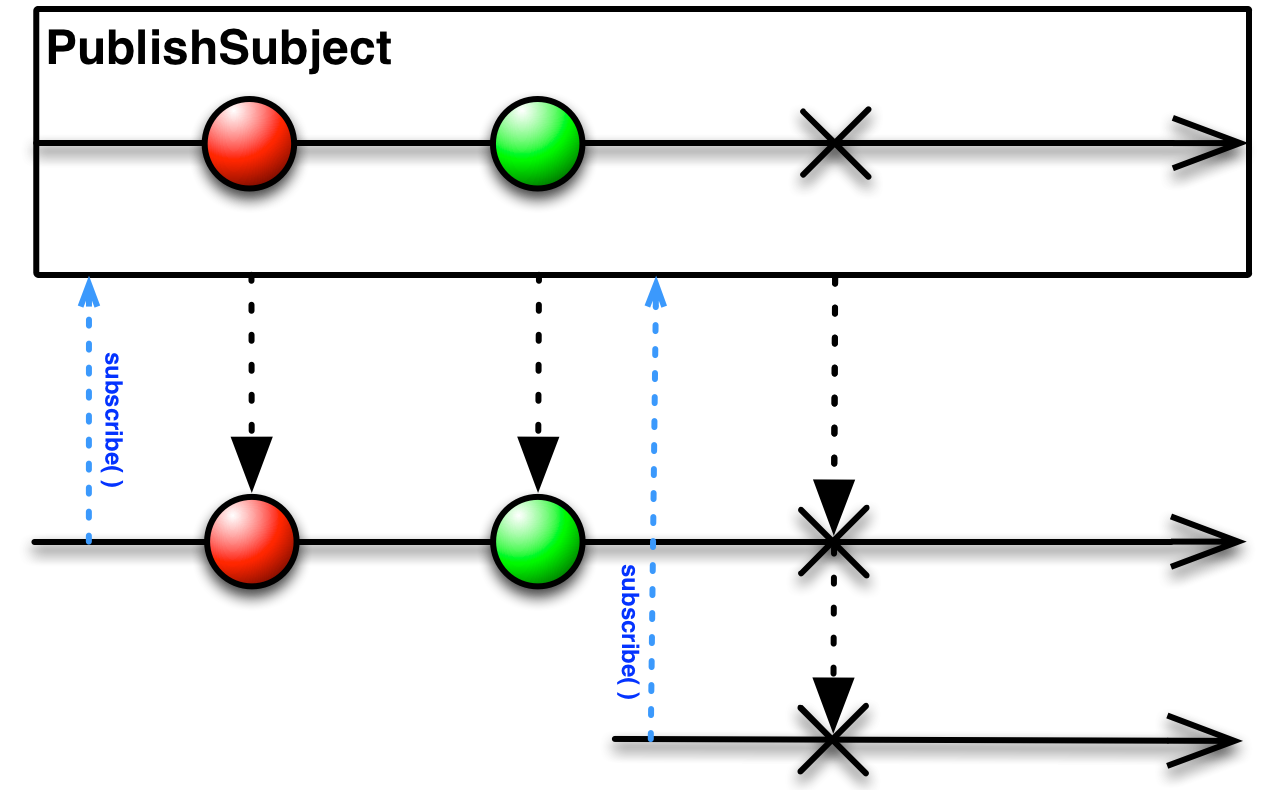
* [Introduction to Rx: BehaviorSubject](http://www.introtorx.com/Content/v1.0.10621.0/02_KeyTypes.html#BehaviorSubject)

### PublishSubject



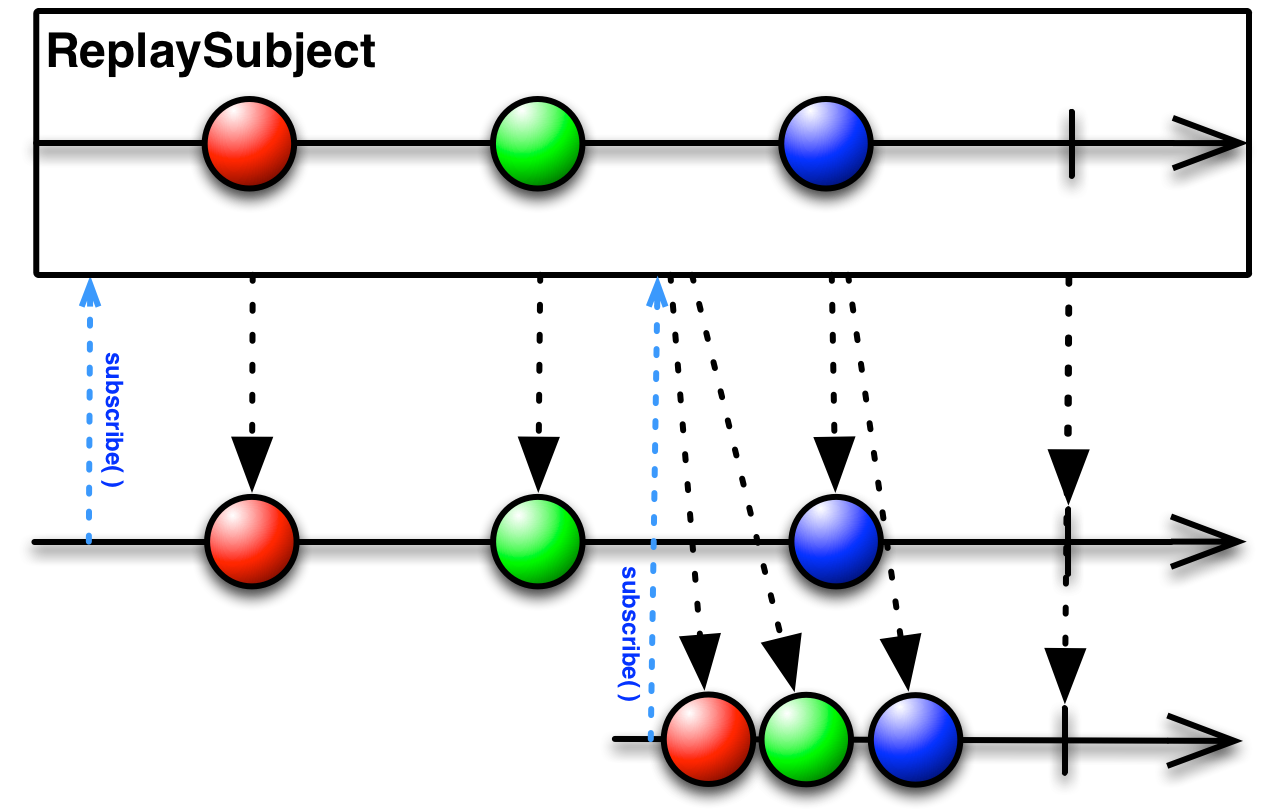
PublishSubject는 구독 이후에 소스 Observable(들)이 배출한 항목들만 옵저버에게 배출한다.

주의할 점은, PublishSubject는 (이를 막지 않는 이상) 생성 시점에서 즉시 항목들을 배출하기 시작할 것이고 이런 특성 때문에 주제가 생성되는 시점과 옵저버가 이 주제를 구독하기 시작하는 그 사이에 배출되는 항목들을 잃어 버릴 수 있다는 단점이 있다. 따라서, 소스 Observable이 배출하는 모든 항목들의 배출을 보장해야 한다면 [Create](http://reactivex.io/documentation/ko/operators/create.html)을 사용해서 명시적으로 "차가운" Observable(항목들을 배출하기 전에 모든 옵저버가 구독을 시작했는지 체크한다)을 생성하거나, PublishSubject 대신 ReplaySubject를 사용해야 한다.



만약, 소스 Observable이 오류 때문에 종료되면 BehaviorSubject는 아무런 항목도 배출하지 않고 소스 Observable에서 발생한 오류를 그대로 전달한다.

### ReplaySubject



ReplaySubject는 옵저버가 구독을 시작한 시점과 관계 없이 소스 Observable(들)이 배출한 모든 항목들을 모든 옵저버에게 배출한다.

ReplaySubject는 몇 개의 생성자 오버로드를 제공하는데 이를 통해, 재생 버퍼의 크기가 특정 이상으로 증가할 경우, 또는 처음 배출 이후 지정한 시간이 경과할 경우 오래된 항목들을 제거한다.

만약, ReplaySubject을 옵저버로 사용할 경우, 멀티 스레드 환경에서는 [Observable 계약](http://reactivex.io/documentation/contract.html) 위반과 주제에서 어느 항목 또는 알림을 먼저 재생해야 하는지 알 수 없는 모호함이 동시에 발생할 수 있기 때문에 (비순차적) 호출을 유발시키는 onNext(또는 그 외 on) 메서드를 사용하지 않도록 주의해야 한다.