Iterator và Observer Pattern (Observable)

- Tại sao lại tìm hiểu Array và Event?
- Có rất nhiều Cấu trúc dữ liệu có thể áp dụng giao thức **Iterable và Iterator** (Object, String, ...)
- **setTimeout** có thể được wrapped bởi **Promise Object**
- DOM Events, XMLHTTPRequest, setInterval, WebSocket, ...
- Array có những `built-in methods` như `map, filter, forEach, ...`
- Observable Pattern giúp chuẩn hóa tất cả các mô hình **Producer-Consumer** về chung cách thức xử lý
    - **Producer** là người kiểm soát việc gửi dữ liệu.
    - **Consumer** là người `subscribe` để nhận dữ liệu từ **Producer**
    - **Consumer** có thể `unsubscribe` để từ chối nhận dữ liệu từ **Producer**
- Chuyện gì sẽ xảy ra nếu tư duy và xử lý dữ liệu gửi đến cho **Consumer** giống như một **Collection**?
- Array và Events đều là một tập hợp (**Collection**) nhiều phần tử
    - Array: Xử lý phần tử liên tục (`forEach`, `map`, `filter`)
    - Observable: Xử lý phần tử ngắt quãng theo thời gian
- Tại sao không dùng Promise?
    - Promise được thiết kế chỉ `resolve` hoặc `reject` một lần
    - Xử lí nhiều dữ liệu phải tạo nhiều Promise
    - Không thể `cancel` một `Promise`

- Một tập hợp các Events gửi đến cho **Consumer** xử lý

```
evt
▶ MouseEvent {isTrusted: true, screenX: 301, screenY: 279, clientX: 301, clientY: 208, …}
evt
▶ MouseEvent {isTrusted: true, screenX: 276, screenY: 297, clientX: 276, clientY: 226, …}
evt
▶ MouseEvent {isTrusted: true, screenX: 254, screenY: 313, clientX: 254, clientY: 242, …}
evt
▶ MouseEvent {isTrusted: true, screenX: 237, screenY: 326, clientX: 237, clientY: 255, …}
evt
▶ MouseEvent {isTrusted: true, screenX: 221, screenY: 337, clientX: 221, clientY: 266, …}
evt
▶ MouseEvent {isTrusted: true, screenX: 208, screenY: 344, clientX: 208, clientY: 273, …}
evt
▶ MouseEvent {isTrusted: true, screenX: 198, screenY: 350, clientX: 198, clientY: 279, …}
evt
▶ MouseEvent {isTrusted: true, screenX: 192, screenY: 352, clientX: 192, clientY: 281, …}
evt
▶ MouseEvent {isTrusted: true, screenX: 179, screenY: 361, clientX: 179, clientY: 290, …}
evt
▶ MouseEvent {isTrusted: true, screenX: 175, screenY: 362, clientX: 175, clientY: 291, …}
evt
▶ MouseEvent {isTrusted: true, screenX: 168, screenY: 365, clientX: 168, clientY: 294, …}
evt
▶ MouseEvent {isTrusted: true, screenX: 162, screenY: 367, clientX: 162, clientY: 296, …}
```

```
// Ex1: Xây dựng API giúp chạy vòng lặp cho một tập hợp Events
  Observable
    .fromEvent('click', boxEl)
    .forEach(function(evt) {
      console.log('evt', evt);
    })

// Ex2: Xây dựng API giúp chạy vòng lặp 3 lần cho Events.
//      Sau 3 lần thì sẽ `unsubscribe` không nhận dữ liệu nữa
  Observable
    .fromEvent('click', boxEl)
    .take(3)
    .forEach(function(evt) {
      console.log('evt', evt);
    })

// Ex3: Xây dựng API giúp `map` từng event ra dạng dữ liệu mới
//      Map từng event ra tọa độ vị trí chuột trên màn hình
//      và chạy vòng lặp để in ra kết quả
  Observable
    .fromEvent('mousemove', boxEl)
    .map(function(evt) {
      return {
        pageX: evt.pageX,
        pageY: evt.pageY
```

```
      }
    })
    .forEach(function(data) {
      console.log('data', data)
    })

// Ex4: Xây dựng API giúp bắt sự kiện double click chuột 2 lần
//    giới hạn trong khoảng thời gian 400ms
const click$ = Observable.fromEvent('click', boxEl)
  click$
    .buffer(
      click$.debounceTime(400)
    )
    .filter(data => data.length === 2)
    .forEach(function(data) {
      console.log('data', data)
    })
```