

RxJS

1. курсы <https://app.pluralsight.com/library/courses/rxjs-angular-reactive-development>
2. [под капотом - setTimeout schedulers](#)
3. В чем преимущество Observable для HttpClient в Angular.
 - их можно отменить
 - перехватчики
4. Promise vs Observable. Какая разница? В чем преимущество?
 - отменяемые
 - много потоков
 - готовые операторы
 - серия значений, а не только одно
 - observable - отложенное вычисление, которое синхронно или асинхронно может вернуть от одного до бесконечного количества значений с момента вызова
 - функция - отложенное вычисление, которое синхронно возвращает одно значение
 - генератор - отложенное вычисление, которое синхронно возвращает от нуля до бесконечного количества значений во время итерации
 - promise - вычисление, которое может вернуть значение
5. Observer - наблюдатель
 - коллекция callback для обработки значений от наблюдаемого

```
const observer = {
  next: value => console.log('Observer got the next value:' +
value),
  error: error => console.log('Observer got an error' + error),
  complete: () => console.log('Observer got a complete
notification'),
};
```

6. Subscription
 - экземпляр выполняемого Observable
 - отменяемый(disposable) через unsubscribe объект
7. [Observable](#) - наблюдаемый поток
 - реализует идею вызываемой коллекции значений или событий
 - lazy push collection - потребитель ждёт данные
 - unicast - однопоточный
 - обратный вызов next -

- обратный вызов error -
- обратный вызов complete -
- lifecycle: creation, subscription, execution, and destruction

```
const myObservable = new Observable(observer =>
  setTimeout(() => {
    observer.next('Hello world!');
  }, 2000)
);
```

8. Subject

- разновидность наблюдаемого потока
- можно подписаться нескольким наблюдателям - multicast
- похож на EventEmitter, единственный способ multicasting
- [asyncSubject](#) - отдаёт только после закрытия входного потока
- [replaySubject](#) - кэширование и повторение
- [behaviorSubject](#) - есть начальное значение

```
const setFabricRunners$: Subject<{ id: number; task$: Observable<any> }[]> = new Subject();
const getFabricRunners$ = setFabricRunners$.asObservable().pipe();
getFabricRunners$.subscribe();

//
const observable = new Observable(subscriber => {
  subscriber.next(1);
  subscriber.next(2);
  subscriber.next(3);
  setTimeout(() => {
    subscriber.next(4);
    subscriber.complete();
  }, 1000);
});

observable.subscribe({
  next(x) { console.log('got value ' + x); },
  error(err) { console.error('something wrong occurred: ' + err); },
  complete() { console.log('done'); }
});
```

9. Operators - чистые функции для обработки коллекций в функциональном стиле как map/concat

10. Schedulers - планировщики

- централизованная отправка значений для контроля конкурентности, времени выполнения вычислений как в setTimeout или requestAnimationFrame
- pipe(observeOn(asapScheduler))

11. Как обработать ошибку в Observable?

- `pipe(catchError())`
- `.subscribe(error())`

12. multicasting

- горячий источник с переиспользованием(`share`) побочных эффектов(источников)
- `publish`
- `multicast`
- `share`
- `shareReplay`

13. Higher order observable

- Observable emits Observable

```
of(1).pipe( // outer observable
  map(item=>of(item)) // inner observable
).subscribe(outer=>outer.subscribe(inner=>console.log(inner, outer)))
```

- наблюдаемые высшего порядка уплощаются операторами `concatMap`, `switchMap`, `MergeMap`

14. Стратегии слияния/схлопывания [flattening](#)

- Merge - слияние
 - без кэширования
 - без потерь
 - без сохранения очередности(`order`)
 - без отписок
- Switch - переключение
 - без кэширования
 - с потерями
 - с сохранением очередности
 - переподписывается(отписывается от старого) на новый поток
 - подходит для автодополнения, выбора из списка с подгрузкой значений
- Concat - объединение потоков целиком
 - кэширует
 - без потерь
 - с сохранением очередности внутри потоков
 - переподписывается только когда текущий поток завершен
 - для получения словарей по `id`
- Combine - парсинг, стартует `pipe`
 - кэширует
 - теряет начальные
 - стартует только когда пришли значения от всех потоков
 - сохраняет очередность

- если необходимо обработать неизвестное количество входных потоков как массив
- Exhaust - истощение
 - с потерями
 - с сохранением очерёдности
 - подходит для авторизации, исключения гонки асинхронных событий
 - игнорирует новые потоки пока не закончится текущий
- share - делиться
 - подписывается на входящий поток, когда подписываются на него внешние подписчики
 - отписывается, если все отписались от него
 - делает поток горячим - новые подписчики получают значения только с момента своей подписки
 - возвращает subject
 - multicasting
 - для кэширования(shareReplay) и websocket

15. подходы к комбинированию потоков

- just in time по требованию - mergeMap, switchMap=>(item)=>mergeMap(item)
- get it all предзагрузка - combineLatest.pipe(filter())

16. лучше комбинировать все потоки в один для упрощения связывания кода в HTML

- ```
<div *ngIf = combineLatest$ | async as model>
 <div>{{model.data1}}
 <div>{{model.data2}}
```

#### 17. Функции создания потоков как combineLatest необходимо импортировать из rxjs, а не из rxjs/operators

#### 18. сигнальные потоки Actions

- Нельзя заменять в сигнальном потоке ошибки через EMPTY - поток может завершиться
- необходимо сразу стартовать поток со значением, чтобы не потерять значения внутренних потоков - BehaviorSubject
- сигнальный поток не завершается, потому некоторые операторы высшего порядка могут бесконечно ждать завершения сигнального потока.

#### 19. Холодные потоки

- обычно unicast
- не стартуют, пока нет подписок
- of(...), from([,...])

#### 20. Горячие потоки

- обычно multicast
- Subject/BehaviorSubject
- стартуют без подписок

