Hello RxSwift

Subscription

- Observables:
  - Just a sequence.
  - Receive elements asynchronously
  - Sequences can have 0 or more elements
  - Once an error or completed event is received, the sequence cannot produce any other element.
- Observers: Receive events.
- Disposing: One additional way an observed sequence can terminate.

Example 1

```
let observable = Observable<Int>.create
{ (observer) -> Disposable in
    observer.on(.next(1))
    observer.onCompleted()
    return Disposables.create()
}
let _ = observable.subscribe { (event) in
    print("--- begin ---")
    sleep(2)
    print("--- end ---")
}
```

Example 2

```
func myInterval(_ interval: TimeInterval) -> Observable<Int> {
    return Observable.create({ (observer) -> Disposable in
        let timer = DispatchSource.makeTimerSource(queue: DispatchQueue.global())
        timer.schedule(deadline: DispatchTime.now() + interval, repeating: interval)

        let cancel = Disposables.create {
            print("dispose")
            timer.cancel()
        }

        var next = 0
        timer.setEventHandler(handler: {
            if cancel.isDisposed {
                print("****")
                return
            }
            observer.on(.next(next))
            next += 1
        })
        timer.resume()

        return cancel
    })
}
```

Example 3

```
extension Reactive where Base: URLSession {
    func response(_ request: URLRequest)
        -> Observable<(Data, HTTPURLResponse)> {
        return Observable.create({ observer -> Disposable in
            let task = self.base.dataTask(with: request,
            completionHandler: { (data, response, error) in
                guard let response = response, let data = data else {
                    observer.on(.error(error ?? RxCocoaURLError.unknown))
                    return
                }
                guard let httpResponse = response as? HTTPURLResponse
                else {
                    let error_tmp = RxCocoaURLError
                        .nonHTTPResponse(response: response)
                    observer.on(.error(error_tmp))
                    return
                }
                observer.on(.next((data, httpResponse)))
                observer.onCompleted()
            })
            task.resume()
            return Disposables.create {
                task.cancel()
            }
        })
    }
}
```

Observable

```
extension ObservableType {
    // MARK: create

    /**
     Creates an observable sequence from a specified subscribe method
implementation.

     - seealso: [create operator on reactivex.io](http://reactivex.io/
documentation/operators/create.html)

     - parameter subscribe: Implementation of the resulting observable
sequence's `subscribe` method.
     - returns: The observable sequence with the specified
implementation for the `subscribe` method.
     */
    public static func create(_ subscribe: @escaping (AnyObserver<E>) ->
Disposable) -> Observable<E> {
        return AnonymousObservable(subscribe)
    }
}
```

Observable

```
final fileprivate class AnonymousObservable<Element> : Producer<Element>
{
    typealias SubscribeHandler = (AnyObserver<Element>) -> Disposable

    let _subscribeHandler: SubscribeHandler

    init(_ subscribeHandler: @escaping SubscribeHandler) {
        _subscribeHandler = subscribeHandler
    }

    override func run<O : ObserverType>(_ observer: O,
                                        cancel: Cancelable)
        -> (sink: Disposable, subscription: Disposable)
        where O.E == Element {
        let sink = AnonymousObservableSink(observer: observer,
                                           cancel: cancel)
        let subscription = sink.run(self)
        return (sink: sink, subscription: subscription)
    }
}
```

subscribe(_:)

```
extension ObservableType {

    /**
     Subscribes an event handler to an observable sequence.

     - parameter on: Action to invoke for each event in the observable sequence.
     - returns: Subscription object used to unsubscribe from the observable sequence.
     */
    public func subscribe(_ on: @escaping (RxSwift.Event<Self.E>)
        -> Swift.Void) -> Disposable

    /**
     Subscribes an element handler, an error handler, a completion handler and disposed handler to an
observable sequence.

     - parameter onNext: Action to invoke for each element in the observable sequence.
     - parameter onError: Action to invoke upon errored termination of the observable sequence.
     - parameter onCompleted: Action to invoke upon graceful termination of the observable sequence.
     - parameter onDisposed: Action to invoke upon any type of termination of sequence (if the
sequence has
     gracefully completed, errored, or if the generation is canceled by disposing subscription).
     - returns: Subscription object used to unsubscribe from the observable sequence.
     */
    public func subscribe(onNext: ((Self.E) -> Swift.Void)? = default,
        onError: ((Error) -> Swift.Void)? = default,
        onCompleted: (() -> Swift.Void)? = default,
        onDisposed: (() -> Swift.Void)? = default) -> Disposable
}
```

subscribe(_:)

subscribe(_:) 是核心方法，subscribe(onNext:onError:onCompleted:onDisposed:) 也是调这个方法

ObservableType 协议默认实现

Observable 类没有实现

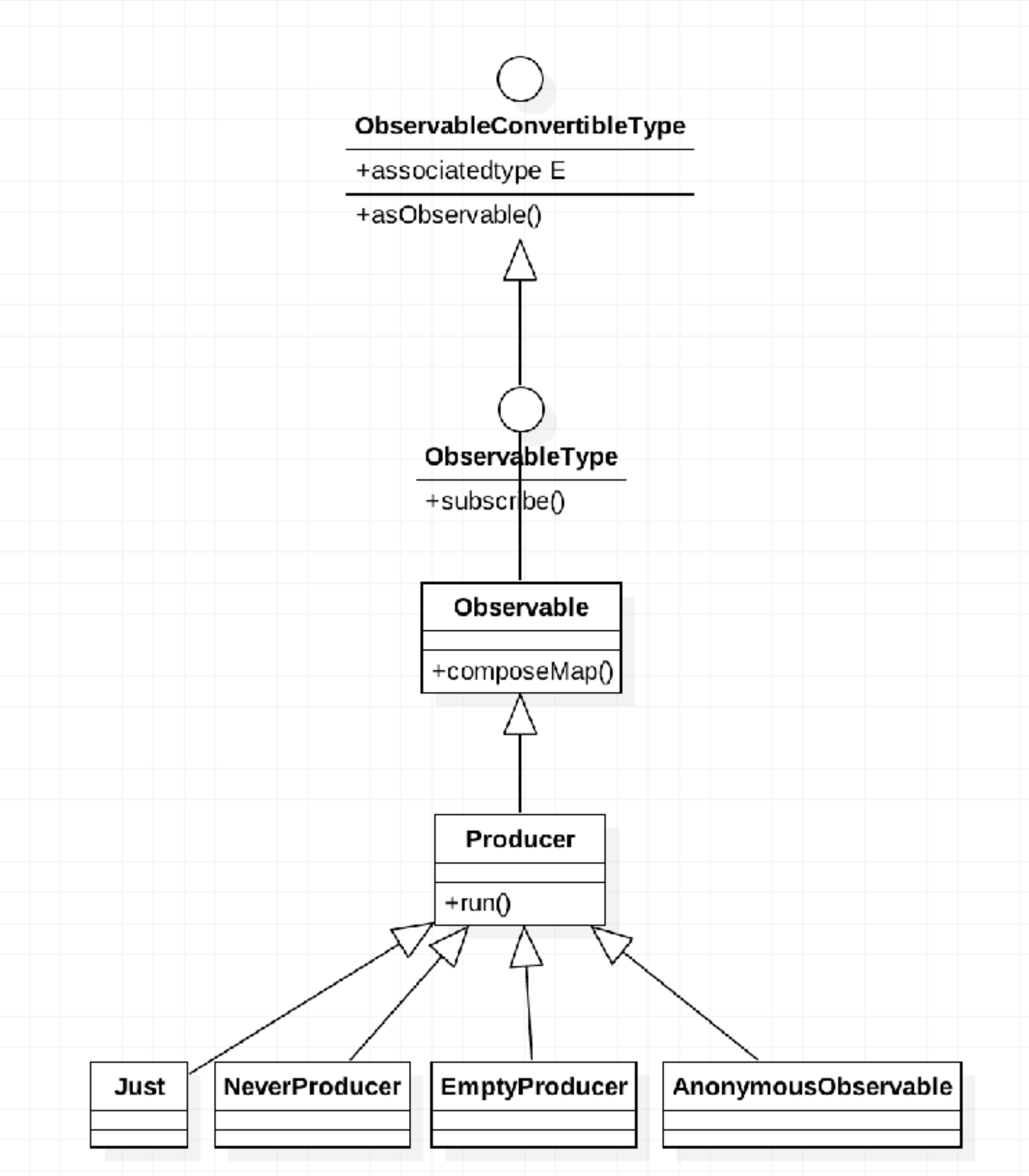
Producer 类实现

Producer - subscribe(_:)

```
override func subscribe<O : ObserverType>(_ observer: O)
    -> Disposable where O.E == Element {
    if !CurrentThreadScheduler.isScheduleRequired {
        // The returned disposable needs to release all references once it was disposed.
        let disposer = SinkDisposer()
        let sinkAndSubscription = run(observer, cancel: disposer)
        disposer.setSinkAndSubscription(sink: sinkAndSubscription.sink,
                                        subscription: sinkAndSubscription
                                            .subscription)

        return disposer
    }
    else {
        return CurrentThreadScheduler.instance.schedule(()) { _ in
            let disposer = SinkDisposer()
            let sinkAndSubscription = self.run(observer, cancel: disposer)
            disposer.setSinkAndSubscription(sink: sinkAndSubscription.sink,
                                            subscription: sinkAndSubscription
                                                .subscription)

            return disposer
        }
    }
}

func run<O : ObserverType>(_ observer: O, cancel: Cancelable)
    -> (sink: Disposable, subscription: Disposable)
    where O.E == Element {
    rxAbstractMethod()
}
```

AnonymousObservable - run(_:cancel:)

```
// AnonymousObservable
override func run<O : ObserverType>(_ observer: O,
                                    cancel: Cancelable)
    -> (sink: Disposable,
    subscription: Disposable) where O.E == Element {
       let sink = AnonymousObservableSink(observer: observer,
                                           cancel: cancel)
        let subscription = sink.run(self)
        return (sink: sink, subscription: subscription)
}

// AnonymousObservableSink -> Sink ~> Disposable
//                      \_ ~> ObserverType
typealias Parent = AnonymousObservable<E>
func run(_ parent: Parent) -> Disposable {
    return parent._subscribeHandler(AnyObserver(self))
}
```



Observables

Reference

- 官方 github：https://github.com/ReactiveX/RxSwift/blob/master/Documentation/GettingStarted.md#disposing
- 订阅流程：http://www.jianshu.com/p/af17ba8e5d14

One More Thing

推荐一个 Keynote 代码高亮工具：

- Highlight
- 中文教程