**ReactiveCocoa比较区分replay, replayLast和replayLazily**

1. **问题引出：**

Subscribing to a Signal

对于一个“普通”的信号，每次订阅都将会导致信号中的代码再执行一遍，且该次订阅者仅接收到该次订阅发送出去的值。

第一个例子演示每次订阅都会重新执行订阅代码。

\_\_block int num = 0;

RACSignal \*signal = [RACSignal createSignal:^RACDisposable \*(id subscriber) {

num++;

NSLog(@"Increment num to: %i", num);

[subscriber sendNext:@(num)];

return nil;

}];

NSLog(@"Start subscriptions");

// Subscriber 1 (S1)

[signal subscribeNext:^(id x) {

NSLog(@"S1: %@", x);

}];

// Subscriber 2 (S2)

[signal subscribeNext:^(id x) {

NSLog(@"S2: %@", x);

}];

// Subscriber 3 (S3)

[signal subscribeNext:^(id x) {

NSLog(@"S3: %@", x);

}];

运行结果如下：

Start subscriptions

Increment num to: 1

S1: 1

Increment num to: 2

S2: 2

Increment num to: 3

S3: 3

可以看到，每次订阅num都在递增，如果不订阅则不会递增。通过这种方式，可以知道信号是懒惰的，如果没有订阅者的话，是不会执行的。

第二个例子演示信号被添加订阅的时候，订阅者是怎么接收发送的值的。

RACSubject \*letters = [RACSubject subject];

RACSignal \*signal = letters;

NSLog(@"Subscribe S1");

[signal subscribeNext:^(id x) {

NSLog(@"S1: %@", x);

}];

NSLog(@"Send A");

[letters sendNext:@"A"];

NSLog(@"Send B");

[letters sendNext:@"B"];

NSLog(@"Subscribe S2");

[signal subscribeNext:^(id x) {

NSLog(@"S2: %@", x);

}];

NSLog(@"Send C");

[letters sendNext:@"C"];

NSLog(@"Send D");

[letters sendNext:@"D"];

NSLog(@"Subscribe S3");

[signal subscribeNext:^(id x) {

NSLog(@"S3: %@", x);

}];

运行结果

Subscribe S1

Send A

S1: A

Send B

S1: B

Subscribe S2

Send C

S1: C

S2: C

Send D

S1: D

S2: D

Subscribe S3

在很多情况下，这是我们想要的预期结果，不过在某些情况下，你不需要订阅的代码再次被执行。例如订阅 一个向网络服务器发送的请求，当服务器返回数据时，多个监听者需要更新(无论有多少个监听者，请求只发送一下（第一个例子就不满足我们的需求）)，或者我们想拿到订阅前信号发送过的值（第二个例子，S2想拿A，B的值或者S3想拿A，B，C，D的值，就不满足我们的需求了）。因此-replay, -replayLast, and -replayLazily应需而生。

1. **replay, replayLast和replayLazily区分比较**

**2.1 Subscribing to a -replay Signal**

　　这个replay方法将返回一个新的信号，当源信号被订阅时，会立即发送给订阅者全部历史的值，不会重复执行源信号中的订阅代码，不仅如此，订阅者还将收到所有未来发送的值。

　　第一个例子演示信号添加新的订阅时，代码是不会再次被执行的。

\_\_block int num = 0;

RACSignal \*signal = [[RACSignal createSignal:^RACDisposable \*(id subscriber) {

num++;

NSLog(@"Increment num to: %i", num);

[subscriber sendNext:@(num)];

return nil;

}] replay];

NSLog(@"Start subscriptions");

// Subscriber 1 (S1)

[signal subscribeNext:^(id x) {

NSLog(@"S1: %@", x);

}];

// Subscriber 2 (S2)

[signal subscribeNext:^(id x) {

NSLog(@"S2: %@", x);

}];

// Subscriber 3 (S3)

[signal subscribeNext:^(id x) {

NSLog(@"S3: %@", x);

}];

Increment num to: 1

Start subscriptions

S1: 1

S2: 1

S3: 1

　　信号首次被订阅时，num立马被递增了，且仅仅递增了一次。这说明了不管有你多个订阅者，订阅代码我只执行了一次。

　　第二个例子演示每个新添加的订阅者接收到信号中全部的值（不管是之前发出的值还是将来发出的值）。

RACSubject \*letters = [RACSubject subject];

RACSignal \*signal = [letters replay];

NSLog(@"Subscribe S1");

[signal subscribeNext:^(id x) {

NSLog(@"S1: %@", x);

}];

NSLog(@"Send A");

[letters sendNext:@"A"];

NSLog(@"Send B");

[letters sendNext:@"B"];

NSLog(@"Subscribe S2");

[signal subscribeNext:^(id x) {

NSLog(@"S2: %@", x);

}];

NSLog(@"Send C");

[letters sendNext:@"C"];

NSLog(@"Send D");

[letters sendNext:@"D"];

NSLog(@"Subscribe S3");

[signal subscribeNext:^(id x) {

NSLog(@"S3: %@", x);

}];

Subscribe S1

Send A

S1: A

Send B

S1: B

Subscribe S2

S2: A

S2: B

Send C

S1: C

S2: C

Send D

S1: D

S2: D

Subscribe S3

S3: A

S3: B

S3: C

S3: D

　　尽管订阅者S3在所有的值发送之后再订阅，然后还能接收到所有的值。

**2.2 Subscribing to a -replayLast Signal**

　　这个replayLast返回一个新的信号，当源信号被订阅时，会立即发送给订阅者最新的值，不会重复执行源信号中的订阅代码。订阅者还会收到信号未来所有的值。

　　对于第一个例子，跟之前replay一样，所以我就不再次演示了。

　　第二个例子演示如何将最新的值提供给新的订阅者

RACSubject \*letters = [RACSubject subject];

RACSignal \*signal = [letters replayLast];

NSLog(@"Subscribe S1");

[signal subscribeNext:^(id x) {

NSLog(@"S1: %@", x);

}];

NSLog(@"Send A");

[letters sendNext:@"A"];

NSLog(@"Send B");

[letters sendNext:@"B"];

NSLog(@"Subscribe S2");

[signal subscribeNext:^(id x) {

NSLog(@"S2: %@", x);

}];

NSLog(@"Send C");

[letters sendNext:@"C"];

NSLog(@"Send D");

[letters sendNext:@"D"];

NSLog(@"Subscribe S3");

[signal subscribeNext:^(id x) {

NSLog(@"S3: %@", x);

}];

Subscribe S1

Send A

S1: A

Send B

S1: B

Subscribe S2

S2: B

Send C

S1: C

S2: C

Send D

S1: D

S2: D

Subscribe S3

S3: D

**2.3 Subscribing to a -replayLazily Signal**

　　这replayLazily方法返回一个新的信号，当源信号被订阅时，会立即发送给订阅者全部历史的值，不会重复执行源信号中的订阅代码。跟replay不同的是，replayLazily被订阅生成新的信号之前是不会对源信号进行订阅的（原文写的有点绕，简单来讲 直到订阅时候才真正创建一个信号，源信号的订阅代码才开始执行）。暂时不理解也没事，看下面的代码输出，和注释。

　这第一个例子会说明跟replay差异。 注意字符串 “Increment num to: 1”是被订阅了之后才打印显示的。而replay和replayLast没被订阅前就打印了“Increment num to: 1” 这个消息。

\_\_block int num = 0;

RACSignal \*signal = [[RACSignal createSignal:^RACDisposable \*(id subscriber) {

num++;

NSLog(@"Increment num to: %i", num);

[subscriber sendNext:@(num)];

return nil;

}] replayLazily]; //跟replay不同的就这么一个地方

NSLog(@"Start subscriptions");

// Subscriber 1 (S1)

[signal subscribeNext:^(id x) {

NSLog(@"S1: %@", x);

}];

// Subscriber 2 (S2)

[signal subscribeNext:^(id x) {

NSLog(@"S2: %@", x);

}];

// Subscriber 3 (S3)

[signal subscribeNext:^(id x) {

NSLog(@"S3: %@", x);

}];

//跟replay比较一下 Increment num to: 1 的显示顺序。

Start subscriptions // 实际订阅

Increment num to: 1 // 信号开始创建

S1: 1

S2: 1

S3: 1

第二个例子演示将全部历史的值提供给任何新的订阅者，就像replay一样。

RACSubject \*letters = [RACSubject subject];

RACSignal \*signal = [letters replayLazily];

NSLog(@"Subscribe S1");

[signal subscribeNext:^(id x) {

NSLog(@"S1: %@", x);

}];

NSLog(@"Send A");

[letters sendNext:@"A"];

NSLog(@"Send B");

[letters sendNext:@"B"];

NSLog(@"Subscribe S2");

[signal subscribeNext:^(id x) {

NSLog(@"S2: %@", x);

}];

NSLog(@"Send C");

[letters sendNext:@"C"];

NSLog(@"Send D");

[letters sendNext:@"D"];

NSLog(@"Subscribe S3");

[signal subscribeNext:^(id x) {

NSLog(@"S3: %@", x);

}];

Subscribe S1

Send A

S1: A

Send B

S1: B

Subscribe S2

S2: A

S2: B

Send C

S1: C

S2: C

Send D

S1: D

S2: D

Subscribe S3

S3: A

S3: B

S3: C

S3: D

　　总结一下：

　　ReactiveCocoa提供了这三个简便的方法允许多个订阅者订阅一个信号，却不会重复执行订阅代码，并且能给新加的订阅者提供订阅前的值。replay和replayLast使信号变成热信号，且会提供所有值(-replay) 或者最新的值(-replayLast) 给订阅者。 replayLazily返回一个冷的信号，会提供所有的值给订阅者。