flutter-Stream介绍

刚介绍过rxSwift中的Subjects的用法,其实flutter中也有一个类似的概念即Stream.

Stream是Dart中自带的封装,代表着事件流.

根据可订阅数,可以分为:

- 单订阅流
  单个订阅流在流的整个生命周期内仅允许有一个listener。它在有收听者之前不会生成事件，并且在取消收听时它会停止发送事件，即使你仍然在Sink.add更多事件。即使在第一个订阅被取消后，也不允许在单个订阅流上进行两次侦听.
- 广播流
  广播流允许任意数量的收听者，且无论是否有收听者，他都能产生事件。所以中途进来的收听者将不会收到之前的消息。

创建单订阅流

1.fromFuture()

- 接收一个Future对象作为参数

```
    Stream<String> stream0 = Stream.fromFuture(getData(0));
    stream0.listen((event) {
      print(event);
    }, onError: (msg) {
      print("fromFuture--error=" + msg);
    }, onDone: () {
      print("fromFuture--done");
    });

  Future<String> getData(int i) async {
    return "fromFutures$i";
  }

// I/flutter (31984): fromFutures0
// I/flutter (31984): fromFuture--done
```

2.fromFutures()

- 接收Future对象数组作为参数

```
    List<Future<String>> list = [getData(1), getData(2),
getData(3)];
    Stream<String> stream1 = Stream.fromFutures(list);//接收
一个Future集合对象作为参数
    stream1.listen((event) {
      print(event);
    }, onError: (msg) {
      print("fromFuture--error=" + msg);
    }, onDone: () {
      print("fromFutures--done");
    });
```

```
// I/flutter (31984): fromFutures1
// I/flutter (31984): fromFutures2
// I/flutter (31984): fromFutures3
// I/flutter (31984): fromFutures--done
```

3.利用 StreamController

- 可以通过 sink 添加事件 sink 下沉，挖掘，灌
- 可以添加错误事件,订阅者在 onError 回调中可以拿到这个错误信息.

```
    StreamController<String> controller =
StreamController();
    controller.add("StreamController--add");
    controller.sink.add("StreamController--sink.add");
    controller.addError("error信息");

    StreamSubscription<String> streamSubscription =
        controller.stream.listen((event) {
      print(event);
    }, onError: (msg) {
      print("StreamController--error=" + msg);
    }, onDone: () {
      print("StreamController--done");
    });
    streamSubscription.onDone(() {
      print("done");
    });
    controller.close(); //不关闭会警告
    // streamSubscription.cancel(); //不cancel会警告
// I/flutter ( 1864): StreamController--add
// I/flutter ( 1864): StreamController--sink.add
// I/flutter ( 1864): StreamController--error=error信息
```

- 可以添加其他流,同时监听多个流.(但这个流如果是单订阅流,不能被订阅过,否则崩溃)

  controller.addStream(stream2);

```
Stream<String> stream2 = Stream.fromIterable(
        ["fromIterable1", 'fromIterable2', iterable 迭代的
'fromIterable3']); //接收一个集合对象作为参数
    // stream2.listen((event) {
    //   print(event);
    // }, onError: (msg) {
    //   print("fromIterable--error=" + msg);
    // }, onDone: () {
    //   print("fromIterable--done");
    // });

    StreamController<String> controller =
StreamController(); //StreamController

    controller.add("StreamController--add");

    controller.sink.add("StreamController--sink.add");
    controller.addError("error信息");
    controller.addStream(stream2);

    StreamSubscription<String> streamSubscription =
        controller.stream.listen((event) {
      print(event);
    }, onError: (msg) {
      print("StreamController--error=" + msg);
    }, onDone: () {
      print("StreamController--done");
    });
    streamSubscription.onDone(() {
      print("done");
    });
```

StreamController使用完成,需要close()
StreamSubscription需要cancle() (在此例中,streamSubscription.cancel()了,订阅者就收不到消息了,所以实际使用时一般可在State的dispose方法中调用)

4.async*异步生成器

```
  /// 返回从1-》to的序列流
  Stream<int> countStream(int to) async* {
    //async*异步生成器
    for (int i = 1; i <= to; i++) {
      yield i;
    }
  }

countStream(9).listen((event) {
      print(event);
    });

//I/flutter ( 1864): 1
//I/flutter ( 1864): 2
//I/flutter ( 1864): 3
```

```
//I/flutter ( 1864): 4
//I/flutter ( 1864): 5
//I/flutter ( 1864): 6
//I/flutter ( 1864): 7
//I/flutter ( 1864): 8
//I/flutter ( 1864): 9
```

yield 使用语法可参考 https://cloud.tencent.com/developer/article/1633899

- 这个其中还可以使用一些操作符

  where

```
countStream(9).where((event) => event % 2 ==
0).listen((event) {
      //筛选
      print("where-(event % 2 == 0)--$event");
    });

//I/flutter ( 1864): where-(event % 2 == 0)--2
//I/flutter ( 1864): where-(event % 2 == 0)--4
//I/flutter ( 1864): where-(event % 2 == 0)--6
//I/flutter ( 1864): where-(event % 2 == 0)--8
```

map

```
 countStream(9).map((event) => event * 2).listen((event) {
      //变换*2
      print("map变换-event * 2 == 0--$event");
    });
//I/flutter ( 1864): map变换-event * 2 == 0--2
```

```
//I/flutter ( 1864): map变换-event * 2 == 0--4
//I/flutter ( 1864): map变换-event * 2 == 0--6
//I/flutter ( 1864): map变换-event * 2 == 0--8
//I/flutter ( 1864): map变换-event * 2 == 0--10
//I/flutter ( 1864): map变换-event * 2 == 0--12
//I/flutter ( 1864): map变换-event * 2 == 0--14
//I/flutter ( 1864): map变换-event * 2 == 0--16
//I/flutter ( 1864): map变换-event * 2 == 0--18
```

take

```
 countStream(9).take(4).listen((event) {
      //指定只发送4个事件
      print("take-只发送4个事件 == 0--$event");
    });
//I/flutter ( 1864): take-只发送4个事件 == 0--1
//I/flutter ( 1864): take-只发送4个事件 == 0--2
//I/flutter ( 1864): take-只发送4个事件 == 0--3
//I/flutter ( 1864): take-只发送4个事件 == 0--4
```

transform

```
    final transformer =
        StreamTransformer<int,
String>.fromHandlers(handleData: (value, sink) {
      if (value == 9) {
        sink.add("是$value 吗? 你猜对了");
      } else {
        sink.add('是$value 吗? 还没猜中, 再试一次吧');
      }
    });
    countStream(9).transform(transformer).listen((event) {
      print(event);
    }, onError: (msg) {
      print("error---" + msg);
    });
//I/flutter ( 1864): 是1 吗? 还没猜中，再试一次吧
//I/flutter ( 1864): 是2 吗? 还没猜中，再试一次吧
//I/flutter ( 1864): 是3 吗? 还没猜中，再试一次吧
//I/flutter ( 1864): 是4 吗? 还没猜中，再试一次吧
//I/flutter ( 1864): 是5 吗? 还没猜中，再试一次吧
//I/flutter ( 1864): 是6 吗? 还没猜中，再试一次吧
//I/flutter ( 1864): 是7 吗? 还没猜中，再试一次吧
//I/flutter ( 1864): 是8 吗? 还没猜中，再试一次吧
//I/flutter ( 1864): 是9 吗? 你猜对了
```

创建多订阅流

1.单订阅流.asBroadcastStream()

- StreamController.stream本身是单订阅流 ,只能被订阅一次.
- StreamController.stream.asBroadcastStream() 将单订阅流转成了多订阅流,但是流本身还是StreamController.stream的流,所以下面代码如果先订阅一次 StreamController.stream,再订阅一次StreamController.stream.asBroadcastStream(), 代码依然会崩溃.

```
StreamController controller2 = StreamController();
```

```
    Stream broadcastStream =
controller2.stream.asBroadcastStream();
//    controller2.stream.listen((event) {
//      print("listen0--");//controller2.stream单订阅流  只能
被订阅一次  Bad state: Stream has already been listened to.
//    });
    broadcastStream.listen((event) {
      print("listen1--" + event);
    });
    broadcastStream.listen((event) {
      print("listen2--" + event);
    });
    controller2.add("broadcastStream");
    controller2.close();
//I/flutter ( 1864): listen1--broadcastStream
//I/flutter ( 1864): listen2--broadcastStream
```

2.StreamController.broadcast()

```
StreamController controller3 =
StreamController.broadcast();
    controller3.stream.listen((event) {
      print("listen1--" + event);
    });
    controller3.stream.listen((event) {
      print("listen2--" + event);
    });
    controller3.add("StreamController.broadcast");
    controller3.close();
//I/flutter ( 1864): StreamController--add
//I/flutter ( 1864): StreamController--sink.add
//I/flutter ( 1864): StreamController--
error=StreamController--add--error
```

StreamBuilder的用法

- StreamBuilder本身就是一个widget
- StreamBuilder需要传入一个stream,通过stream内部不同的event值,builder中可以返回不同的widget
- 可实现局部控件刷新

```
class StreamPage extends StatefulWidget {
  @override
  _StreamPageState createState() => _StreamPageState();
}

class _StreamPageState extends State<StreamPage> {
  StreamController<String> builderController =
StreamController();

  @override
  void initState() {
    // TODO: implement initState
    super.initState();
    // print(
    //     "单订阅流--------------------------------\n单个订阅流在流的整个生命周期内仅允许有一个listener。它在有收听者之前不会生成事件，并且在取消收听时它会停止发送事件，即使你仍然在Sink.add更多事件。即使在第一个订阅被取消后，也不允许在单个订阅流上进行两次侦听");
    // Stream<String> stream0 =
Stream.fromFuture(getData(0)); //接收一个Future对象作为参数
    // stream0.listen((event) {
    //   print(event);
    // }, onError: (msg) {
    //   print("fromFuture--error=" + msg);
    // }, onDone: () {
    //   print("fromFuture--done");
    // });

    // List<Future<String>> list = [getData(1), getData(2),
getData(3)];
    // Stream<String> stream1 =
Stream.fromFutures(list); //接收一个Future集合对象作为参数
    // stream1.listen((event) {
    //   print(event);
    // }, onError: (msg) {
    //   print("fromFuture--error=" + msg);
    // }, onDone: () {
    //   print("fromFutures--done");
    // });

    // Stream<String> stream2 = Stream.fromIterable(
    //     ["fromIterable1", 'fromIterable2',
'fromIterable3']); //接收一个集合对象作为参数
    // stream2.listen((event) {
    //   print(event);
    // }, onError: (msg) {
    //   print("fromIterable--error=" + msg);
    // }, onDone: () {
    //   print("fromIterable--done");
    // });

    // StreamController<String> controller =
StreamController(); //StreamController

    // controller.add("StreamController--add");
    // ///Cannot add event after closing
    // controller.sink.add("StreamController--sink.add");
    // controller.addError("error信息");
    // // controller.addStream(stream2);

    // StreamSubscription<String> streamSubscription =
    //     controller.stream.listen((event) {
    //   print(event);
    // }, onError: (msg) {
    //   print("StreamController--error=" + msg);
    // }, onDone: () {
    //   print("StreamController--done");
    // });
    // streamSubscription.onDone(() {
    //   print("done");
    // });

    // streamSubscription.onDone(() {
    //   print("done");
    // });
    // controller.close(); //不关闭会警告
    // // streamSubscription.cancel(); //不cancel会警告

    // countStream(9).listen((event) {
    //   print(event);
    // });
    // countStream(9).where((event) => event % 2 ==
0).listen((event) {
    //   //筛选
    //   print("where-(event % 2 == 0)--$event");
    // });
    // countStream(9).map((event) => event *
2).listen((event) {
    //   //变换*2
    //   print("map变换-event * 2 == 0--$event");
    // });
    // countStream(9).take(4).listen((event) {
    //   //指定只发送4个事件
    //   print("take-只发送4个事件 == 0--$event");
    // });

    // final transformer =
    //     StreamTransformer<int,
String>.fromHandlers(handleData: (value, sink) {
    //   if (value == 9) {
    //     sink.add("是$value 吗? 你猜对了");
    //   } else {
    //     sink.add('是$value 吗? 还没猜中，再试一次吧');
    //   }
    // });

    // countStream(9).transform(transformer).listen((event)
{
    //   print(event);
    // }, onError: (msg) {
    //   print("error---" + msg);
    // });

    // print(
    //     "broadcast streams 多订阅流---------\n广播流允许任意
数量的收听者，且无论是否有收听者，他都能产生事件。所以中途进来的收听者将
不会收到之前的消息。");

    // StreamController controller2 = StreamController();
    // Stream broadcastStream =
controller2.stream.asBroadcastStream();
    // // controller2.stream.listen((event) {
    // //   print(
    // //       "listen0--"); //controller2.stream单订阅流
只能被订阅一次  Bad state: Stream has already been listened
```

```
to.
    // // });
    // broadcastStream.listen((event) {
    //   print("listen1--" + event);
    // });
    // broadcastStream.listen((event) {
    //   print("listen2--" + event);
    // });
    // controller2.add("broadcastStream");
    // controller2.close();

    // StreamController controller3 =
StreamController.broadcast();
    // controller3.stream.listen((event) {
    //   print("listen1--" + event);
    // });
    // controller3.stream.listen((event) {
    //   print("listen2--" + event);
    // });
    // controller3.add("StreamController.broadcast");
    // controller3.close();

    Future.delayed(Duration(seconds: 2), () {
      builderController.add("我变化了");
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("Stream"),
      ),
      body: StreamBuilder<String>(
          stream: builderController.stream,
          builder: (context, snapshot) {
            return snapshot.hasData
                ? Text(' ${snapshot.data}')
                : Text('waiting for data');
          }),
    );
  }

  // Future<String> getData(int i) async {
  //   return "fromFutures$i";
  // }

  // /// 返回从1-》to的序列流
  // Stream<int> countStream(int to) async* {
  //   //async*异步生成器
  //   for (int i = 1; i <= to; i++) {
  //     yield i;
  //   }
  // }

  @override
  void dispose() {
    builderController.close();
    super.dispose();
  }
}
```