# Flutter中的Stream

## 一、什么是Stream?

众所周知，dart是单线程语言，所以异步就非常重要的。而Stream就是处理异步事件的。rxdart，Bloc，flutter_redux,fish_redux都对Stream做了封装。Stream是dart语言自带的。

Stream从表面意思来看就是流，使用RxJava的朋友应该是很熟悉的，它们都是对把事件放到流上处理。核心设计模式就是观察者模式。

## 二、如何创建一个Stream

- 通过构造方法
- 使用StreamController

### 1.Stream有三种构造方法

- Stream.fromFuture(Future<T> future);传递一个异步任务Future就可以创建一个Stream。使用如下：

```
Stream stream =
```

```
Stream.fromFuture(Future.delayed(Duration(seconds: 1),(){
      return "Hello Stream";
    }));

   stream.listen((event) {
     print(event);
   },onError: (e){
     print("onError");
   },onDone: (){
     print("onDone");
   });
  }
```

事件正常会走 print(event);打印出"Hello Stream"。如果Future中的处理逻辑报错会走 print("onError");不管有没有报错，处理完成都会走 print("onDone");

- Stream.fromFutures（Iterable<Future<T>> futures）一组Future订阅一个单订阅流，每个Future都会触发data或者error回调，整个完成之后会回调onDone，然后关闭流。使用如下：

```
Stream stream =  Stream.fromFutures([getFuture(1,"hello
Stream1"),getFuture(2,"hello Stream2"),
      getFuture(3,"hello Stream3")]);
  stream.listen((event) {
    print(event);
  },onError: (e){
    print("onError:$e");
  }).onDone(() {
```

```
    print("onDone");
  });

///创建Future
  Future getFuture(int seconds,String result){
   return Future.delayed(Duration(seconds: seconds),(){
      return result;
    });
  }
```

打印结果如下：

```
I/flutter ( 4923): hello Stream1
I/flutter ( 4923): hello Stream2
I/flutter ( 4923): hello Stream3
I/flutter ( 4923): onDone
```

- Stream.fromIterable(Iterable<T> elements):从一个集合中获取数据的单订阅流。每个数据都会有自己的回调。使用如下：

```
Stream.fromIterable(["a", "b", "c"]).listen((event) {
      print(event);
    }, onError: (e) {
      print(e);
    }).onDone(() {
      print("onDone");
    });
```

打印结果：

```
I/flutter ( 4923): a
I/flutter ( 4923): b
I/flutter ( 4923): c
I/flutter ( 4923): onDone
```

## 2.通过 StreamController 创建 Stream

- 创建 StreamController
- 获取 StreamSink 用作事件入口
- 使用 stream 对象用于监听
- 通过监听得到的 StreamSubscription 管理订阅事件，最后在不需要的时候关闭即可。

```
import 'dart:async';

class DataBloc {
  ///定义一个controller
  StreamController<List<String>> _dataController =
StreamController<List<String>>();

  ///获取StramSink 做add入口
  StreamSink<List<String>> get dataSink =>
_dataController.sink;

  ///获取Stream 用于监听
  Stream<List<String>> get dataStream =>
_dataController.stream;

///时间订阅对象
  StreamSubscription _dataSubscription;
```

```
  init() {
    ///监听事件
    _dataSubscription = dataStream.listen((value){
        print("dataStream 监听到了");
        print("$value");
    });
    ///添加时间
    dataSink.add(["first","second","three","more"]);

  }

  close(){
    ///关闭
    _dataSubscription.cancel();
    _dataController.close();
  }
}
```

# 三、Steam的种类

流有两种：

- "Single-subscription" streams 单订阅流，只允许订阅一次，当有多个订阅者的时候会报如下错误""Bad state: Stream has already been listened to.
- "broadcast" streams **多订阅流 后面的订阅者不会接收到之前的事件。而单订阅流中，订阅者是可以接收到订阅之前的事件。**

单订阅流可以通过如下代码转换成多订阅流。

```
Stream stream =  dataStream.asBroadcastStream();
```

# 四、StreamBuilder

StreamBuilder是Flutter中的一个Widget，它可以和Steam结合起来使用。如下：

```
child:StreamBuilder<List<String>>(
            stream: dataBloc.dataStream,
            initialData: ["none"],
            builder: (BuildContext
context,AsyncSnapshot<List<String>> snapshot){
              ///snapshot 是数据快至的意思
              var data = snapshot.data;
              print("$data");
              return Text("helloe");
            },
          ) ,
```

上面的steam接受一个流，initData可以接受一个初始化事件。在builder里面处理接收到的信息，然后渲染到子控件中。

五、Stream的异步实现

首先要说下dart的异步实现：dart是单线程语言，和大多单线程语言一样，dart是通过消息循环机制来运行的。这里面主要包括两个：一个是microtask的内部队列，一个是event

的外部队列。microtask的优先级高于evnet的优先级。

dart中的触摸，io，点击，滑动，绘制都属于event外部队列。microTask内部队列主要由dart内部产生，stream中执行的异步模式就是StreamMicrotask。microtask优先级高于event外部事件，所以太多的微任务（microtask）会造成绘制，点击的阻塞卡顿。

最后附上参考连接

https://www.jianshu.com/p/b7cca3a89618?utm_source=desktop&utm_medium=timeline