Dart | 什么是Stream


Vadaski LV.5
2018年10月01日 14:34 · 阅读 28072

关注

🌏前言

Stream 和 **Future**都是**Dart:async**库的核心API，对异步提供了非常好的支持。

我思考了很久，究竟应该如何向大家介绍Stream（流）。因为Stream非常有用，它是为处理**异步事件**而生的。而在应用中有大量的场景需要使用异步事件，例如请求网络，和用户交互等等，它们都无法同步完成。Stream能够极大的帮助我们处理这些问题。😍

但是对于刚接触的新手来说，流确实足够抽象，以至于大家需要花费非常多的时间来理解它。

所以我将会尽我所能向大家介绍Stream。

🚄Stream

🤔什么是 Stream

Stream非常有特点但却不太好理解，我与其按照字面意思把它看作流，更愿意把它看成一个工厂或者是机器。

我们来看看这个机器它有什么特点：

- 它有一个入口，可以放东西/指令（anything）
- 这个机器不知道入口什么时候会放东西进来
- 中间的机器能够生产或者加工，这应该会耗费一些时间
- 他有一个出口，应该会有产品从那出来
- 我们也不知道到底什么时候产品会从出口出来

整个过程，时间都是一个不确定因素，我们随时都可以向这个机器的入口放东西进去，放进去了以后机器进行处理，但是我们并不知道它多久处理完。所以出口是需要专门派人盯着的，等待机

器流出东西来。整个过程都是以异步的眼光来看的。

✈我们将机器模型转化成Stream

- 这个大机器就是**StreamController**，它是创建流的方式之一。
- StreamController有一个入口，叫做**sink**
- sink可以使用**add**方法放东西进来，放进去以后就**不再关心**了。
- 当有东西从sink进来以后，我们的机器就开始工作啦，空空空。
- StreamController有一个出口，叫做stream
- 机器处理完毕后就会把产品从出口丢出来，但是我们并不知道什么时候会出来，所以我们需要使用listen方法一直监听这个出口。
- 而且当多个物品被放进来了之后，它不会打乱顺序，而是先入先出。

通过这个例子，相信大家对流应该都有了基础印象，那么要解释后面的东西就不难了。

🤔如何使用 Stream

获得 Stream 的方法：

- 通过构造函数
- 使用StreamController
- IO Stream

stream有三个构造方法：

- **Stream.fromFuture**:从Future创建新的单订阅流,当future完成时将触发一个data或者error，然后使用Down事件关闭这个流。
- **Stream.fromFutures**:从一组Future创建一个单订阅流，每个future都有自己的data或者error事件，当整个Futures完成后，流将会关闭。如果Futures为空，流将会立刻关闭。
- **Stream.fromIterable**:创建从一个集合中获取其数据的单订阅流。

dart 复制代码

```dart
Stream.fromItreable([1,2,3]);
```

🧐监听 Stream 的方法

监听一个流最常见的方法就是listen。当有事件发出时，流将会通知listener。Listen方法提供了这几种触发事件：

- onData(必填)：收到数据时触发
- onError：收到Error时触发
- onDone：结束时触发
- unsubscribeOnError：遇到第一个Error时是否取消订阅，默认为false

使用 await for 处理 Stream

除了通过 listen，我们还可以使用 await for 来处理。

dart 复制代码

```dart
Future<int> sumStream(Stream<int> stream) async {
  var sum = 0;
  await for (var value in stream) {
    sum += value;
  }
  return sum;
}
```

这段代码将会接收一个 Stream 然后统计所有事件之和，然后返回结果。await for 能够在每个事件到来的时候处理它。我们知道，一个 stream 它接收事件的时机是不确定的，那什么时候应该退出 await for 循环呢？答案是，当这个 Stream 完成或关闭的时候。

你可以复制这段代码到 DartPad 里面多做几次试验。

😏 StreamController

如果你想创建一条新的流的话，非常简单！😀 使用StreamController，它为你提供了非常丰富的功能，你能够在streamController上发送数据，处理错误，并获得结果！

dart 复制代码

```dart
//任意类型的流
StreamController controller = StreamController();
controller.sink.add(123);
controller.sink.add("xyz");
controller.sink.add(Anything);

//创建一条处理int类型的流
StreamController<int> numController = StreamController();
numController.sink.add(123);
```

泛型定义了我们能向流上推送什么类型的数据。它可以是任何类型！

我们再来看看如何获取最后的结果。

dart 复制代码

```dart
StreamController controller = StreamController();

//监听这个流的出口，当有data流出时，打印这个data
StreamSubscription subscription =
controller.stream.listen((data)=>print("$data"));

controller.sink.add(123);
```

输出: 123

你需要将一个方法交给stream的listen函数，这个方法入参(data)是我们的StreamController处理完毕后产生的结果，我们监听出口，并获得了这个结果(data)。这里可以使用lambda表达式，也可以是其他任何函数。

（这里我为了方便区分，把listen说成函数，(data)=>print(data)说成方法，其实是一个东西。）

通过 async* 生成 stream

如果我们有一系列事件需要处理，我们也许会需要把它转化为 stream。这时候可以使用 *async* - *yield** 来生成一个 Stream。

dart 复制代码

```dart
Stream<int> countStream(int to) async* {
  for (int i = 1; i <= to; i++) {
    yield i;
  }
}
```

当循环退出时，这个 Stream 也就 done 了。我们可以结合之前说的 await for 更加深刻的体验一下。

你可以在 这里 直接运行我的样例代码。

🤠 Transforming an existing stream

假如你已经有了一个流，你可以通过它转化成为一条新的流。非常简单！流提供了map()，where()，expand()，和take()方法，能够轻松将已有的流转化为新的流。

where

如果你想要筛选掉一些不想要的事件。例如一个猜数游戏，用户可以输入数字，当输入正确的时候，我们做出一定反应。而我们必须筛选掉所有错误的答案，这个时候我们可以使用where筛选掉不需要的数字。

dart 复制代码

```
stream.where((event){...})
```

where函数接收一个事件，每当这个流有东西流到where函数的时候，这就是那个事件。我们或许根本不需要这个事件，但是必须作为参数传入。

take

如果你想要控制这个流最多能传多少个东西。比如输入密码，我们可能想让用户最多输四次，那么我们可以使用take来限制。

dart 复制代码

```
stream.take(4);
```

take函数接收一个int，代表最多能经过take函数的事件次数。当传输次数达到这个数字时，这个流将会关闭，无法再传输。

transform

如果你需要更多的控制转换，那么请使用transform()方法。他需要配合StreamTransformer进行使用。我们先来看下面一段猜数游戏，然后我会向你解释。

dart 复制代码

```
StreamController<int> controller = StreamController<int>();

final transformer = StreamTransformer<int,String>.fromHandlers(
    handleData:(value, sink){
        if(value==100){
      sink.add("你猜对了");
    }
        else{ sink.addError('还没猜中，再试一次吧');
    }
```

```
    });

    controller.stream
              .transform(transformer)
              .listen(
                  (data) => print(data),
                  onError:(err) => print(err));

      controller.sink.add(23);
      //controller.sink.add(100);
```

输出: 还没猜中，再试一次吧

StreamTransformer<S,T>是我们stream的检查员，他负责接收stream通过的信息，然后进行处理返回一条新的流。

- S代表之前的流的输入类型，我们这里是输入一个数字，所以是int。
- T代表转化后流的输入类型，我们这里add进去的是一串字符串，所以是String。
- handleData接收一个value并创建一条新的流并暴露sink，我们可以在这里对流进行转化。
- 我们还可以addError进去告诉后面有问题。

然后我们监听transform之后的流，当转换好的event流出时，我们打印这个event，这个event就是我们刚才add进sink的数据。onError能够捕捉到我们add进去的err。

🤔Stream的种类

流有两种

- "Single-subscription" streams 单订阅流
- "broadcast" streams 多订阅流

"Single-subscription" streams

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。

单订阅流通常用于流式传输更大的连续数据块，如文件I / O.

dart 复制代码

```dart
StreamController controller = StreamController();

controller.stream.listen((data)=> print(data));
controller.stream.listen((data)=> print(data));

controller.sink.add(123);
```

输出： Bad state: Stream has already been listened to. **单订阅流不能有多个收听者。**

"Broadcast" streams

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。

如果多个收听者想要收听单个订阅流，请使用asBroadcastStream在非广播流之上创建广播流。

如果在触发事件时将收听者添加到广播流，则该侦听器将不会接收当前正在触发的事件。如果取消收听，收听者会立即停止接收事件。

一般的流都是单订阅流。从Stream继承的广播流必须重写isBroadcast 才能返回true。

dart 复制代码

```dart
StreamController controller = StreamController();
//将单订阅流转化为广播流
Stream stream = controller.stream.asBroadcastStream();

stream.listen((data)=> print(data));
stream.listen((data)=> print(data));

controller.sink.add(123);
```

输出： 123 123

参考文章

- 异步编程：使用 stream

写在最后

以上就是关于Dart中Stream的简单介绍，如果你还有任何疑问或者建议，欢迎在下方评论区或者邮箱告诉我！我会在24小时内尽快联系您😉

下一篇文章我将向大家介绍一种非常棒的状态管理方式，它是Google团队力推的状态管理方法，我认为它真的很酷！

所以，准备好迎接BLoC了吗😎

分类： Android 标签： Flutter