Flutter - Stream 基础知识

在本指南中，我们将介绍 Dart 中 Steam 的基础知识，如何使用，管理和创建 Steam。

什么是 Stream

关于如何可视化 Stream 有很多比较，因此我将使用一个常见的比较。Stream 就像管道，您将一个值放在一端，如果另一端有一个侦听器，则该侦听器将接收该值。一个 Stream 可以有多个侦听器，并且将所有这些侦听器放入管道后将收到相同的值。将值放在流上的方式是使用 StreamController。

完整视频可以在这里看到

如何创建流

如果要创建一个 Stream，可以在其中添加值，请从 StreamController 开始。

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

这将构造一个控制器，您可以使用该控制器来操纵该控制器

管理的流。可以通过`stream`属性访问控制器流

```
Stream stream = controller.stream;
```

如何使用 Steam

接下来要做的是能够从 Steam 中获取值。这通常称为订阅或收听 Steam。订阅流时，将仅获得订阅后发出的值（放入 Steam 中）。您可以通过调用该`listen`函数并为它提供一个 Function 来订阅流，以在有新值可用时回调该值，通常将其称为回调函数，或仅称为回调。

```
stream.listen((value) {
  print('Value from controller: $value');
});
```

发射 / 向 Steam 添加值

当您订阅了 Steam 时，这意味着某个函数正在某处等待执行。通过流发出值的方式是通过调用 ~~add~~~~streams~~ 控制器。

```
controller.add(12);
```

调用该函数时，将执行上一节中提供的回调。

```
Value from controller: 12
```

这就是Stream的所有基础知识。现在让我们继续管理Stream。

管理Stream

监听调用返回`StreamSubscription`Stream类型 。这可以用来管理Stream的订阅。订阅的最常见用法是在不再需要接收数据时取消监听。对Stream的订阅将一直处于活动状态，直到整个内存（通常是应用程序的整个生命周期）被破坏为止。在某些情况下这是完全可以的，而在其他情况下则不是。

订阅Stream后，~~您必须将其取消后，~~可以将其存储在`SteamSubscription` subscription 订阅

```
StreamSubscription<double> streamSubscription =
stream.listen((value) {
  print('Value from controller: $value');
});
```

这将为您提供注册回调的订阅对象。

取消 stream

当您要执行此操作时，让我们过去。在 flutter 中，流通常与一起使用，`StreamBuilder`一旦窗口小部件销毁，该流就会在内部为您管理和取消订阅 Stream。要遵循的一个好规则是，当您订阅 Stream 时，保留订阅并在 dispose 方法中编写代码以调用 cancel。如果您的流在应用程序的整个过程中都需要保持活动状态，则无需在处置时或不需要时取消该 Stream。

```
streamSubscription.cancel();
```

常见 Stream 错误

当开发人员在 dart 中使用 Streams 时，很常见的一件事是“Stream 已订阅”消息。很多人认为这是因为存在有效的订阅和取消操作，可以消除该错误，但这不是事实。让我们看看如何自己创建此异常，然后我们将找出解决方法。

```
stream.listen((value) {
  print('1st Sub: $value');
});
stream.listen((value) {
  print('2nd Sub: $value');
});
```

这将引发错误“错误状态：流已被收听”。现在，即使您取消第一个订阅并再次订阅，您仍然会收到此错误，这是设计使然。

```
streamSubscription = stream.listen((value) {
    print('1st Sub: $value');
  });
  await streamSubscription.cancel();
  stream.listen((value) {
    print('2nd Sub: $value');
  });
```

上面的代码仍将引发“错误状态”stream错误。这样做的原因是因为stream有两种类型：

单一订阅流：与一系列事件结合使用，这些事件是一个较大的整体的一部分。诸如读取文件或Web请求之类的事情。为了确保首先订阅的订阅者以正确的顺序获取所有正确的信息，存在一个限制，即您只能在流存在的生命周期中订阅一次。

广播stream：这种流用于单独的发射，可以一次处理一次，而无需上下文或先前事件的知识。

您可以像我一样将两个都用于单个事件，但只是对第一个订阅策略感到厌倦。当使用`StreamBuilder`in Flutter时，您很可

能总是会遇到异常，因为在构建函数调用期间，Stream将被多次订阅（这经常发生）。

修复错误状态stream错误

要解决此问题，您必须专门创建一个广播StreamController，以便将基础流构建和管理为允许多个预订的广播stream。

```
StreamController<double> controller =
StreamController<double>.broadcast();
```

重要说明—使用这些Stream类型中的任何一种并不意味着您不必管理订阅。如果您手动订阅了一个Stream，则如果有更改可能需要再次订阅它，则必须对其进行清理（取消）。多个订阅会导致内存泄漏，请确保您的代码何时超出范围或超出所处置的范围，然后根据需要重新订阅。

Manual Streams

创建 Stream的另一种方法也是一种常见方法是通过`async *`函数。这是一个函数，它将异步运行并在有新值时返回（产生）一个值，但不会停止该函数的执行。为了更有意义，让我们这样看。下面是一个Future，它将在等待1秒后返回一

个随机值。

```
Future<double> getRandomValue() async {
  var random = Random(2);
  await Future.delayed(Duration(seconds: 1));
  return random.nextDouble();
}
```

可以使用此代码，一旦关闭，您将获得一个随机值，该函数将完成执行。这意味着如果您想要另一个随机值，则必须调用并再次等待该函数。像下面。

```
var value1 = await getRandomValue();
var value2 = await getRandomValue();
```

如果您想一次调用该函数并连续从该函数获取随机值而不停止执行该怎么办？那就是`async*`产生收益的地方。让我们创建一个返回Stream的函数，每一秒钟将发出一个新的随机值。

```
Stream<double> getRandomValues() async* {
    var random = Random(2);
    while (true) {
      await Future.delayed(Duration(seconds: 1));
      yield random.nextDouble();
    }
  }
```

这称为生成器功能。它看起来与上一个相似，但是让我们看一下它们之间的区别。

- 首先要注意的是，我们现在返回一个 Stream 而不是一个 Future。这意味着我们不必等待值，而必须订阅 Stream。
- 第二个区别是 `async*` 而不是 `async`。这告诉运行时该代码应异步运行，但是即使“返回”值后执行仍将继续。
- 上次不同的是更换 `return` 用 `yield`。这基本上是一个返回函数，但不会退出该函数。相反，它在 yield 之后继续执行其余代码。 yield 产生，屈服

那么如何使用此 Stream（生成器功能）？与上述相同。

```
getRandomValues().listen((value) {
    print('1st: $value');
  });
```

这将打印出类似以下内容的内容，其中每秒钟延迟后将打印一行。

```
1st: 0.0007835923093592O4
1st: 0.2323259230935920 45
1st: 0.4560783592309359 20
1st: 0.5657835923093592O4
```

默认情况下会广播以这种方式创建的 Stream，并允许多个订阅。就所有的使用基础而言，您必须了解 Stream。在基础方面，这是您仅需了解 Stream，如何使用它们以及如何有效地管理它们而又不会引起代码错误的全部知识。了解 Streams 以及它们如何工作并需要更多功能后，您可以查看 RxDart。

翻译自：https://medium.com/flutter-community/flutter-stream-basics-for-beginners-eda23e44e32f