



Future 和 **Stream** 类是 Dart 异步编程的核心。

- **Future** 表示一个不会立即完成的计算过程。与普通函数直接返回结果不同的是异步函数返回一个将会包含结果的 **Future**。该 **Future** 会在结果准备好时通知调用者。
- **Stream** 是一系列异步事件的序列。其类似于一个异步的 **Iterable**，不同的是当你向 **Iterable** 获取下一个事件时它会立即给你，但是 **Stream** 则不会立即给你而是在它准备好时告诉你。

接收 Stream 事件

Stream 可以通过许多方式创建 所有的创建方式都可以使用 异步 *for* 循环（通常我们直接称之为 **await for**）来迭代 **Stream** 中的事件

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

接收整型事件流中的每一个事件并将它们相加，然后返回（被 **Future** 包裹）相加后的整型值。当循环体结束时，函数会暂停直到下一个事件到达或 **Stream** 完成。

内部使用 **await for** 循环的函数需要使用 **async** 关键字标记。

```
import 'dart:async';  
  
Future<int> sumStream(Stream<int> stream) async {  
  var sum = 0;  
  await for (var value in stream) {  
    sum += value;  
  }  
  return sum;  
}  
  
Stream<int> countStream(int to) async* {  
  for (int i = 1; i <= to; i++) {  
    yield i;  
  }  
}  
  
main() async {  
  var stream = countStream(10);  
  var sum = await sumStream(stream);  
  print(sum); // 55  
}
```

async* 函数生成一个简单的整型 **Stream**

错误事件

当 **Stream** 再也没有需要处理的事件时会变为完成状态，与此同时，调用者可以像接收到新事件回调那样接收 **Stream** 完成的事件回调。当使用 **await for** 循环读取事件时，循环会在 **Stream** 完成时停止。

有时在 **Stream** 完成前会出现错误；比如从远程服务器获取文件时出现网络请求失败，或者创建事件时出现 bug，尽管错误总是会有可能存在，但它出现时应该告知使用者。

Stream 可以像提供数据事件那样提供错误事件。大多数 **Stream** 会在第一次错误出现后停止，但其也可以提供多次错误并可以在在出现错误后继续提供数据事件。在本篇文档中我们只讨论 **Stream** 最多出现并提供一次错误事件的情况。

当使用 **await for** 读取 **Stream** 时，如果出现错误，则由循环语句抛出，同时循环结束。你可以使用 **try-catch** 语句捕获错误。下面的示例会在循环迭代到参数值等于 4 时抛出一个错误：

```
import 'dart:async';  
  
Future<int> sumStream(Stream<int> stream) async {  
  var sum = 0;  
  try {  
    await for (var value in stream) {  
      sum += value;  
    }  
  } catch (e) {  
    return -1;  
  }  
  return sum;  
}  
  
Stream<int> countStream(int to) async* {  
  for (int i = 1; i <= to; i++) {  
    if (i == 4) {  
      throw new Exception('Intentional exception');  
    } else {  
      yield i;  
    }  
  }  
}  
  
main() async {  
  var stream = countStream(10);  
  var sum = await sumStream(stream);  
  print(sum); // -1  
}
```

Stream 的两种类型

Stream 有两种类型。

Single-Subscription 类型的 Stream

最常见的类型是一个 Stream 只包含了某个众多事件序列的一个。而这些事件需要按顺序提供并且不能丢失。当你读取一个文件或接收一个网页请求时就需要使用这种类型的 Stream。

这种 Stream 只能设置一次监听。重复设置则会丢失原来的事件，而导致你所监听到的剩余其它事件毫无意义。当你开始监听时，数据将以块的形式提供和获取。

Broadcast 类型的 Stream

另一种流是针对单个消息的，这种流可以一次处理一个消息。例如可以将其用于浏览器的鼠标事件。

你可以在任何时候监听这种 Stream，且在此之后你可以获取到任何触发的事件。这种流可以在同一时间设置多个不同的监听器同时监听，同时你也可以在取消上一个订阅后再次对其发起监听。

listen() 方法

最后一个重要的方法是 listen()。这是一个“底层”方法，其它所有的 Stream 方法都根据 listen() 方法定义。

```
StreamSubscription<T> listen(void Function(T event) onData,
    {Function onError, void Function() onDone, bool cancelOnError});
```

你只需继承 Stream 类并实现 listen() 方法来创建一个 Stream 类型的子类。Stream 类中所有其它的方法都依赖于对 listen() 方法的调用。


listen() 方法可以让你对一个 Stream 进行监听。在你对一个 Stream 进行监听前，它只不过是个性对象，该对象描述了你想要查看的事件。当你对其进行监听后，其会返回一个 StreamSubscription 对象，该对象用以表示一个生产事件的活跃的 Stream。这与 Iterable 对象的实现方式类似，不同的是 Iterable 对象可返回迭代器并可以进行真实的迭代操作。

Stream 允许你暂停、继续甚至完全取消一个订阅。你也可以为其设置一个回调，该回调会在每一个数据事件、错误事件以及 Stream 自身关闭时通知调用者。

分类: flutter

标签: flutter

好文要顶 关注我 收藏该文



肖无情

粉丝 - 2 关注 - 5


+加关注

0 推荐 0 反对

« 上一篇： 02-10 flutter hintText和光标不对齐

» 下一篇： 02-12 flutter Hot Reload是怎么做到的? .md

刷新评论 刷新页面 返回顶部

 登录后才能查看或发表评论，立即 登录 或者 逛逛 博客园首页

【推荐】阿里云新人特惠，爆款云服务器2核4G低至0.46元/天

编辑推荐:

- gRPC 入门与实操 (.NET 篇)
- dotnet 代码优化 聊聊逻辑圈复杂度
- 一个棘手的生产问题，但是我写出来之后，就是你的了
- 你可能不知道的容器镜像安全实践
- .Net 6 使用 Consul 实现服务注册与发现

阅读排行:

- Redux与前端表格施展“组合拳”，实现大屏展示应用的交互增强
- gRPC入门与实操(.NET篇)
- 博客园主题修改分享 - 过年篇
- 如何优雅地校验后端接口数据，不做前端背锅侠
- 产品与研发相处之道