Flutter之Stream、StreamController的使用(二)

下面的例子展示了如何使用 `StreamController` 在 `Flutter` 应用中创建一个简单的计数器。其中用到了一个 `StreamBuilder` 组件来监听 `StreamController` 的流，并且在新数据发送到流时更新 UI。

```
import 'dart:async';

import 'package:flutter/material.dart';

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
      home: CounterScreen(),
    );
  }
}

class CounterScreen extends StatefulWidget {
  const CounterScreen({super.key});

  @override
  _CounterScreenState createState() =>
_CounterScreenState();
}
```

```dart
class _CounterScreenState extends State<CounterScreen> {
  final StreamController<Map<String,dynamic>>
_counterController =
StreamController<Map<String,dynamic>>();

  int _counter = 0;
  Map<String,dynamic> _map = {"test":0};

  @override
  void dispose() {
    _counterController.close(); // 关闭 StreamController 以释
放资源
    super.dispose();
  }

  // 增加计数并将新值发送到流中
  void _incrementCounter() {
    _counter = _counter + 1;
    _counterController.sink.add({"test":_counter});

  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text("Counter App")),
      body: Center(
        child: StreamBuilder<Map<String,dynamic>>(
          stream: _counterController.stream,
          initialData: _map, // 初始数据
          builder: (context, snapshot) {
            return Text(
              'Counter Value: ${snapshot.data?["test"]}',
              style: TextStyle(fontSize: 20),
```

```
            );
          },
        ),
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: _incrementCounter,
        child: const Icon(Icons.add),
      ),
    );
  }
}
```

在 `Flutter` 中，当我们要管理可变状态时，我们通常会使用 `StatefulWidget`。但是如果我们在 `StatelessWidget` 中使用外部状态管理（比如 `StreamController`），我们也能够在没有 `StatefulWidget` 的情况下响应状态变化。下面的示例中，我们将使用一个 `StatelessWidget` 和 `StreamController` 来实现计数器的功能：

```
import 'dart:async';
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  final StreamController<Map<String, dynamic>>
_counterController = StreamController<Map<String,
dynamic>>.broadcast(); // 创建一个广播控制器
  int _counter = 0;

  // 增加计数并将新值发送到流中
  void incrementCounter() {
```

```
    _counter = _counter + 1;
    _counterController.sink.add({"test": _counter});
  }

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: Text("Counter App")),
        body: Center(
          child: StreamBuilder<Map<String, dynamic>>(
            stream: _counterController.stream,
            initialData: {"test": 0}, // 初始数据
            builder: (context, snapshot) {
              return Text(
                'Counter Value: ${snapshot.data?["test"]}',
                style: TextStyle(fontSize: 20),
              );
            },
          ),
        ),
        floatingActionButton: FloatingActionButton(
          onPressed: incrementCounter,
          child: Icon(Icons.add),
        ),
      ),
    );
  }
}
```

在上面的代码中，我们创建了一个广播型的 `StreamController(.broadcast())`，这允许流有多个监听器。`incrementCounter` 方法使计数器增加，并通过流控制器的 `sink` 将新值发送到流

中。

我们使用 `StreamBuilder` 来监听流，并在流中的数据发生变化时重新构建界面。

注意点

我们在这里没有使用 `dispose` 来关闭 `StreamController`，因为在 `StatelessWidget` 中没有 `dispose` 方法。在实际应用中，请确保在适当的地方关闭流，以防止内存泄漏。例如，你可以在页面切换时关闭它，或使用其他状态管理方法（如 `GetX` 或 `Riverpod`）来更安全地管理流的生命周期。