【–Flutter/Dart 语法补遗–】 sync* 和 async* 、yield 和yield* 、async 和 await

类别	关键字	返回类型	搭档
多元素同步	sync*	Iterable<T>	yield、yield*
单元素异步	async	Future<T>	await
多元素异步	async*	Stream<T>	yield、yield* 、await

下面就用几个emoji，认识一下这几个关键字吧

一、多元素同步函数生成器

1. `sync*` 和 `yield`

`sync*`是一个dart语法关键字。它标注在函数{ 之前，其方法必须返回一个 `Iterable<T>`对象

？的码为`\u{1f47f}`。下面是使用`sync*`生成后10个emoji迭代`(Iterable)`对象的方法

代码语言：javascript

复制

```
main() {
  getEmoji(10).forEach(print);
}

Iterable<String> getEmoji(int count) sync* {
  Runes first = Runes('\u{1f47f}');
  for (int i = 0; i < count; i++) {
    yield String.fromCharCodes(first.map((e) => e +
i));
  }
}
```

代码语言：javascript

复制

```
?
?
?
?
?
?
?
?
?
?
```

2、`sync*` 和 `yield*`

yield*又是何许人也？记住一点yield*后面的表达式是一个Iterable<T>对象

比如下面getEmoji方法是核心，现在想要打印每次的时间，使用getEmojiWithTime yield*之后的getEmoji(count).map((e)...便是一个可迭代对象Iterable<String>

代码语言：javascript

复制

```
main() {
  getEmojiWithTime(10).forEach(print);
}

Iterable<String> getEmojiWithTime(int count) sync*
{
  yield* getEmoji(count).map((e) => '$e -- $
{DateTime.now().toIso8601String()}');
}

Iterable<String> getEmoji(int count) sync* {
  Runes first = Runes('\u{1f47f}');
  for (int i = 0; i < count; i++) {
    yield String.fromCharCodes(first.map((e) => e +
i));
  }
}
```

复制代码

代码语言：javascript

复制

```
? -- 2020-05-20T07:01:07.163407
? -- 2020-05-20T07:01:07.169451
? -- 2020-05-20T07:01:07.169612
? -- 2020-05-20T07:01:07.169676
? -- 2020-05-20T07:01:07.169712
? -- 2020-05-20T07:01:07.169737
? -- 2020-05-20T07:01:07.169760
? -- 2020-05-20T07:01:07.169789
? -- 2020-05-20T07:01:07.169812
? -- 2020-05-20T07:01:07.169832
```

二、异步处理: async和await

async是一个dart语法关键字。它标注在函数{ 之前，其方法必须返回一个 Future<T>对象

对于耗时操作，通常用Future<T>对象异步处理，下面fetchEmoji方法模拟2s加载耗时

代码语言：javascript

复制

```
main() {
  print('程序开启--$
{DateTime.now().toIso8601String()}');
  fetchEmoji(1).then(print);
```

```
}

Future<String> fetchEmoji(int count) async{
  Runes first = Runes('\u{1f47f}');
  await Future.delayed(Duration(seconds: 2));//模拟
耗时
  print('加载结束--$
{DateTime.now().toIso8601String()}');
  return String.fromCharCodes(first.map((e) => e +
count));
}
复制代码
```

代码语言：javascript

复制

```
加载开始--2020-05-20T07:20:32.156074
加载结束--2020-05-20T07:20:34.175806
?
```

三、多元素异步函数生成器:

1.`async*`和`yield`、`await`

`async*`是一个dart语法关键字。它标注在函数{ 之前，其方法必须返回一个 `Stream<T>`对象

下面`fetchEmojis`被`async*`标注，所以返回的必然是`Stream`对象

注意被`async*`标注的函数，可以在其内部使用`yield`、`yield*`、`await`关键字

代码语言：javascript

复制

```
main() {
  fetchEmojis(10).listen(print);
}

Stream<String> fetchEmojis(int count) async*{
  for (int i = 0; i < count; i++) {
    yield await fetchEmoji(i);
  }
}

Future<String> fetchEmoji(int count) async{
  Runes first = Runes('\u{1f47f}');
  print('加载开始--$
{DateTime.now().toIso8601String()}');
  await Future.delayed(Duration(seconds: 2));//模拟
耗时
  print('加载结束--$
{DateTime.now().toIso8601String()}');
  return String.fromCharCodes(first.map((e) => e +
count));
}
复制代码
```

代码语言：javascript

复制

```
加载开始--2020-05-20T07:28:28.394205
加载结束--2020-05-20T07:28:30.409498
?
加载开始--2020-05-20T07:28:30.416714
加载结束--2020-05-20T07:28:32.419157
?
加载开始--2020-05-20T07:28:32.419388
加载结束--2020-05-20T07:28:34.423053
?
加载开始--2020-05-20T07:28:34.423284
加载结束--2020-05-20T07:28:36.428161
?
加载开始--2020-05-20T07:28:36.428393
加载结束--2020-05-20T07:28:38.433409
?
加载开始--2020-05-20T07:28:38.433647
加载结束--2020-05-20T07:28:40.436491
?
加载开始--2020-05-20T07:28:40.436734
加载结束--2020-05-20T07:28:42.440696
?
加载开始--2020-05-20T07:28:42.441255
加载结束--2020-05-20T07:28:44.445558
?
加载开始--2020-05-20T07:28:44.445801
加载结束--2020-05-20T07:28:46.448190
?
加载开始--2020-05-20T07:28:46.448432
加载结束--2020-05-20T07:28:48.452624
?
```

2.async*和yield*、await

和上面的yield*同理，async*方法内使用yield*,其后对象必须是Stream<T>对象

如下getEmojiWithTime对fetchEmojis流进行map转换，前面需要加yield*

代码语言：javascript

复制

```
main() {
  getEmojiWithTime(10).listen(print);
}

Stream<String> getEmojiWithTime(int count) async* {
  yield* fetchEmojis(count).map((e) => '$e -- $
{DateTime.now().toIso8601String()}');
}

Stream<String> fetchEmojis(int count) async*{
  for (int i = 0; i < count; i++) {
    yield await fetchEmoji(i);
  }
}

Future<String> fetchEmoji(int count) async{
  Runes first = Runes('\u{1f47f}');
```

```
  await Future.delayed(Duration(seconds: 2));//模拟耗时
  return String.fromCharCodes(first.map((e) => e + count));
}
复制代码
```

代码语言：javascript

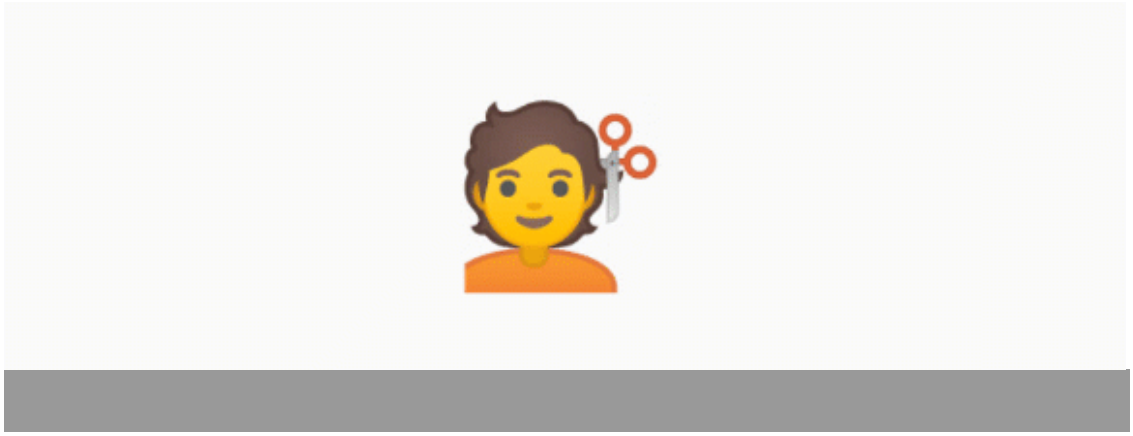
复制

```
? -- 2020-05-20T07:35:09.461624
? -- 2020-05-20T07:35:11.471223
? -- 2020-05-20T07:35:13.476712
? -- 2020-05-20T07:35:15.482848
? -- 2020-05-20T07:35:17.489429
? -- 2020-05-20T07:35:19.491214
? -- 2020-05-20T07:35:21.497086
? -- 2020-05-20T07:35:23.500867
? -- 2020-05-20T07:35:25.505379
? -- 2020-05-20T07:35:27.511723
```

四、Stream的使用–StreamBuilder

Stream在组件层面最常用的就数`StreamBuilder`,本文只是简单用一下，以后会有专文

`StreamBuilder`组件使用的核心就是，它接受一个`Stream对象`,根据`builder函数`在流元素的不同状态下构建不同的界面。



1.顶部组件

代码语言：javascript

复制

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        title: 'Flutter Demo',
        theme: ThemeData(
          primarySwatch: Colors.blue,
        ),
        home: Scaffold(
          appBar: AppBar(),
          body: HomePage(),
        ));
```

```
  }
}
```

2.StreamBuilder组件的使用

代码语言：javascript

复制

```
class HomePage extends StatefulWidget {

  @override
  _HomePageState createState() => _HomePageState();
}

class _HomePageState extends State<HomePage> {
   Stream<String> _stream;
  @override
  void initState() {
    super.initState();
    _stream= fetchEmojis(10);
  }

  @override
  Widget build(BuildContext context) {
    return Center(
      child: StreamBuilder<String>(
        builder: _buildChildByStream,
        stream: _stream,
```

```
      ),
    );
  }

  Widget _buildChildByStream(
      BuildContext context, AsyncSnapshot<String>
snapshot) {
    switch(snapshot.connectionState){
      case ConnectionState.none:
        break;
      case ConnectionState.waiting:
        return CircularProgressIndicator();
        break;
      case ConnectionState.active:
        return Text(snapshot.requireData,style:
TextStyle(fontSize: 60));
        break;
      case ConnectionState.done:
        return Text('Stream Over--$
{snapshot.requireData}',style: TextStyle(fontSize:
30),);
        break;
    }
    return Container();
  }

   Stream<String> fetchEmojis(int count) async* {
     for (int i = 0; i < count; i++) {
       yield await fetchEmoji(i);
```

```
    }
  }

  Future<String> fetchEmoji(int count) async {
    Runes first = Runes('\u{1f47f}');
    await Future.delayed(Duration(seconds: 1)); //
模拟耗时
    return String.fromCharCodes(first.map((e) => e
+ count));
  }
}
```
复制代码

题外话:

如果你使用过`flutter_bloc`,会用到`async*`,现在再来看，是不是更清楚了一点。

代码语言：javascript

复制

```
class CounterBloc extends Bloc<CounterEvent, int> {
  @override
  int get initialState => 0;

  @override
  Stream<int> mapEventToState(CounterEvent event)
async* {
    switch (event) {
```

```
      case CounterEvent.decrement:
        yield state - 1;
        break;
      case CounterEvent.increment:
        yield state + 1;
        break;
    }
  }
}
```