创建一个基本list

基本List

显示数据列表是移动应用程序常见的需求。Flutter包含的`ListView` Widget，使列表变得轻而易举！

创建一个ListView

使用标准`ListView`构造函数非常适合仅包含少量条目的列表。我们使用内置的`ListTile` Widget来作为列表项。

```
new ListView(
  children: <Widget>[
    new ListTile(
      leading: new Icon(Icons.map),
      title: new Text('Maps'),
    ),
    new ListTile(
      leading: new Icon(Icons.photo_album),
      title: new Text('Album'),
    ),
    new ListTile(
      leading: new Icon(Icons.phone),
      title: new Text('Phone'),
    ),
  ],
);
```

完整的例子

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    final title = 'Basic List';

    return new MaterialApp(
      title: title,
      home: new Scaffold(
        appBar: new AppBar(
          title: new Text(title),
        ),
```

```
        body: new ListView(
          children: <Widget>[
            new ListTile(
              leading: new Icon(Icons.map),
              title: new Text('Map'),
            ),
            new ListTile(
              leading: new Icon(Icons.photo),
              title: new Text('Album'),
            ),
            new ListTile(
              leading: new Icon(Icons.phone),
              title: new Text('Phone'),
            ),
          ],
        ),
      ),
    );
  }
}
```

创建一个水平list

有时，您可能想要创建一个水平滚动（而不是垂直滚动）的列表。ListView本身就支持水平list。

在创建ListView时，设置`scrollDirection`为水平方向以覆盖默认的垂直方向。

```
new ListView(
  // This next line does the trick.
  scrollDirection: Axis.horizontal,
  children: <Widget>[
    new Container(
      width: 160.0,
      color: Colors.red,
    ),
    new Container(
      width: 160.0,
      color: Colors.blue,
    ),
    new Container(
      width: 160.0,
```

```
      color: Colors.green,
    ),
    new Container(
      width: 160.0,
      color: Colors.yellow,
    ),
    new Container(
      width: 160.0,
      color: Colors.orange,
    ),
  ],
)
```

完整的例子

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    final title = 'Horizontal List';

    return new MaterialApp(
      title: title,
      home: new Scaffold(
        appBar: new AppBar(
          title: new Text(title),
        ),
        body: new Container(
          margin: new EdgeInsets.symmetric(vertical: 20.0),
          height: 200.0,
          child: new ListView(
            scrollDirection: Axis.horizontal,
            children: <Widget>[
              new Container(
                width: 160.0,
                color: Colors.red,
              ),
              new Container(
                width: 160.0,
                color: Colors.blue,
              ),
              new Container(
                width: 160.0,
```

```
                color: Colors.green,
              ),
              new Container(
                width: 160.0,
                color: Colors.yellow,
              ),
              new Container(
                width: 160.0,
                color: Colors.orange,
              ),
            ],
          ),
        ),
      ),
    ),
  );
}
}
```

使用长列表

标准的ListView构造函数适用于小列表。 为了处理包含大量数据的列表，最好使用ListView.builder构造函数。

ListView的构造函数需要一次创建所有项目，但ListView.builder的构造函数不需要，它将在列表项滚动到屏幕上时创建该列表项。

1. 创建一个数据源

首先，我们需要一个数据源来。例如，您的数据源可能是消息列表、搜索结果或商店中的产品。大多数情况下，这些数据将来自互联网或数据库。

在这个例子中，我们将使用List.generate构造函数生成拥有10000个字符串的列表

```
final items = new List<String>.generate(10000, (i) => "Item $i");
```

2. 将数据源转换成Widgets

为了显示我们的字符串列表，我们需要将每个字符串展现为一个Widget ！

这正是ListView.builder发挥作用的地方。在我们的例子中，我们将每一行显示一个字符串：

```
new ListView.builder(
  itemCount: items.length,
  itemBuilder: (context, index) {
```

```
    return new ListTile(
      title: new Text('${items[index]}'),
    );
  },
);
```

完整的例子

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

void main() {
  runApp(new MyApp(
    items: new List<String>.generate(10000, (i) => "Item $i"),
  ));
}

class MyApp extends StatelessWidget {
  final List<String> items;

  MyApp({Key key, @required this.items}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    final title = 'Long List';

    return new MaterialApp(
      title: title,
      home: new Scaffold(
        appBar: new AppBar(
          title: new Text(title),
        ),
        body: new ListView.builder(
          itemCount: items.length,
          itemBuilder: (context, index) {
            return new ListTile(
              title: new Text('${items[index]}'),
            );
          },
        ),
      ),
    );
  }
}
```

使用不同类型的子项创建列表

我们经常需要创建显示不同类型内容的列表。例如，我们可能正在制作一个列表，其中显示一个标题，后面跟着与该标题相关的几个子项，再后面是另一个标题，等等。我们如何用Flutter创建这样的结构？

步骤

1. 使用不同类型的数据创建数据源
2. 将数据源转换为Widgets列表

1. 使用不同类型的数据创建数据源

条目(子项)类型

为了表示列表中的不同类型的条目，我们需要为每个类型的条目定义一个类。

在这个例子中，我们将在一个应用程序上显示一个标题，后面跟着五条消息。因此，我们将创建三个类：`ListItem`、`HeadingItem`、和`MessageItem`。

```
// The base class for the different types of items the List can contain
abstract class ListItem {}

// A ListItem that contains data to display a heading
class HeadingItem implements ListItem {
  final String heading;

  HeadingItem(this.heading);
}

// A ListItem that contains data to display a message
class MessageItem implements ListItem {
  final String sender;
  final String body;

  MessageItem(this.sender, this.body);
}
```

创建Item列表

大多数时候，我们会从互联网或本地数据库中读取数据，并将该数据转换成item的列表。

对于这个例子，我们将生成一个Item列表来处理。该列表将包含一个标题、后跟五条消息，然后重复。

```
final items = new List<ListItem>.generate(
  1200,
  (i) => i % 6 == 0
      ? new HeadingItem("Heading $i")
      : new MessageItem("Sender $i", "Message body $i"),
);
```

2. 将数据源转换为Widgets列表

为了将每个item转换为Widget，我们将使用`ListView.builder`构造函数。

通常，我们需要提供一个`builder`函数来检查我们正在处理的item类型，并返回该item类型对应的Widget。

在这个例子中，使用`is`关键字来检查我们正在处理的item的类型，这个速度很快，并会自动将每个item转换为适当的类型。 但是，如果您更喜欢另一种模式，也有不同的方法可以解决这个问题！

```
new ListView.builder(
  // Let the ListView know how many items it needs to build
  itemCount: items.length,
  // Provide a builder function. This is where the magic happens! We'll
  // convert each item into a Widget based on the type of item it is.
  itemBuilder: (context, index) {
    final item = items[index];

    if (item is HeadingItem) {
      return new ListTile(
        title: new Text(
          item.heading,
          style: Theme.of(context).textTheme.headline,
        ),
      );
    } else if (item is MessageItem) {
      return new ListTile(
        title: new Text(item.sender),
        subtitle: new Text(item.body),
      );
    }
  },
);
```

完整的例子

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

void main() {
```

```
  runApp(new MyApp(
    items: new List<ListItem>.generate(
      1000,
      (i) => i % 6 == 0
          ? new HeadingItem("Heading $i")
          : new MessageItem("Sender $i", "Message body $i"),
    ),
  ));
}

class MyApp extends StatelessWidget {
  final List<ListItem> items;

  MyApp({Key key, @required this.items}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    final title = 'Mixed List';

    return new MaterialApp(
      title: title,
      home: new Scaffold(
        appBar: new AppBar(
          title: new Text(title),
        ),
        body: new ListView.builder(
          // Let the ListView know how many items it needs to build
          itemCount: items.length,
          // Provide a builder function. This is where the magic happens! We'll
          // convert each item into a Widget based on the type of item it is.
          itemBuilder: (context, index) {
            final item = items[index];

            if (item is HeadingItem) {
              return new ListTile(
                title: new Text(
                  item.heading,
                  style: Theme.of(context).textTheme.headline,
                ),
              );
            } else if (item is MessageItem) {
              return new ListTile(
                title: new Text(item.sender),
                subtitle: new Text(item.body),
```

```
                );
              }
            },
          ),
        ),
      );
    }
}

// The base class for the different types of items the List can contain
abstract class ListItem {}

// A ListItem that contains data to display a heading
class HeadingItem implements ListItem {
  final String heading;

  HeadingItem(this.heading);
}

// A ListItem that contains data to display a message
class MessageItem implements ListItem {
  final String sender;
  final String body;

  MessageItem(this.sender, this.body);
}
```