

处理点击

我们不仅希望向用户展示信息，还希望我们的用户与我们的应用互动！那么，我们如何响应用户基本操作，如点击和拖动？在Flutter中我们可以使用

[GestureDetector](#) Widget！

假设我们想要创建一个自定义按钮，当点击时显示一个SnackBar。我们如何解决这个问题？

步骤

1. 创建一个button。
2. 把它包装在 `GestureDetector` 中并提供一个 `onTap` 回调。

```
// Our GestureDetector wraps our button
new GestureDetector(
  // When the child is tapped, show a snackbar
  onTap: () {
    final snackBar = new SnackBar(content: new Text("Tap"));

    Scaffold.of(context).showSnackBar(snackBar);
  },
  // Our Custom Button!
  child: new Container(
    padding: new EdgeInsets.all(12.0),
    decoration: new BoxDecoration(
      color: Theme.of(context).buttonColor,
      borderRadius: new BorderRadius.circular(8.0),
    ),
    child: new Text('My Button'),
  ),
);
```

注意

1. 如果您想将Material 水波效果添加到按钮中，请参阅[添加Material 触摸水波](#)。
2. 虽然我们已创建了一个自定义按钮来演示这些概念，但Flutter也提供了一些其它开箱即用的按钮：[RaisedButton](#)、[FlatButton](#)和[CupertinoButton](#)。

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 = 'Gesture Demo';

    return new MaterialApp(
      title: title,
      home: new MyHomePage(title: title),
    );
  }
}

class MyHomePage extends StatelessWidget {
  final String title;

  MyHomePage({Key key, this.title}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return new Scaffold(
      appBar: new AppBar(
        title: new Text(title),
      ),
      body: new Center(child: new MyButton()),
    );
  }
}

class MyButton extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    // Our GestureDetector wraps our button
    return new GestureDetector(
      // When the child is tapped, show a snackbar
      onTap: () {
        final snackBar = new SnackBar(content: new Text("Tap"));

        Scaffold.of(context).showSnackBar(snackBar);
      },
    );
  }
}
```

```
// Our Custom Button!
child: new Container(
  padding: new EdgeInsets.all(12.0),
  decoration: new BoxDecoration(
    color: Theme.of(context).buttonColor,
    borderRadius: new BorderRadius.circular(8.0),
  ),
  child: new Text('My Button'),
),
);
}
```

添加Material触摸水波效果

在设计应遵循Material Design指南的应用程序时，我们希望在点击时将水波动画添加到Widgets。

Flutter提供了[InkWell](#)Widget来实现这个效果。

步骤

1. 创建一个可点击的Widget。
2. 将它包裹在一个[InkWell](#)中来管理点击回调和水波动画。

```
// The InkWell Wraps our custom flat button Widget
new InkWell(
  // When the user taps the button, show a snackbar
  onTap: () {
    Scaffold.of(context).showSnackBar(new SnackBar(
      content: new Text('Tap'),
    ));
  },
  child: new Container(
    padding: new EdgeInsets.all(12.0),
    child: new Text('Flat Button'),
  ),
);
```

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 = 'InkWell Demo';

    return new MaterialApp(
      title: title,
      home: new MyHomePage(title: title),
    );
  }
}

class MyHomePage extends StatelessWidget {
  final String title;

  MyHomePage({Key key, this.title}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return new Scaffold(
      appBar: new AppBar(
        title: new Text(title),
      ),
      body: new Center(child: new MyButton()),
    );
  }
}

class MyButton extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    // The InkWell Wraps our custom flat button Widget
    return new InkWell(
      // When the user taps the button, show a snackbar
      onTap: () {
        Scaffold.of(context).showSnackBar(new SnackBar(
          content: new Text('Tap'),
        ));
      },
      child: new Container(
        padding: new EdgeInsets.all(12.0),
        child: new Text('Flat Button'),
      ),
    );
  }
}

```

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

实现滑动关闭

“滑动删除”模式在移动应用中很常见。例如，如果我们正在编写一个电子邮件应用程序，我们希望允许我们的用户在列表中滑动电子邮件。当他们这样做时，我们需要将该条目从收件箱移至垃圾箱。

Flutter通过提供[Dismissable](#) Widget 使这项任务变得简单。

步骤

1. 创建item列表。
2. 将每个item包装在一个[Dismissable](#) Widget中。
3. 提供滑动背景提示。

1. 创建item列表

第一步是创建一个我们可以滑动的列表。有关如何创建列表的更多详细说明，请按照[使用长列表](#)进行操作。

创建数据源

在我们的例子中，我们需要20个条目。为了简单起见，我们将生成一个字符串列表。

```
final items = new List<String>.generate(20, (i) => "Item ${i + 1}");
```

将数据源转换为List

首先，我们将简单地在屏幕上的列表中显示每个项目(先不支持滑动)。

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

将每个item包装在Dismissible Widget中

现在我们希望让用户能够将条目从列表中移除，用户删除一个条目后，我们需要从列表中删除该条目并显示一个Snackbar。在实际的场景中，您可能需要执行更复杂的逻辑，例如从Web服务或数据库中删除条目。

这是我们就可以使用`Dismissable`。在下面的例子中，我们将更新`itemBuilder`函数以返回一个`DismissableWidget`。

```
new Dismissible(  
  // Each Dismissible must contain a Key. Keys allow Flutter to  
  // uniquely identify Widgets.  
  key: new Key(item),  
  // We also need to provide a function that will tell our app  
  // what to do after an item has been swiped away.  
  onDismissed: (direction) {  
    // Remove the item from our data source  
    items.removeAt(index);  
  
    // Show a snackbar! This snackbar could also contain "Undo" actions.  
    Scaffold.of(context).showSnackBar(  
      new SnackBar(content: new Text("$item dismissed")));  
  },  
  child: new ListTile(title: new Text('$item')),  
);
```

3. 提供滑动背景提示

现在，我们的应用程序将允许用户从列表中滑动项目，但用户并不知道滑动后做了什么，所以，我们需要告诉用户滑动操作会移除条目。为此，我们将在滑动条目时显示指示。在下面的例子中，我们通过将背景设置为红色表示为删除操作。

为此，我们为`Dismissable`提供一个`background`参数。

```
new Dismissible(  
  // Show a red background as the item is swiped away  
  background: new Container(color: Colors.red),  
  key: new Key(item),  
  onDismissed: (direction) {  
    items.removeAt(index);  
  
    Scaffold.of(context).showSnackBar(  
      new SnackBar(content: new Text("$item dismissed")));  
  },  
  child: new ListTile(title: new Text('$item')),  
);
```

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(20, (i) => "Item ${i + 1}"),
  ));
}

```

```

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

```

```

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

```

```

  @override

```

```

  Widget build(BuildContext context) {
    final title = 'Dismissing Items';

```

```

    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) {
            final item = items[index];

```

```

            return new Dismissible(
              // Each Dismissible must contain a Key. Keys allow Flutter to
              // uniquely identify Widgets.
              key: new Key(item),
              // We also need to provide a function that will tell our app
              // what to do after an item has been swiped away.
              onDismissed: (direction) {
                items.removeAt(index);

```

```

                Scaffold.of(context).showSnackBar(
                  new SnackBar(content: new Text("$item dismissed")));
              },

```

```

              // Show a red background as the item is swiped away
              background: new Container(color: Colors.red),
              child: new ListTile(title: new Text('$item')),
            );
          },
        ),
      ),
    );

```

