一、StatefulWidget简介

StatefulWidget是UI可以变化的Widget

代码范例实现如下：

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

void main() => runApp(TestApp ("Hello World"));
class TestApp extends StatefulWidget {
 String content;

 TestApp(this.content);

 @override
  State<StatefulWidget> createState() {
  // TODO: implement createState
  return _TestState();
  }
}

class _TestState extends State<TestApp> {
  bool isShowText = true;

  void increment() {
  setState(() {
  widget.content += "d";
  });
  }
```

```
24
25  @override
26  Widget build(BuildContext context) {
27  // TODO: implement build
28  return new MaterialApp(
29  title: "Flutter App",
30  home: Scaffold(
31  appBar: AppBar(
32  title: Text("StatefulWidget"),
33  ),
34  body: Center(
35  child: GestureDetector(
36  child: isShowText ? Text(widget.content) : null,
37  onTap: increment,
38  ),
39  ),
40  ),
41  );
42  }
43 }
```

从代码逻辑来看，StatefulWidget分为两部分：

- StatefulWidget
- State

而显示组件以及方法逻辑都在State中。

1、StatefulWidget

StatefuWidget的主要功能是创建State

```
1 class TestApp extends StatefulWidget {
2  String content;
3
4  TestApp(this.content);
5
6  @override
7  State<StatefulWidget> createState() {
8  // TODO: implement createState
9  return _TestState();
10  }
11 }
```

定义一个StatefulWidget需要完成两部：

- 继承StatefulWidget

```
class TestApp extends StatefulWidget
```

- 实现createState()方法

```
@override
State<StatefulWidget> createState() {
// TODO: implement createState
return _TestState();
}
```

该方法的返回值是State<StatefulWidget>，因此该方法需要返回我们定义的State

2、State

State就是组件的状态

```
class _TestState extends State<TestApp> {
 bool isShowText = true;

 void increment() {
 setState(() {
 widget.content += "d";
 });
 }

  @override
  Widget build(BuildContext context) {
  // TODO: implement build
  return new MaterialApp(
  title: "Flutter App",
  home: Scaffold(
  appBar: AppBar(
  title: Text("StatefulWidget"),
  ),
  body: Center(
  child: GestureDetector(
  child: isShowText ? Text(widget.content) : null,
  onTap: increment,
  ),
  ),
  ),
  );
  }
}
```

定义State需要完成三步：

- 继承State，State中的泛型就是我们之前定义的StatefulWidget组件

```
class _TestState extends State<TestApp>
```

- 实现build方法

```
@override
 Widget build(BuildContext context) {
 // TODO: implement build
 return new MaterialApp(
 title: "Flutter App",
 home: Scaffold(
 appBar: AppBar(
 title: Text("StatefulWidget"),
 ),
  body: Center(
  child: GestureDetector(
  child: isShowText ? Text(widget.content) : null,
  onTap: increment,
  ),
  ),
  ),
  );
  }
```

在该方法中我们定义需要的Widget并返回

- 更新数据后，需要更新UI，调用setState()方法

二、State

1、State的功能

State的功能主要有两部分：

- build()——构建Widget
- setState()——刷新UI

(1) build()

build方法中创建用于显示的Widget

(2) setState()

setState用于更新组件状态，刷新UI，在调用setState后，会触发State的build方法，用于强制重建Widget，重建Widget的时候，会重新绑定数据，此时数据已经改变，从而达到刷新UI的目的

2、seState源码分析

setState方法会触发State的build（）方法，从而引起组件的重建，重建的时候会重新绑定数据，从而刷新UI

setState源码

```
@protected
void setState(VoidCallback fn) {
final dynamic result = fn() as dynamic;
_element.markNeedsBuild();
}
```

其中VoidCallback是一个无参构造函数，上面我们通过

```
setState(() {
widget.content += "d";
});
```

其中

```
() {
widget.content += "d";
}
```

其实就是传入了一个无参构造函数的实现

首先会执行final dynamic result = fn() as dynamic;，其中fn就是我们传入的无参构造函数

执行该函数后会继续执行_element.markNeedsBuild();

该方法的作用是触发Widget的重建，会调用State的build方法

3、State的成员变量

State有是三个成员变量：

- widget
- context
- mounted

（1）widget

widget是State的成员变量