Flutter系列之Widget生命周期

躬行之 Lv3
2021年01月11日 00:15 · 阅读 2257

关注

> PS：你的计划很完美，但世界变化是在太快。

上篇文章学习了 Flutter 中的图片加载及源码分析，做过移动端开发的朋友都知道组件的生命周期，Flutter 中也是一样，了解和学习好 Flutter 中的组件的生命周期非常重要，同系列文章如下：

- Flutter系列之Navigator使用详解
- Flutter系列之Flex布局详解
- Flutter系列之图片加载详解

本篇文章将主要介绍 Flutter 中 Widget 的生命周期，具体如下：

1. StatelessWidget
2. StatefulWidget
3. State生命周期状态
4. State生命周期方法

StatelessWidget

StatelessWidget 派生的组件是无状态组件，无状态组件只是在构建的时候渲染一次，不支持动态变化，即无法通过其他用户操作重绘组件，只能接收传入时的参数进行构建，如下 ：

java复制代码

```
/// StatelessWidget
/// 表示无状态Widget
class StatelessSamplePage extends StatelessWidget {

  // 外部传入数据
  final String data;
  StatelessSamplePage(this.data);
```

```
  @override
  Widget build(BuildContext context) {
    return Container(
      color: Colors.lightBlue,
      child: Text(data),
    );
  }
}
```

如上传入的参数只能使用 final 修饰，否则会出现如下警告：

java复制代码

```
This class (or a class which this class inherits from) is marked as '
```

提示 Widget 被 @immutable 注解修饰，如下：

java复制代码

```
@immutable
abstract class Widget extends DiagnosticableTree {
```

此时只能使用 final 来修饰变量，Dart 中被 final 修饰的变量只能被初始化一次，这也符合 StatelessWidget 的无状态特征。

StatefulWidget

StatefulWidget 派生的组件是有状态组件，有状态的组件是支持随着数据变化多次构建 Widget 以完成动态界面的渲染，如果要实现一个实时显示当前时间的界面，显然 StatelessWidget 是不能完成的，只能使用有状态的 StatefulWidget 来实现，如下：

java复制代码

```
/// StatefulWidget
/// 表示有状态的 Widget
class StatefulSamplePage extends StatefulWidget {
  @override
  _StatefulSamplePageState createState() => _StatefulSamplePageState();
}

class _StatefulSamplePageState extends State<StatefulSamplePage> {
  DateFormat _dateFormat;
  String _currentTime;
  Timer _timer;
```

```
  @override
  void initState() {
    super.initState();
    _dateFormat = new DateFormat("yyyy-MM-dd HH:mm:ss");
    // 初始化当前时间
    _currentTime = _dateFormat.format(DateTime.now());
    // 更新时间
    _startTime();
  }

  @override
  Widget build(BuildContext context) {
    return Center(
      child: Scaffold(
        appBar: AppBar(
          title: Text("Stateful Widget sample"),
          centerTitle: true,
        ),
        body: Align(
          alignment: Alignment.topCenter,
          child: Text("当前时间：$_currentTime"),
        ),
      ),
    );
  }

  @override
  void dispose() {
    super.dispose();
    if(_timer!=null){
      _timer.cancel();
    }
  }

  _startTime() {
    const Duration duration = Duration(seconds: 1);
    _timer = Timer.periodic(
        duration,
        (Timer timer) => {
              // 刷新界面状态
              setState(() {
                _currentTime = _dateFormat.format(DateTime.now());
              })
            });
  }
}
```

效果如下：

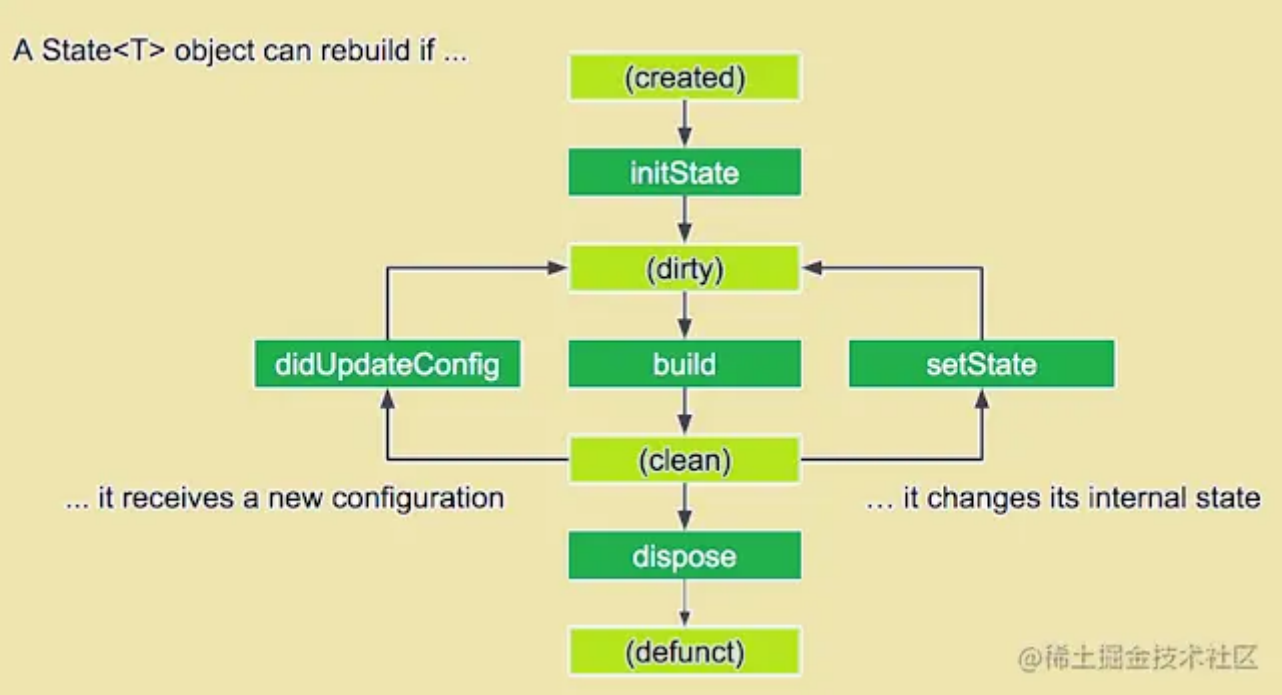
实际上，如果只是静态界面 StatelessWidget 和 StatefulWidget 是完全没有区别的，都是能够实现的，唯一区别就是 StatefulWidget 可以通过 setState 方法触发 Widget 的重新构建， State 类就是就是 Stateless -> Stateful 的桥梁。

State生命周期状态

Flutter 生命周期实际上各个组件的生命周期：

- StatelessWidget：无状态组件的生命周期只有 build 构建这个过程；
- StatefulWidget：无状态组件的生命周期指的是 State 的生命周期。

Flutter 生命周期实际上就是无状态组件的生命周期，即 State 的生命周期，如下图所示：

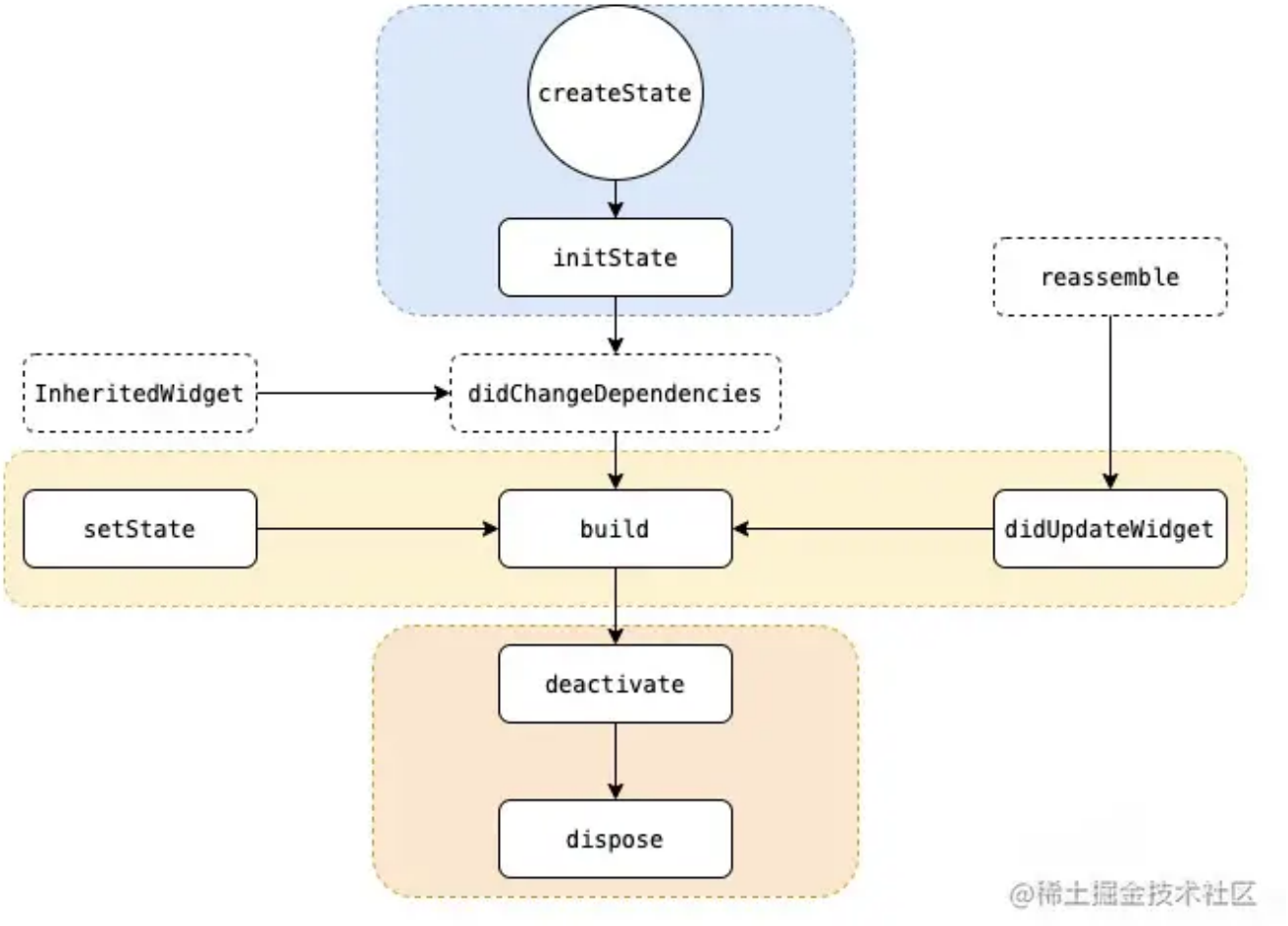


如上各个 State 的生命周期状态主要是三个：

- created：指的是 State 的创建状态，当调用 createState 方法之后就处于 created state；
- dirty：指的是当调用 setState 等方法数据发生变化，但是 Widget 还未重新构建时的状态；
- clean：指的是 Widget 构建后的状态；
- defunct：指的是 State.dispose 调用之后的状态，此时对应的 Widget 已被销毁不能够被再次构建。

State生命周期方法

有状态组件的生命周期就是 State 的生命周期，其具体调用过程与 build 触发时机如下图所示：



其生命周期方法具体含义如下：

- createState：StatefulWidget 中用于创建 State；
- initState：State 的初始化操作，如变量的初始化等；
- didChangeDependencies：initState 调用之后调用，或者使用了 InheritedWidgets 组件会被调用，其中 InheritedWidgets 可用于Flutter 状态管理；
- build：用于 Widget 的构建；
- deactivate：包含此 State 对象的 Widget 被移除之后调用，若此 Widget 被移除之后未被添加到其他 Widget 树结构中，则会继续调用 dispose 方法；
- dispose：该方法调用后释放 Widget 所占资源；
- reassemble：用于开发阶段，热重载的时候会被调用，之后会重新构建；
- didUpdateWidget：父 Widget 构建的时候子 Widget 的 didUpdateWidget 方法会被调用。

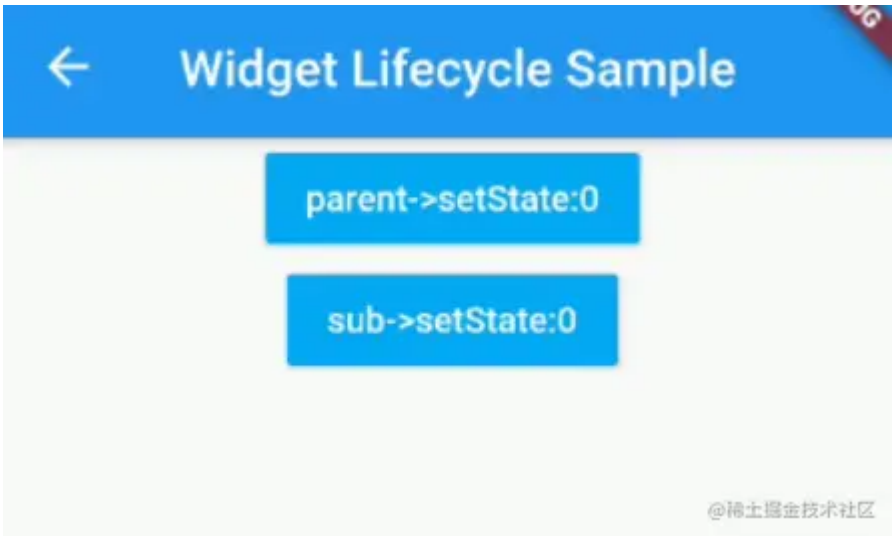
在对应的 Widget 生命周期方法中加上日志，验证上述生命周期方法的执行，父 Widget 源码如下：

java复制代码

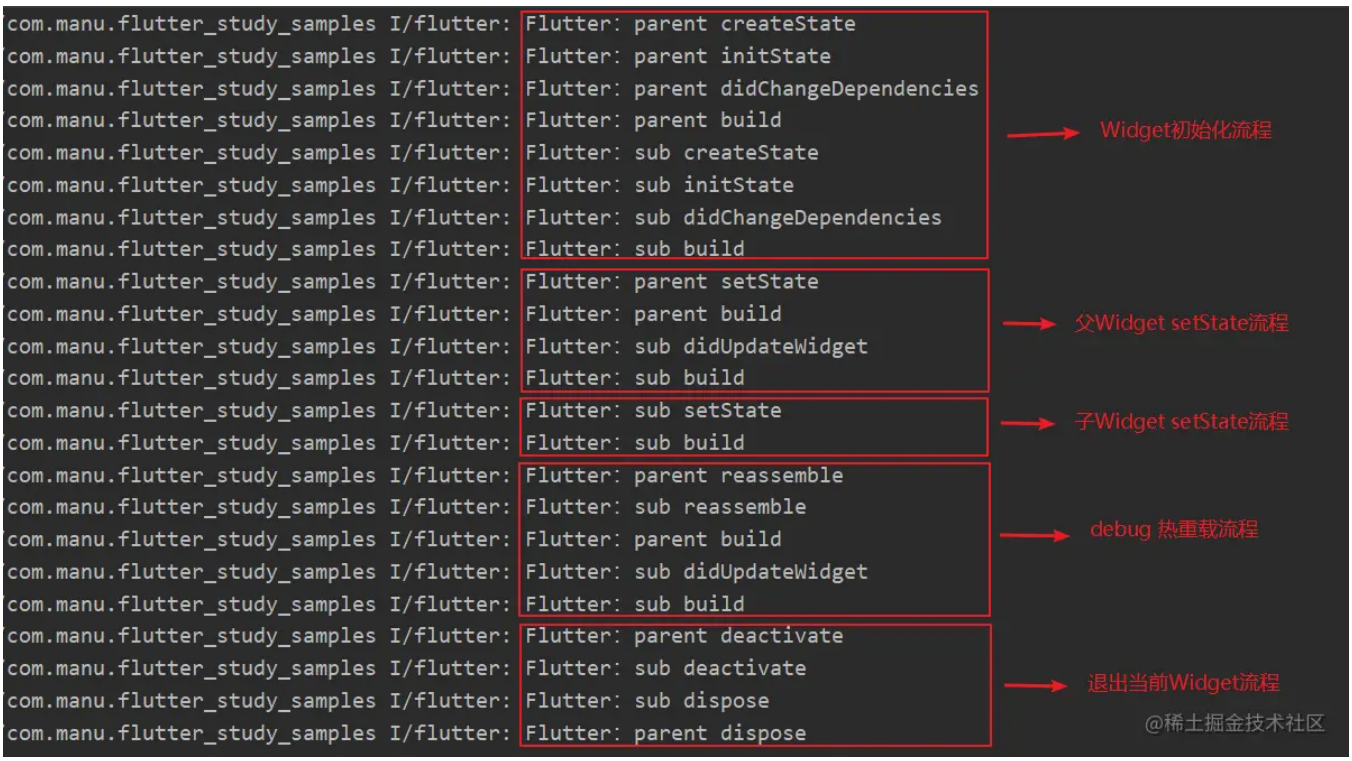
```
const String TAG = "Flutter";
/// Widget生命周期
class WidgetLifecycleSamplePage extends StatefulWidget {
  @override
  _WidgetLifecycleSamplePageState createState() {
    Log.info(TAG, "parent createState");
    return _WidgetLifecycleSamplePageState();
  }
}

class _WidgetLifecycleSamplePageState extends State<WidgetLifecycleSam

  num _count = 0;

  @override
  void initState() {
    super.initState();
    Log.info(TAG, "parent initState");
  }

  @override
  Widget build(BuildContext context) {
    Log.info(TAG, "parent build");
    return Scaffold(
      appBar: AppBar(
        title: Text("Widget Lifecycle Sample"),
        centerTitle: true,
      ),
      body: Column(
        children: <Widget>[
          Center(
            child: RaisedButton(
                textColor:Colors.white,
                color: Colors.lightBlue,
                child: Text("parent->setState:$_count",style: TextSt
                onPressed: (){
                  setState(() {
                    Log.info(TAG, "parent setState");
                    _count++;
                  });
                }),
          ),
          SubWidgetLifecycleSamplePage(),
        ],
      )
    );
  }

  @override
```

```
  void didUpdateWidget(WidgetLifecycleSamplePage oldWidget) {
    super.didUpdateWidget(oldWidget);
    Log.info(TAG, "parent didUpdateWidget");
  }

  @override
  void didChangeDependencies() {
    super.didChangeDependencies();
    Log.info(TAG, "parent didChangeDependencies");
  }

  @override
  void deactivate() {
    super.deactivate();
    Log.info(TAG, "parent deactivate");
  }

  @override
  void reassemble() {
    super.reassemble();
    Log.info(TAG, "parent reassemble");
  }

  @override
  void dispose() {
    super.dispose();
    Log.info(TAG, "parent dispose");
  }
}

/// 子Widget
class SubWidgetLifecycleSamplePage extends StatefulWidget {
  @override
  _SubWidgetLifecycleSamplePageState createState() {
    Log.info(TAG, "sub createState");
    return _SubWidgetLifecycleSamplePageState();
  }
}
```

其中子 Widget 实现与父 Widget 实现相似就不贴了，可在公众号后台回复关键字【Lifecycle】获取完成源码，上述代码显示效果如下：



其对应的生命周其方法调用如下：



分析如下：

1. Widget 初始化流程指的是进入这个 Widge 所在页面的生命周期流程，先父 Widget 后子 Widget，然后都依次调用 createState、initState、didChangeDepandencies、build 方法；
2. 当父 Widget seteState 更新数据时触发了父 Widget 的构建，故子 Widget 调用了 didUpdateWidget 方法；
3. 其他流程比如热重载、Widget 销毁都与上面的 State 生命周期方法流程图相同，这里不再赘述。

可在公众号后台回复关键字【Lifecycle】获取完成源码，更多内容见微信公众号**躬行之**。

分类： Android 标签： Flutter

文章被收录于专栏：

Flutter学习之旅

专注Flutter重点知识总结，力争有原理有实践的讲解，一起来学习...

关注专栏