TODO：

那么flutter 是如何在代码中获取到子节点的？可以获取到吗？
GlobalKey 那到底开销是有多大呢？是什么开销呢？内存吗？
我们要查找的父级Widget不存在会返回什么呢？

概念，什么是Widget?

Widget 比原生中的组件的概念更广。

原生中的组件通常是指 UI元素， 而在Flutter中，一些功能性的元素也能成为组件。
eg: 手势检测组件 GestureDetector Widget.

Widget 与 Element

Widget 更像是一个 UIWrapper。 Element 相当于原生中一个个的View。
在 Widget中，会配置一些通用的 const 配置。所有的 Element 共用这个配置。

UI树中，同一个Widget 会对应多个Element。
如果我们向UI树中添加了多个组件，其实，真正被添加到UI树上的是Widget对应的Element。
所有的Element都对应同一个Widget。

这个有没有特例呢？如果我们想让不同的Element对应不同的Widget呢？
我暂时想到的是，可以定义不同的Widget，继承自同一个Widget。
应该能够达到同样效果，未测试。

Widget 主要接口

canUpdate()

canUpdate()是一个静态方法，用于在Widget树中，判断是否使用新的Widget对象更新旧UI树所对应的Element 对象的配置。

为什么会有新的Widget 对象？该方法何时调用?
这里未做详细解释，说是之后会具体解释Widget复用的问题。

常用Widget基类

我们常用的是 StatelessWidget 或 StatefulWidget 类，实现间接集成Widget类方式，来实现Widget的继承。

StatelessWidget

StatelessWidget 相对比较简单，不需要维护状态，它是通过在build方法中嵌套其他Widget 的形式来构建UI的。
这里有几个惯例需要注意一下：

1. 命名参数中的如果有必要参数，需要传 @required标注。
2. 如果需要接收子Widget child 或者 children。 该参数一定要作为最后一个参数。
3. Widget的属性，应该尽可能的是final类型的。防止被意外改变。

Context

build 中的context参数，是 BuildContext类的实例。
这里的context 和Android中的类似，她是一个句柄，能够在当前Widget树中，当前所在的位置执行相关操作。主要有两个功能：

1. 从当前Widget开始，向上遍历Widget树。
2. 按照Widget类型，来查找父级Widget，调用父级的方法或属性。

```
# 在Widget树中，向上查找  最近的 父级 Scaffold Widget。
Scaffold scaffold = context.findAncestorWidgetOfExactType<Scaffold>();
```

这里有一个疑问，如果， 我们要查找的父级Widget不存在会返回什么呢?
估计会返回null，代码中测试一下。

注意！ context 只能从当前节点，向上遍历父节点，不能用于向下遍历子节点。这点是一定要注意的。
TODO： 那么flutter 是如何在代码中获取到子节点的？可以获取到吗?

StatefulWidget

StatefulWidget 和 StatelessWidget 不同的点，其实从名字上就能看出来。
StatefulWidget 是有一个 State状态的，会通过 createState()方法创建state。
而且我们每添加一个StatefulElement，就会调用一次createState()方法。

State

State 是用来保存StatefulWidget要维护的状态信息，比如说计数器实例中的 当前点击次数counter字段。
既然要维护状态信息，最起码 get 和 set 两个模式：

1. set
    a. 什么时候被set?
        i. 在Widget生命周期中，可以通过我们定义的各种逻辑和方法进行改变。例如我们改变了点击次数counter字段，然后调用setState()方法通知 Flutter framework状态发生改变， Flutter framework 收到消息后，会重新调用build方法，重新构建Widget树，从而达到更新UI的目的。
    b. 什么时候被get?
        i. Widget构建时可以读取 State的初始值。

State 中有两个属性：

1. widget
    a. 含义：表示与State关联的Widget，注意，这种关联不是固定的。
    b. 来源：Flutter framework 动态设置。注意，这里是动态。
    c. 为什么是动态的？绑定关系是非固定的?
        i. 因为在应用生命周期中，某一个节点Widget在 重新构建的时候可能会发生改变。
        ii. 但State实例只会在第一次插入数中时被创建。
        iii. 在重建时，如果Widget被修改了，Flutter framework 会动态设置state.widget为 新的Widget实例。
    d. 为什么设计成动态的，绑定关系是非固定?
        i. 我理解的是，这样能够保证状态State的连续性。
        ii. 虽然Widget被重新创建了，但新的Widget仍然需要旧widget所处的状态。
        iii. 因此直接将当前state和新的widget绑定在一起，避免的state的重复创建，完成state的复用。
2. context
    a. 与 StatelessWidget 中的context类似。

State生命周期

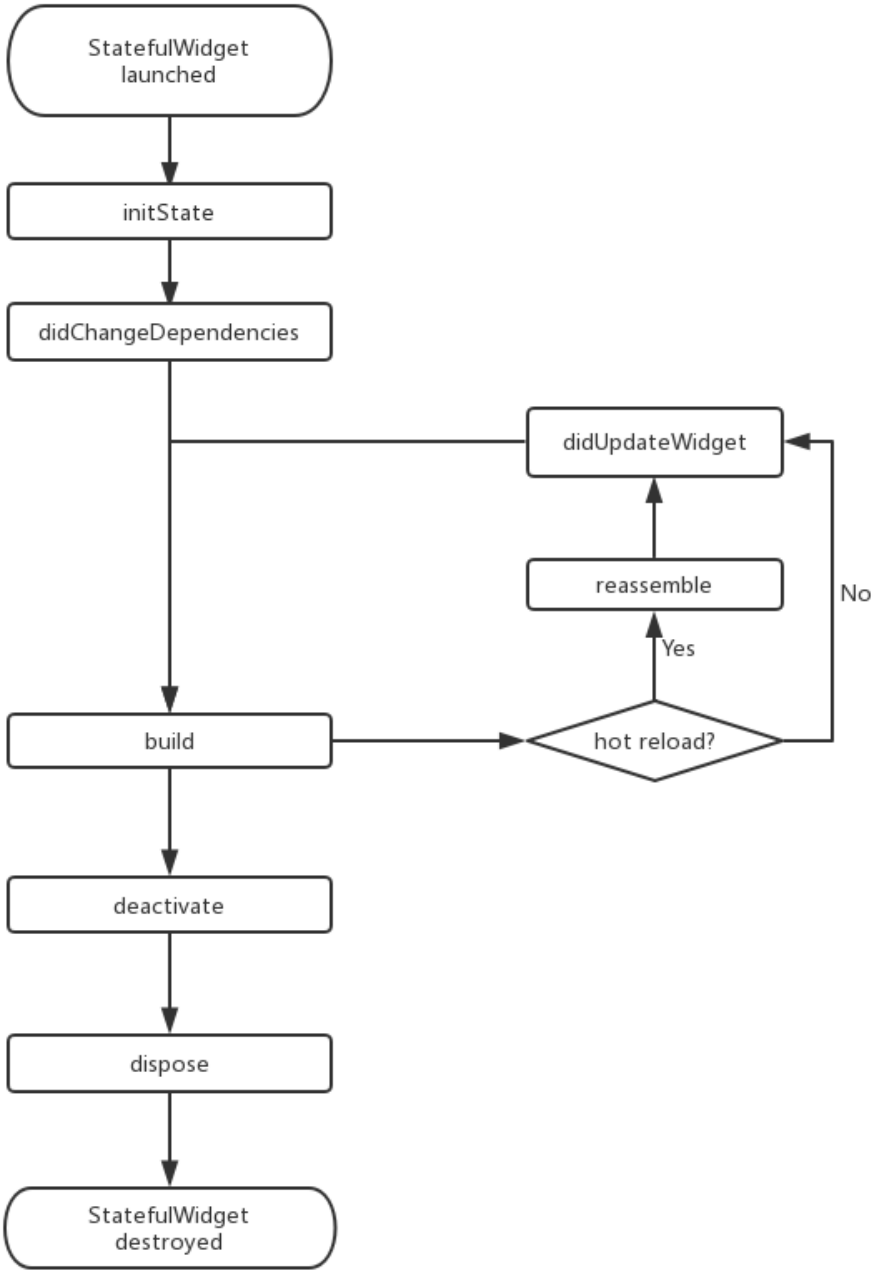
1. initState
    a. 调用时机：Widget 第一次插入widget树的时候会调用
    b. 调用次数： 一个state对象，只会调用一次
    c. 用途：常用于一次性操作，eg:初始化状态，订阅**子树**的事件通知。这里的子树应该是指从该节点开始往下的所有子view。
    d. 注意： 不能在该方法中调用 context.dependOnInheritedWidgetOfExactType,理由是InheritFromWidget可能发生变化。之后InheritFromWidget中会讲解。
2. didChangedependencies()
    a. State对象的依赖发生变化时会调用。目前我的理解是，State所依赖的一些配置，eg: 系统语言Locale或 应用主题，发生改变时，frameWork 会通知 widget调用此回调。
3. build
    a. 用途： 构建Widget 子树
    b. 调用时机： initState()/ didUpdateWidget() / setState() / didChangeDependencies() / State对象更换位置之后。
4. reassemble
    a. 调用时机与用途：调试模式下会调用该方法，调试的时候可以用它来设置一些值。release模式下不会调用该方法。
5. didUpdateWidget
    a. 调用时机：
        i. widget重新构建
        ii. widget.canUpdate返回为true
        iii. 何时为true？ 新旧widget的 key 和 runtimeType同时相同时，会返回true。
    e. 用途： widget重新构建后，用于更新UI。
6. deactive
    a. 调用时机：state对象从树中被移除的时候调用。
    b. 那什么时候会被移除呢？是widget被移除的时候吗?
7. dispose

a. 调用时机：如果移除State对象后没有重新插入，就会调用dispose()方法。
b. 用途：通常在这里释放资源。

生命周期主要流程

1. 初始化 第一次打开页面：
    a. initState()
    b. -> didChangeDependencies()
    c. -> build()
2. 点击热重载：
    a. reassemble
    b. -> didUpdateWidget
    c. -> build
3. 移除 Widget:
    a. reassemble
    b. -> deactive
    c. -> dispose

注意：标注有@mustCallSuper的父类方法，必须在子类中优先调用。



图片来源于网络，侵删

3.1.7 在widget中获取State对象

通过 Context获取

```
# 获取最近的 指定父类的 state
ScaffoldState _state = context.findAncestorStateOfType<ScaffoldState>();
# 调用获取到的state的方法
_state.showSnackBar()...
```

a. 不过，通常State是私有的， 不应该直接被调用。感觉这个提示没啥用，Java里边我们不是照样反射拿private属性调用private方法...

b. 如果widget希望state被调用，会默认提供of()方法，没错。名字就叫Widget.of(context)方法。

c. 如果我们希望自定义的State希望被外界调用，可以参照这个设计思路。

通过 GlobalKey

d. 总共分两步：

i. 给目标 StatefulWidget 添加 GlobalKey。

```
static GlobalKey<ScaffoldState> _globalKey = GlobalKey();
...
Scaffold(
    key: _globalKey, //设置key
)

```

ii. 通过GlobalKey 来获取 State对象。

```
_globalKey.currentState.stuff();
```

名词解释：GlobalKey 是在整个APP中，引用Element的机制。从名字上也能看出来。都Global了。

如果一个Widget设置了GlobalKey，我们就可以调用下面的方法：

```
globalKey.currentWidget 获取widget
globalKey.currentElement 获取 Element
globalKey.currentState  如果是StatefulWidget，可以获取 state对象。
```

但是 GlobalKey的开销很大，有备选方案，尽量避免使用它。

那到底开销是有多大呢？是什么开销呢？内存吗？

注意：GlobalKey在整个Widget树种是唯一的，不能重复。