

FLUTTER 跨平台UI框架

FLUTTER的特点

Flutter与Cordova、React Native相比，具有许多独特的实现，以下是它众多特点中最有特色的3项：

- 平台提供Surface和Canvas
- UI控件和平台无关
- 使用Dart而非JS

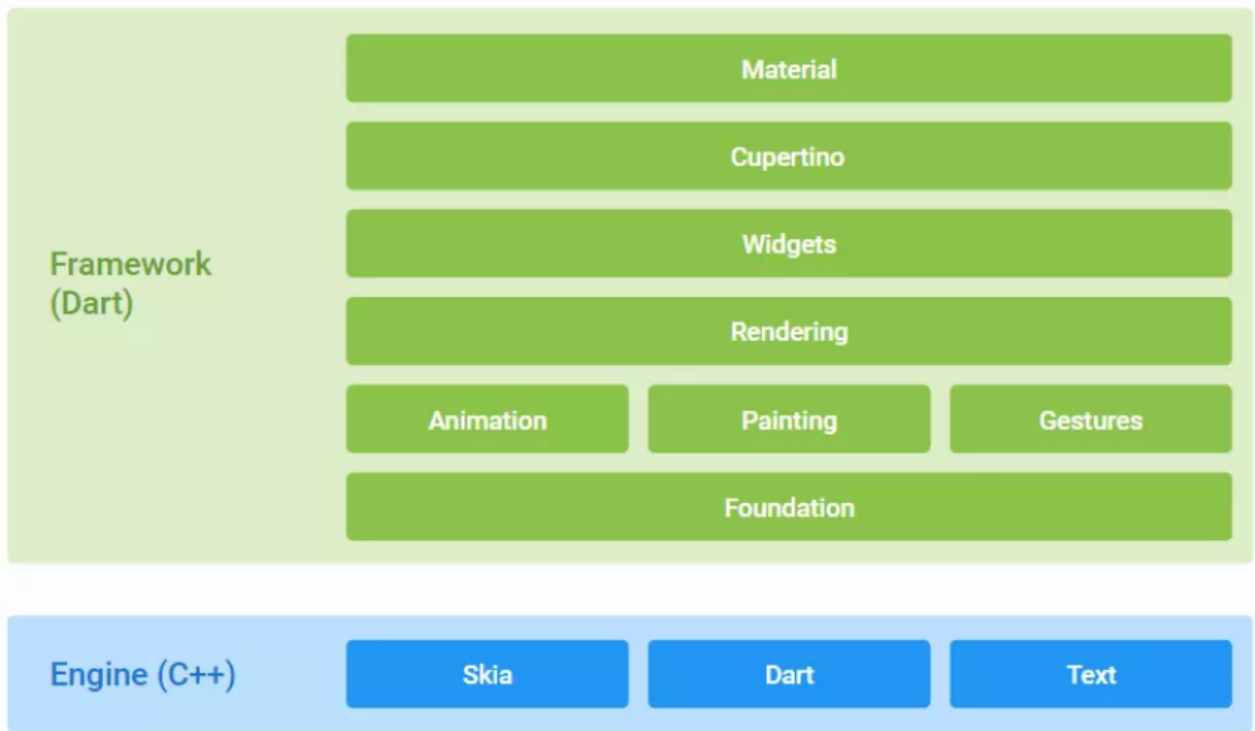
Flutter与其它大多数框架不同，因为Flutter既不使用WebView（如Cordova），也不使用操作系统的原生控件（如RN），而是使用自己的高性能渲染引擎来绘制 `Widget`（后续会进一步介绍什么是 `Widget`）。这样可以保证在多平台上UI的一致性，也可以打破对原生控件依赖而带来的限制。

Flutter所使用的Dart语言支持AOT，在AOT下速度远超JS。同时，Flutter使用自己的渲染引擎来绘制UI，布局数据等由Dart语言直接控制，所以在布局过程中不需要像RN那样要在JavaScript和Native之间通信，可以减少性能开销。

Cordova

特点：

- 本地Web代码
- 平台WebView/UIWebView
- CodovaJS加载插件



FLUTTER WIDGET

在Flutter中，`Widget` 的功能是“描述一个UI元素的配置数据”，Flutter内一切皆 `Widget`，而 `Widget` 本身是不可变的，没个 `Widget` 状态都代表了一帧。

那 `Widget` 是怎么工作的呢？

`Widget` 其实并不是表示最终绘制在设备屏幕上的显示元素，而它只是描述显示元素的一个配置数据。

实际上，Flutter中真正代表屏幕上显示元素的类是 `Element`，也就是说 `Widget` 只是描述 `Element` 的配置数据。`Widget` 只是UI元素的一个配置数据，并且一个 `Widget` 可以对应多个 `Element`，这是因为同一个 `Widget` 对象可以被添加到UI树的不同部分，而真正渲染时，UI树的每一个 `Element` 节点都会对应一个 `Widget` 对象。

以下是一个简单的 `Stateless Widget` 示例：

```
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Welcome to Flutter',
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Welcome to Flutter'),
        ),
        body: const Center(
          child: const Text('Hello World'),
        ),
      ),
    );
  }
}
```

```

    ),
  ),
);
}
}

```

`Widget` 有很多类别，接下来我们进一步展开。

StatelessWidget和StatefulWidget

StatelessWidget

在上面的示例中，我们已经看到了 `StatelessWidget`，`StatelessWidget` 相对比较简单，它继承自 `Widget` 类，重写了 `createElement()` 方法：

```

@override
StatelessElement createElement() => new StatelessElement(this);

```

`StatelessElement` 间接继承自 `Element` 类，与 `StatelessWidget` 相对应（作为其配置数据）。`StatelessWidget` 用于不需要维护状态的场景，它通常在 `build` 方法中通过嵌套其它 `Widget` 来构建UI，在构建过程中会递归的构建其嵌套的 `Widget`。

StatefulWidget

和 `StatelessWidget` 一样，`StatefulWidget` 也是继承自 `Widget` 类，并重写了 `createElement()` 方法，不同的是返回的 `Element` 对象并不相同；另外 `StatefulWidget` 类中添加了一个新的接口 `createState()`。

```

abstract class StatefulWidget extends Widget {
  const StatefulWidget({ Key key }) : super(key: key);

  @override
  StatefulElement createElement() => new StatefulElement(this);

  @protected
  State createState();
}

```

- `StatefulElement` 间接继承自 `Element` 类，与 `StatefulWidget` 相对应（作为其配置数据）。`StatefulElement` 中可能会多次调用 `createState()` 来创建状态(State)对象。
- `createState()` 用于创建和 `StatefulWidget` 相关的状态，它在 `StatefulWidget` 的生命周期中可能会被多次调用，本质上就是一个 `StatefulElement` 对应一个State实例。

容器Widget和渲染Widget

`Text`，`Slider`，`ListTile` 等都属于渲染Widget，其内部主要是 `RenderObjectElement`。

`StatelessWidget` 和 `StatefulWidget` 等属于容器 Widget，其内部使用的是 `ComponentElement`，本身是不存在 `RenderObject` 的。

获取Widget的位置和大小等，都需要通过RenderObject获取。