Flutter 详解 Key

兜兜转转

github:https://github.com/ifgyong

Key 是什么

用官方的说法就是：

`key`是用来作为`Widget`、`Element`和`SemanticsNode`的标示，仅仅用来更新`widget->key`相同的小部件的状态。
`Key`子类包含`LocalKey`和`GlobalKey`。

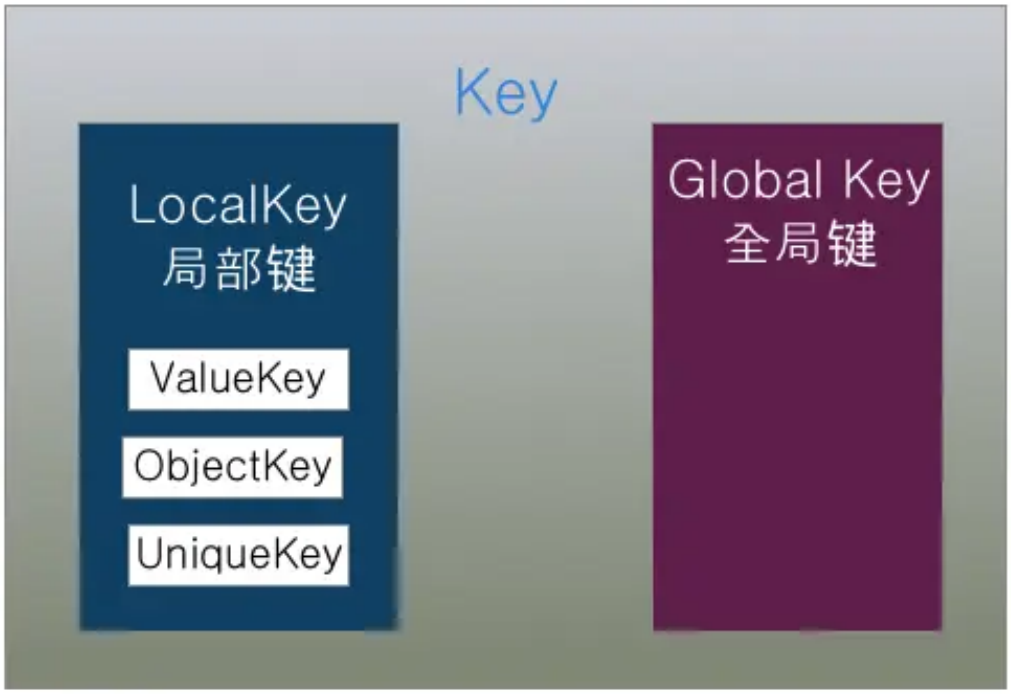
LocalKey

看下`LocalKey`的定义：

```
abstract class LocalKey extends Key {
  const LocalKey() : super.empty();
}
```

`LocalKey`定义了初始化函数，默认为值空。

`LocalKey`子类包含`ValueKey`/`ObjectKey`/`UniqueKey`，如图所示：



image

ValueKey

`ValueKey`顾名思义是比较的是值

看下关键函数

```
@override
bool operator ==(Object other) {
  if (other.runtimeType != runtimeType)
    return false;
  return other is ValueKey<T>
      && other.value == value;
}
```

那么使用起来也是很简单的,当我们想要系统根据我们所给的`key`来判断是否可以刷新时，可以使用该`key`。

```
TextField(
          key: ValueKey('value1'),
        ),
        TextField(
          key: ValueKey('value2'),
        ),
```

image

当我们来交换顺序时,`TextField`的值也交换了，也就是我们的`key`带走了值。

```
TextField(
  key: ValueKey('value2'),
),
TextField(
  key: ValueKey('value1'),
),
```

image

如果我们使用其他类来传值呢？我们把类`Student`作为`value`传值进去。

```
class Student {
  final String name;

  Student(this.name);

  @override
  int get hashCode => name.hashCode;
}

TextField(
  key: ObjectKey(Student('老王')),
),
TextField(
  key: ObjectKey(Student('老王')),
),
```

刷新之后并无报错，使用正常。

当我们在`Student`重写了操作符`==`之后再看下,我们将`Student`代码稍微改动下

```
class Student {
  final String name;

  Student(this.name);

  @override
  int get hashCode => name.hashCode;

  @override
  bool operator ==(Object other) =>
      identical(this, other) ||
      other is Student &&
          runtimeType == other.runtimeType &&
          name == other.name;
}
```

然后`hot reload`,结果报错了

```
If multiple keyed nodes exist as children of another node, they must have unique keys.
```

刚才我们所改的`Student`操作符`==`导致了，在`Key`对比`Value`的时候重载了`Student`的操作符，才导致的报错，我们需要设置不同姓名的同学，来区分不同的同学。

ObjectKey

顾名思义是比较对象的`key`,那么这个`key`是如何比较对象呢?我们看下源码;

```
  @override
  bool operator ==(Object other) {
    if (other.runtimeType != runtimeType)
      return false;
    return other is ObjectKey
        && identical(other.value, value);
  }
```

官方显示比较类型，当类型不一致，判定为不是通过一个对象，如果另外一个也是`ObjectKey`,则判断地址是否相同，只有地址相同才判定为同一个对象。

测试数据；

```
class Student {
  final String name;

  Student(this.name);

  @override
  int get hashCode => name.hashCode;

  @override
  bool operator ==(Object other) =>
      identical(this, other) ||
      other is Student &&
          runtimeType == other.runtimeType &&
          name == other.name;
}
```

```
TextField(
    key: ObjectKey(Student('老王')),
  ),
  TextField(
    key: ObjectKey(Student('老王')),
  ),
```

刷新界面之后，并无报错。

`ObjectKey`稍微修改

```
   _student = Student('老王');

  TextField(
    key: ObjectKey(_student),
  ),
  TextField(
    key: ObjectKey(_student),
  ),
```

刷新之后报错了，存在了相同的`key`。

```
If multiple keyed nodes exist as children of another node,
they must have unique keys.
```

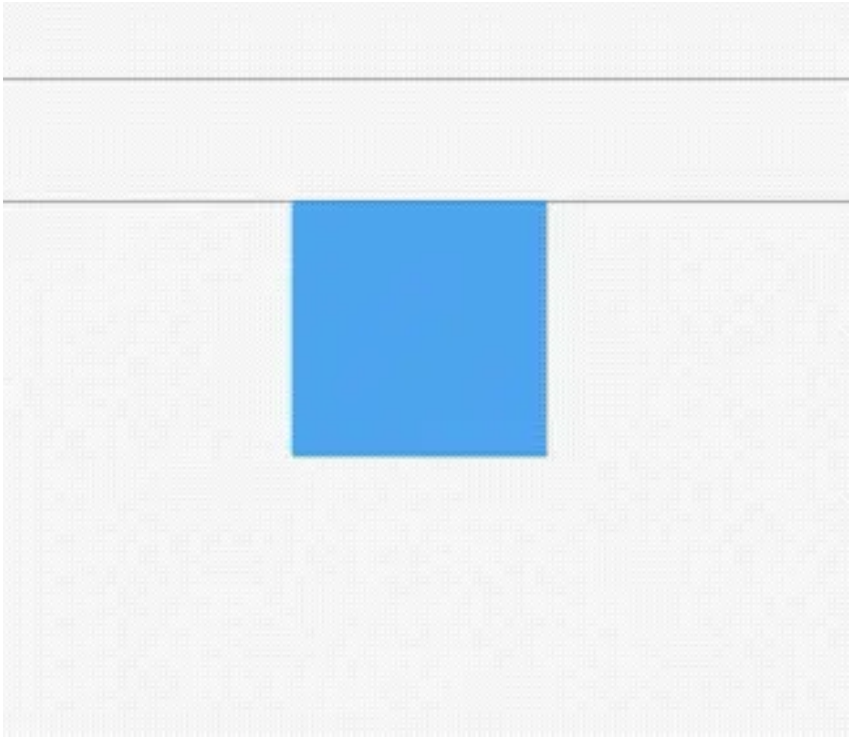
UniqueKey

每次生成不同的值，当我们每次刷新都需要一个新的值，那么正是这个存在的意义。

我们每次刷新就生成一个新的 颜色，并且渐隐渐显效果。

```
AnimatedSwitcher(
  duration: Duration(milliseconds: 1000),
  child: Container(
    key: UniqueKey(),
    height: 100,
    width: 100,
    color: Colors.primaries[count %
Colors.primaries.length],
  ),
)
```

效果：



image

GlobalKey & GlobalObjectKey

作为全局使用的`key`,当跨小部件我们通常可以使用`GlobalKey`来刷新其他小部件。

`GlobalObjectKey`和`ObjectKey`是否相等的判定条件是一致的，`GlobalObjectKey`继承`GlobalKey`,通过`GlobalKey<T extends State<StatefulWidget>>`来指定继承`state`，并实现`StatefulWidget`接口的类，然后可以通过`GlobalKey.currentState`来获取当前`state`,然后调用`state.setState((){})`完成当前小部件标记为`dirty`，在下一帧刷新当前**小部件**。

例子

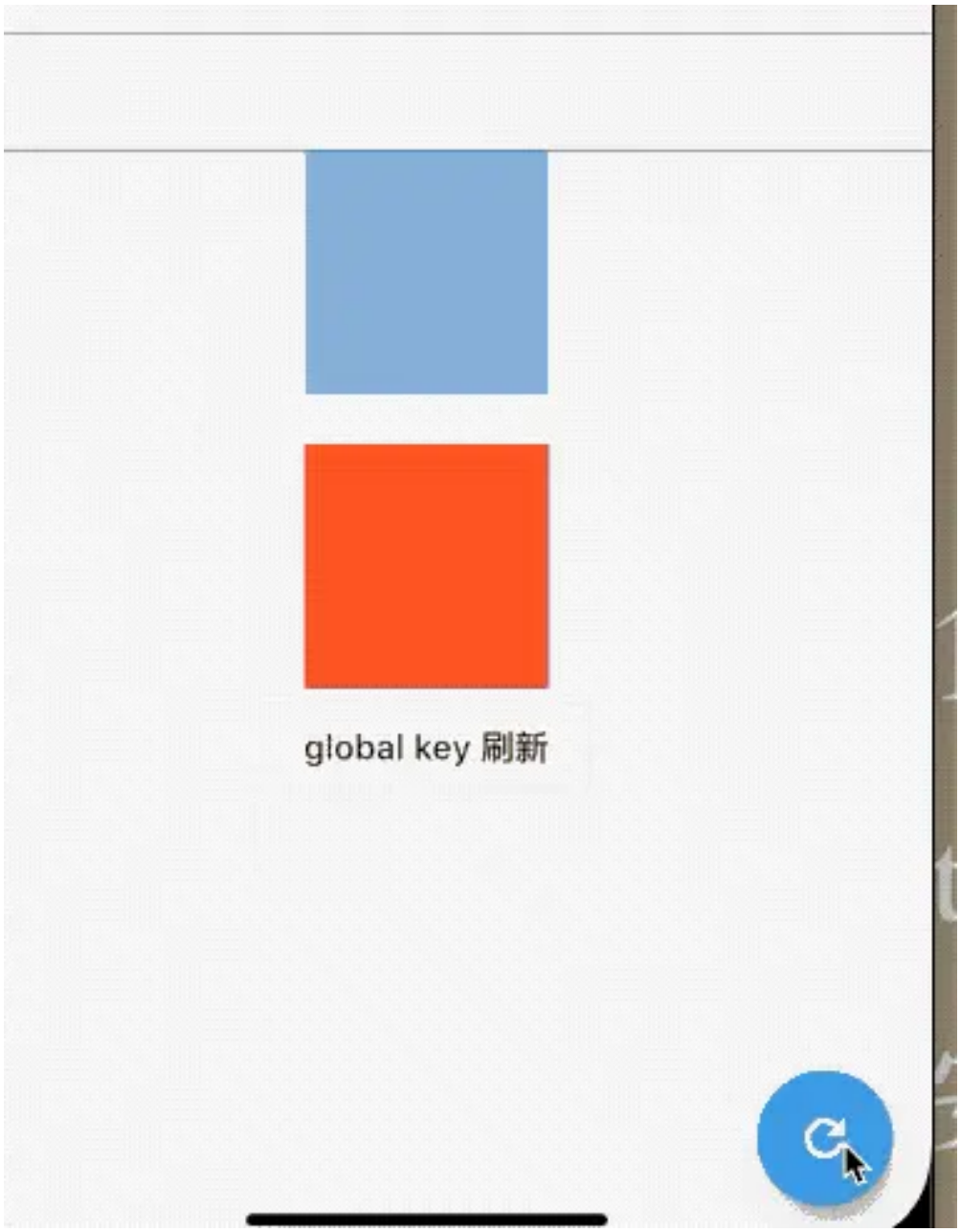
点击按钮刷新小部件的背景颜色。

```
GlobalKey _key = GlobalKey();
_Container(_key),
OutlineButton(
  child: Text('global key 刷新'),
  onPressed: () {
    _key.currentState.setState(() {});
  },
```

点击`globalKey`刷新局部小部件，点击右下角刷新整个页面。可以看到局部刷新时，只有下边的小部件改变颜色，整个页

面刷新时。

效果：



image