# Understand the key in flutter

Time：2019-12-12

> Understand the role of key in flutter (https://developpaper.com/tag/flutter/)'s rendering mechanism through the actual case, so as to use the reasonable key at a reasonable time and place

## overview

stay `Flutter` You probably know how to update the interface view: by modifying `State` De trigger `Widget` The operations to rebuild, trigger and update are `Flutter` Framework. But sometimes even if it's modified `State` , `Flutter` The frame doesn't seem to trigger either `Widget` Reconstruction

It's implicit `Flutter` The update mechanism within the framework needs to be used in combination in some cases `Key` To trigger a real "rebuild.".

The following three aspects (when, where, which) will explain how to use a reasonable key at a reasonable time and place.

## When: when should I use it `Key`

### Practical example

Requirement: click a button on the interface, and then exchange two color blocks in the line.

Statelesswidget implementation

Use `StatelessWidget` ( `StatelessColorfulTile` ) do `child` ( `tiles` ):

```
class PositionedTiles extends StatefulWidget {
  @override
  State<StatefulWidget> createState() => PositionedTilesState();
}

class PositionedTilesState extends State<PositionedTiles> {
  List<Widget> tiles;

  @override
  void initState() {
    super.initState();
    tiles = [
      StatelessColorfulTile(),
      StatelessColorfulTile(),
    ];
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
        body: SafeArea(
            child: Center(
                child: Row(
                    mainAxisAlignment: MainAxisAlignment.center,
                    children: tiles))),
        floatingActionButton: FloatingActionButton(
            child: Icon(Icons.sentiment_very_satisfied), onPressed: swapTiles));
  }
```

Update when the button is clicked `PositionedTilesState` Stored in `tiles` :

```
void swapTiles() {
    setState(() {
      tiles.insert(1, tiles.removeAt(0));
    });
  }
}
```

```
class StatelessColorfulTile extends StatelessWidget {
  final Color color = UniqueColorGenaretor.getColor();

  StatelessColorfulTile({Key key}) : super(key: key);

  @override
  Widget build(BuildContext context) => buildColorfulTile(color);
}
```

Result

Successful implementation of requirements^_^

Statefulwidget implementation

Use `StatefulWidget` ( `StatefulColorfulTile` ) do `child` ( `tiles` ):

```
class StatefulColorfulTile extends StatefulWidget {
  StatefulColorfulTile({Key key}) : super(key: key);

  @override
  State<StatefulWidget> createState() => StatefulColorfulTileState();
}

class StatefulColorfulTileState extends State<StatefulColorfulTile> {
  //Store the color in the state statefulcolorfultilestate of statefulcolorfultilestate
  Color color;

  @override
  void initState() {
    super.initState();
    color = UniqueColorGenaretor.getColor();
  }

  @override
  Widget build(BuildContext context) => buildColorfulTile(color);
}
```

Modify external container `PositionedTiles` in `tiles` :

```
  @override
  void initState() {
    super.initState();
    tiles = [
      StatefulColorfulTile(),
      StatefulColorfulTile(),
    ];
  }
```

Result

It doesn’t seem to work-_-

Why use `StatefulWidget` Can’t update it successfully? You need to understand the following first.

## Update principle of widget by fluter

In the flutter framework, the view is maintained in the tree structure. The widgets we write are nested one by one, and finally combined into a tree.

StatelessWidget

In the first use `StatelessWidget` When flutter renders these widgets, `Row` A widget provides an ordered set of slots for its child widgets. For each widget, flutter will build a corresponding `Element` 。 Build this `Element` Tree is quite simple, save only about each `Widget` Type information and pairs `Widget` Quotation. You can put this `Element` Tree is like the skeleton of your flutter app. It shows the structure of the app, but other information needs to be referenced by the original `Widget` Look for it.

 StatelessWidget Tree & Element Tree"

When we exchange two color blocks in a row, the flutter traverses `Widget` Tree to see if the skeleton structure is the same. It from `Row` The widget starts, and then moves to its child widget. The element tree checks whether the widget is the same type and `Key` 。 If they are the same, it updates the reference to the new widget. In our case, widget has no key set, so `Flutter` Just check the type. It does the same thing to the second child. So the element tree will be updated according to the widget tree.

When the update of element tree is completed, flutter will build a render object tree according to element tree, and finally start the rendering process.

StatefulWidget

When used `StatefulWidget` When implemented, the structure of the control tree is similar, but now the color information is not stored in the control itself, but in the external state object.

Now, let’s click the button to exchange the order of the controls. The fluent will traverse the element tree and check the widget tree `Row` Control and update the reference in the element tree, then the first tile control checks whether its corresponding control is of the same type, and it finds that the other is of the same type; then the second tile control does the same thing, and finally

causes flutter to think that neither of the two controls has changed. Flutter uses the state of the element tree and its corresponding control to determine the content to be displayed on the device, so the element tree does not change, and the displayed content will not change.

Statefullwidget with key

Now, for `StatefulColorfulTile` Pass a `Key` Participants:

```
void initState() {
  super.initState();
  tiles = [
    //Use uniquekey
    StatefulColorfulTile(key: UniqueKey()),
    StatefulColorfulTile(key: UniqueKey()),
  ];
}
```

Run again:

Successful swap!

Added `Key` Later structure:

When executing swap now, the statafulwidget control in the element number will compare the type as well as the type `key` Equal or not:

Only type and `key` The corresponding widget can only be found when all of them match. Therefore, after the widget tree is exchanged, the correspondence between the child control and the original control of the element tree is disrupted, so the flutter will rebuild the element tree until the controls correspond correctly.

So now that the element tree is updated correctly, the swapped color blocks will eventually be displayed.

### Usage scenarios

If you want to change the order or number of controls in the collection, `Key` It will be very useful.

## Where: where to set the key

Normally, it should be set in the top-level widget of the current widget tree.

Go back to `StatefulColorfulTile` In the example, add one for each color block `Padding` At the same time `key` Or in the same place:

```
@override
void initState() {
  super.initState();
  tiles = [
    Padding(
      padding: const EdgeInsets.all(8.0),
      child: StatefulColorfulTile(key: UniqueKey()),
    ),
    Padding(
      padding: const EdgeInsets.all(8.0),
      child: StatefulColorfulTile(key: UniqueKey()),
    ),
  ];
}
```

After clicking the button, you can see that the color of the two color blocks will change randomly, but my expectation is that the two fixed colors will exchange with each other.

### Why is there a problem

When two in the widget tree `Padding` When they are exchanged, the color blocks they wrap are also exchanged:

Then the flutter will check to update the element tree accordingly: flutter's `Elemetn to Widget` The matching algorithm will check only one level of the tree at a time:

- At the first level, `Padding` Widgets all match correctly.

- At the second level, flutter notices that the `Key` If there is no match, the tile element will be deactivated and the connection between widget and element will be deleted

- We use it here `Key` yes `UniqueKey` , it is a `LocalKey`

`LocalKey` When a widget matches an element, the flutter looks for the matching key only at a specific level in the tree. Therefore, flutter cannot find a tile widget with the key in the peer, so it will create a new element and initialize a new state. That's why the color of the color block changes randomly. Each exchange is equivalent to generating two new widgets.

- To solve this problem: will `Key` Set to upper widget `Padding` upper

When two in the widget tree `Padding` After the exchange, the flutter can `Padding` upper `Key` Change, update `Element` Two in the tree `Padding` To achieve the exchange.

```
@override
void initState() {
  super.initState();
  tiles = [
    Padding(
      key: UniqueKey(),
      padding: const EdgeInsets.all(8.0),
      child: StatefulColorfulTile(),
    ),
    Padding(
      key: UniqueKey(),
      padding: const EdgeInsets.all(8.0),
      child: StatefulColorfulTile(),
    ),
  ];
}
```

## Which type of key should be used

`Key` The purpose is to specify a unique identity for each widget and what to use `Key` It depends on the specific use scenario.

- ValueKey

For example, in a `ToDo` List application, each `Todo` The text of an item is constant and unique. In this case, it is suitable for use `ValueKey` , value is text.

- ObjectKey

Suppose that each subwidget stores a more complex combination of data, such as an address book application for user information. Any single field, such as name or birthday, may be the same as another entry, but each data combination is unique. under these circumstances, `ObjectKey` The most suitable.

- UniqueKey

If there are multiple widgets with the same value in the collection, or if you want to make sure that each widget is different from other widgets, you can use the `UniqueKey` 。 We used it in our example `UniqueKey` Because we didn't store any other constant data on our color block, and we didn't know what color was until we built the widget.

Not in `Key` The random number is used in. If you set it like that, a new random number will be generated every time you build a widget, and the element tree will not be updated consistently with the widget tree.

- GlobalKeys

Global keys has two uses.

- They allow widgets to change their parents anywhere in the application without losing state, or they can be used to access information about another widget in a completely different part of the widget tree.
    - For example, to display the same widget on two different screens while maintaining the same state, you need to use globalkeys.

- In the second case, you may want to verify the password, but you do not want to share the status information with other widgets in the tree, and you can use the `GlobalKey<FromState>` Hold a form `Form` Of `State` 。 The example building a form with validation is available on fluent.dev.

In fact, globalkeys looks like a global variable. There are other better ways to achieve the role of globalkeys, such as inheritedwidget, Redux or block pattern.

### summary

How to use it properly `Key` :

- When: when you want to keep the state of the widget tree, use the `Key` 。 For example, when modifying a widget collection of the same type, such as in a list
- Where: will `Key` Set at the top of the widget tree to indicate unique identity
- Which: select different types of `Key`

---

> Reference resources
>
> - https://flutter.dev/docs/deve…
> - https://api.flutter.dev/flutt…
> - https://www.youtube.com/watch…
> - https://www.yuque.com/xytech/…
>
> Example code mentioned above: https://github.com/stefanji/f

Tags: flutter (https://developpaper.com/tag/flutter/)

---

### Recommended Today

## IDE usage (https://developpaper.com/ide-usage/)

Convenience method ctrl+D: Copy the current line to the next line Enter 100.for and press Enter to generate a for loop statement for(int i=0;i<100;i++){}or directly fori In the for-each loop, arrays.for and then enter to complete alt+insert, click OK in the pop-up window to generate a

Use simple code to describe Angular parent components…
Kylin operating system (kylinos) from entry to proficient –…
LeetCode | Interview Question 59 – II. Maximum Queue …
Mybatis Technology Insider–Mybatis Log Interceptor and…
Deploy react-app to Github Pages using Github Actions (…
Teach you how to encapsulate a flexible and highly reus…
Summary of common methods of JS (https://developpap…
Express logistics query interface API (https://developpap…
Configure the development environment through Larago…
Huan 笱 吭 ラ dies (https://developpaper.com/huan-%e7…

Pre: Talk about state management & concept of design (https://developpaper.com/talk-about-state-

Next: Brief tutorial of nodejs (2) (https://developpaper.com/brief-tutorial-of-nodejs-2/)

java (https://developpaper.com/question/tag/java/)
php (https://developpaper.com/question/tag/php/)
python (https://developpaper.com/question/tag/python/)
linux (https://developpaper.com/question/tag/linux/)
windows (https://developpaper.com/question/tag/windows/)
android (https://developpaper.com/question/tag/android/)
ios (https://developpaper.com/question/tag/ios/)
mysql (https://developpaper.com/question/tag/mysql/)
html (https://developpaper.com/question/tag/html/)
.net (https://developpaper.com/question/tag/net/)
github (https://developpaper.com/question/tag/github/)
node.js (https://developpaper.com/question/tag/node-js/)

Search