The ultimate solution to prevent widget rebuild by flutter

Time：2021-3-2

background

As we all know, flutter is a framework developed based on the idea of the front-end framework react. There are many similarities, but there are also some differences that I can’t see. What I feel most deeply at present is the ubiquitous rebuild of flutter. Is there any way to prevent rebuild?

Add const before widget

This method can be used once and for all, but once you add const, your widget will never be updated. Unless you are writing static pages, you’d better not use it

Write your components as “leaf” components

reference resourcesFlutter documentation (https://developpaper.com/go.php?go=aHR0cHM6Ly9hcGkuZmx1dHRlci5kZXYvZmx1dHRlci93aWRnZXRzL1N0YXRlZnVsV2lkZ2V0LWNsYXNzLmh0bWwjcGVyZm9ybWFuY2UtY29uc2lkZXJhdGlvbnM=)
That is to define your components as leaves, the bottom layer of the tree, and then you change the state (https://developpaper.com/tag/state/) inside the leaf components, so that the leaves do not affect each other, In my opinion, this is the opposite of react’s idea of state promotion, because in order not to affect each other, you can’t put the state on the root node, put it on the root node, and rebuild it as soon as you call setstate. I always used this idea to solve the problem of rebuild at the beginning,
For example, using `StreamBuilder` This can wrap your components, and then use the stream to trigger the internal rebuild of streambuilder, and isolate the external components through streambuilder. There is a small disadvantage in this way. I need to write an additional stream and close the stream, which is very wordy.

Use other libraries, such as provider

The implementation methods of these libraries are similar to those of streambuilder. They isolate other widgets through one widget and limit the update to the internal. However, they all have one thing in common. You need to work with additional external variables to trigger internal updates

The ultimate solution

Everyone who has used react knows that the class component of react has a very important life cycle called `shouldComponentUpdate` We can rewrite the declaration cycle inside the component to optimize performance.

How to optimize is to compare the values of the properties of the old and new props of the component. If they are consistent, there is no need to update the component
Does flutter have a similar life cycle? No,

The flutter team thinks that the rendering speed of flutter is fast enough, and flutter actually has a diff algorithm similar to react to compare whether the element needs to be updated. They have done optimization and caching, because updating the element of flutter is a very expensive operation, while the rebuild widget is just a new one This paper introduces an instance of a widget, just like executing a dart code (https://developpaper.com/tag/code/), which does not involve any changes to the UI layer. In addition, they also diff the old and new widgets to reduce the changes to the element layer. In any case, as long as the element is not destroyed or rebuilt, the performance will not be affected.

But through Google and Baidu, you can still find someone searching how to prevent rebuild, which shows that there is still demand in the market. Personally, I don’t think it’s excessive optimization. In fact, there are scenarios that need to be optimized. For example, the state management library provider recommended by Google provides a way to reduce unnecessary rebuilds

I don’t want to talk too much

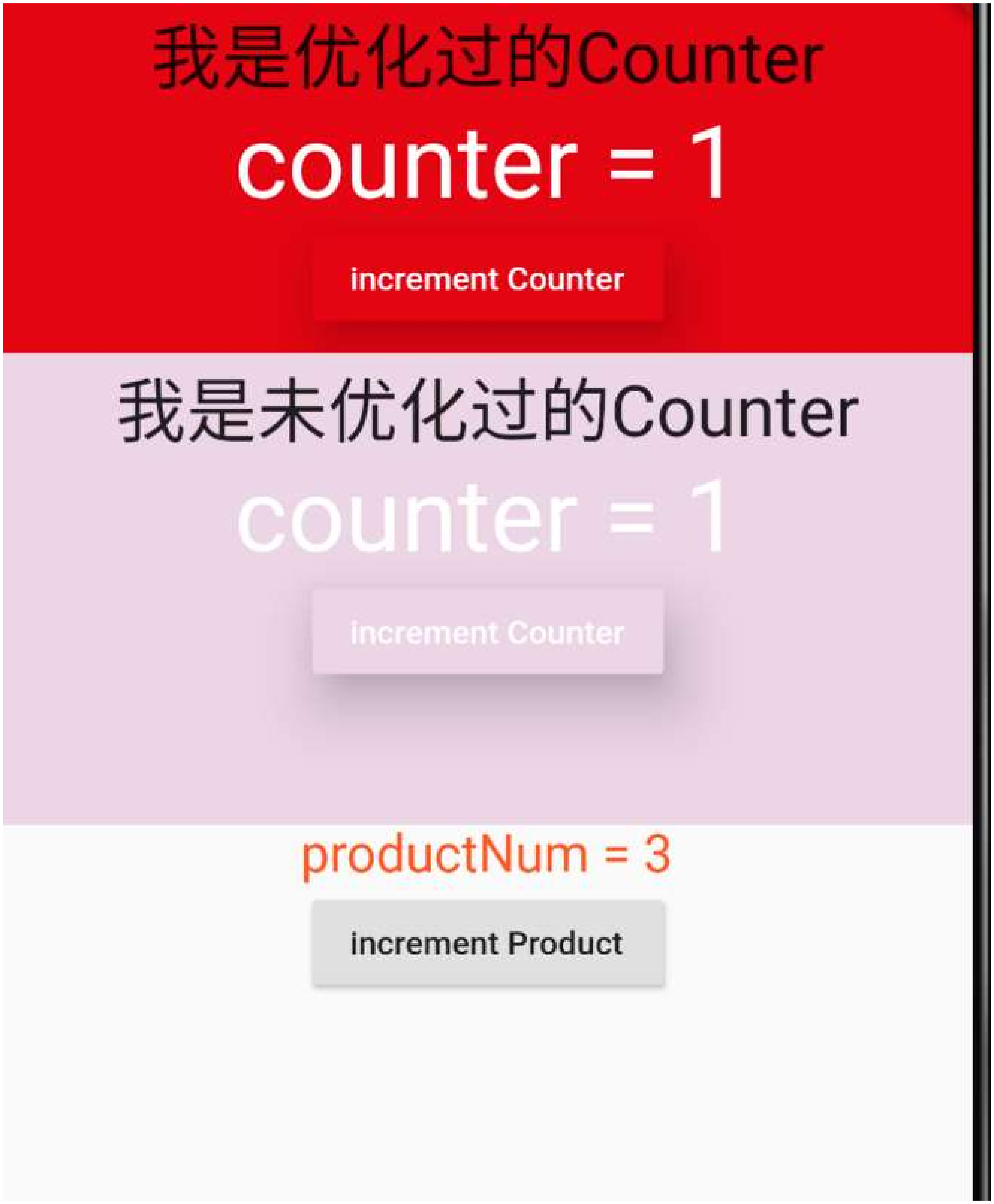
```
library should_rebuild_widget;

import 'package:flutter/material.dart';

typedef ShouldRebuildFunction<T> = bool Function(T oldWidget, T newWidget);

class ShouldRebuild<T extends Widget> extends StatefulWidget {
  final T child;
  final ShouldRebuildFunction<T> shouldRebuild;
  ShouldRebuild({@required this.child, this.shouldRebuild}):assert((){
    if(child == null){
      throw FlutterError.fromParts(
          <DiagnosticsNode>[
            ErrorSummary('ShouldRebuild widget: builder must be not  null')]
      );
    }
    return true;
  }());
  @override
  _ShouldRebuildState createState() => _ShouldRebuildState<T>();
}

class _ShouldRebuildState<T extends Widget> extends State<ShouldRebuild> {
  @override
  ShouldRebuild<T> get widget => super.widget;
  T oldWidget;
  @override
  Widget build(BuildContext context) {
    final T newWidget = widget.child;
    if (this.oldWidget == null || (widget.shouldRebuild == null ? true : widget.shouldReb
uild(oldWidget, newWidget))) {
      this.oldWidget = newWidget;
    }
    return oldWidget;
  }
}
```

These are just a few lines of code, less than 40 lines of code

Look at the test code:

```
import 'dart:math';

import 'package:flutter/material.dart';
import 'package:should_rebuild_widget/should_rebuild_widget.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: Test(),
    );
  }
}

class Test extends StatefulWidget {
  @override
  _TestState createState() => _TestState();
}

class _TestState extends State<Test> {
  int productNum = 0;
  int counter = 0;

  _incrementCounter(){
    setState(() {
      ++counter;
    });
  }
  _incrementProduct(){
    setState(() {
      ++productNum;
    });
  }
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: SafeArea(
        child: Container(
          constraints: BoxConstraints.expand(),
          child: Column(
            children: <Widget>[
              ShouldRebuild<Counter>(
                shouldRebuild: (oldWidget, newWidget) => oldWidget.counter != newWidget.c
ounter,
                child: Counter(counter: counter,onClick: _ Incrementcounter, Title: 'I am
an optimized counter'),
              ),
              Counter(
                counter: counter,onClick: _ Incrementcounter, Title: 'I am an unoptimized
counter',
              ),
              Text('productNum = $productNum',style: TextStyle(fontSize: 22,color: Colors
.deepOrange),),
              RaisedButton(
                onPressed: _incrementProduct,
                child: Text('increment Product'),
              )
            ],
          ),
        ),
      ),
    );
  }
}



class Counter extends StatelessWidget {
  final VoidCallback onClick;
  final int counter;
  final String title;
  Counter({this.counter,this.onClick,this.title});
  @override
  Widget build(BuildContext context) {
    Color color = Color.fromRGBO(Random().nextInt(256), Random().nextInt(256), Random().n
extInt(256), 1);
    return AnimatedContainer(
      duration: Duration(milliseconds: 500),
      color:color,
      height: 150,
      child:Column(
        children: <Widget>[
          Text(title,style: TextStyle(fontSize: 30),),
          Row(
            mainAxisAlignment: MainAxisAlignment.center,
            children: <Widget>[
              Text('counter = ${this.counter}',style: TextStyle(fontSize: 43,color: Color
s.white),),
            ],
          ),
          RaisedButton(
            color: color,
            textColor: Colors.white,
            elevation: 20,
            onPressed: onClick,
            child: Text('increment Counter'),
          ),
        ],
      ),
    );
  }
}
```

Layout rendering:



- We define a counter component. Counter will change its background color (https://developpaper.com/tag/background-color/) in the process of building. Every time we execute build, it will generate a random background color, so that we can observe whether the component is built or not. In addition, counter receives the value counter from the parent component and displays it. It also receives a title to distinguish different counter names

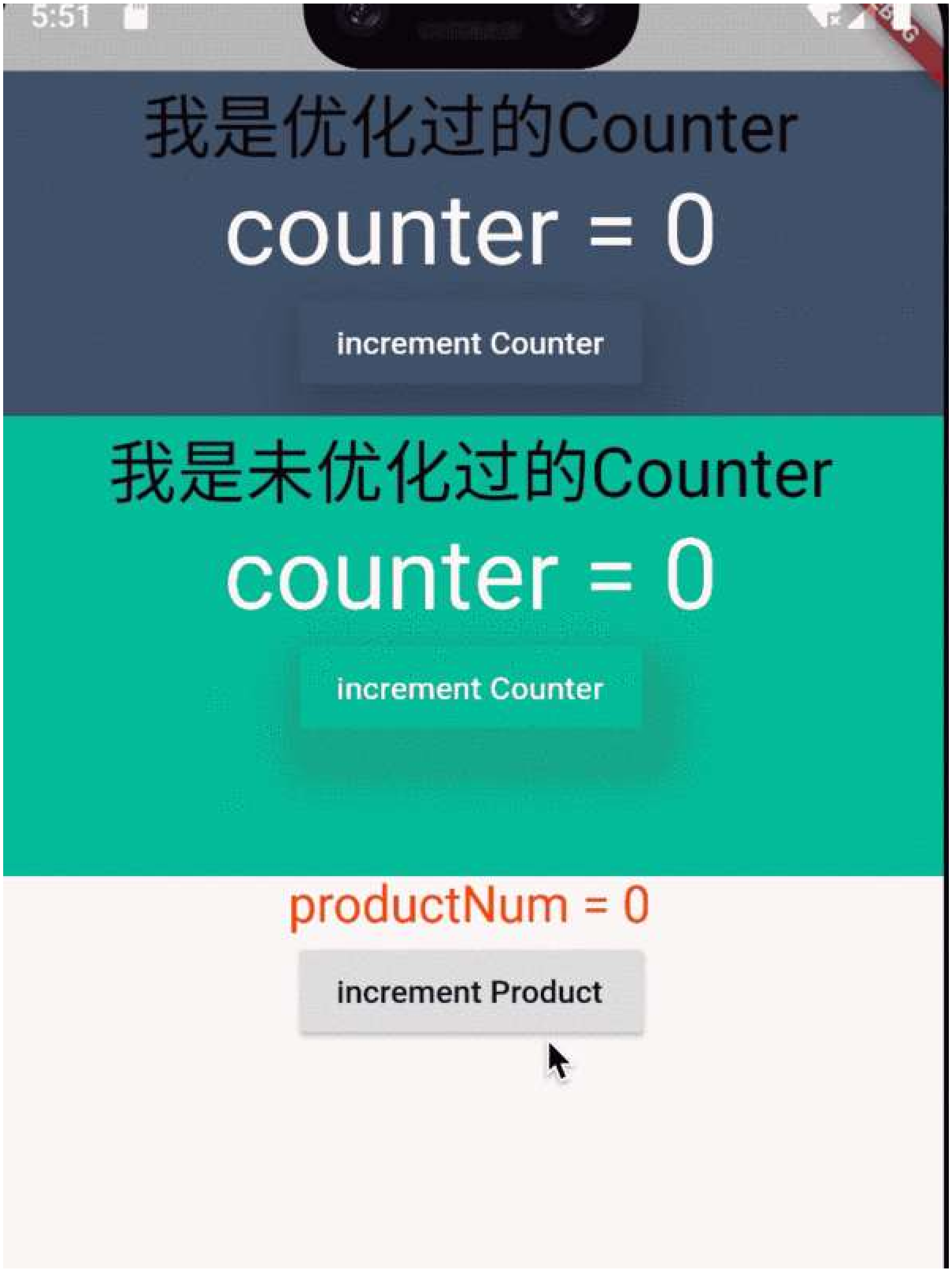
- Look at the code here

```
Column(
            children: <Widget>[
              ShouldRebuild<Counter>(
                shouldRebuild: (oldWidget, newWidget) => oldWidget.counter != newWidget.c
ounter,
                child:  Counter(counter: counter,onClick: _ Incrementcounter, Title: 'I a
m an optimized counter'),
              ),
              Counter(
                counter: counter,onClick: _ Incrementcounter, Title: 'I am an unoptimized
counter',
              ),
              Text('productNum = $productNum',style: TextStyle(fontSize: 22,color: Colors
.deepOrange),),
              RaisedButton(
                onPressed: _incrementProduct,
                child: Text('increment Product'),
              )
            ],
          )
```

The counter above is wrapped by shouldrebuild. At the same time, the shouldrebuild parameter passes in a user-defined condition. When the counter received by the counter is inconsistent, it will be rebuilt. If the new counter is consistent with the old counter, it will not be rebuilt,
The counter below is not optimized.

- We click the button to add product `increment Product` , will trigger the addition of productnum, but counter is not added at this time, so the counter wrapped by shouldrebuild is not rebuilt, while the counter wrapped below is rebuilt

Let’s take a look at GIF



Revealing the principle

In fact, the principle is consistent with the widget declared with const. Let’s take a look at the source code of flutter

```
Element updateChild(Element child, Widget newWidget, dynamic newSlot) {
...
      if (child.widget == newWidget) {
        if (child.slot != newSlot)
          updateSlotForChild(child, newSlot);
        return child;
      }
      if (Widget.canUpdate(child.widget, newWidget)) {
        if (child.slot != newSlot)
          updateSlotForChild(child, newSlot);
        child.update(newWidget);
        assert(child.widget == newWidget);
        assert(() {
          child.owner._debugElementWasRebuilt(child);
          return true;
        }());
        return child;
      }

...
}
```

Excerpt a part of it,
first

```
if (child.widget == newWidget) {
        if (child.slot != newSlot)
          updateSlotForChild(child, newSlot);
        return child;
   }
```

Here's the key, flutter found child.widget In other words, the old widget and the new widget are the same. If the reference is consistent, the child will be returned directly

If there's a discrepancy, we'll go here

```
if (Widget.canUpdate(child.widget, newWidget)) {
        if (child.slot != newSlot)
          updateSlotForChild(child, newSlot);
        child.update(newWidget);
        assert(child.widget == newWidget);
        assert(() {
          child.owner._debugElementWasRebuilt(child);
          return true;
        }());
        return child;
      }
```

If it can be updated here, it will go child.update Once this method is gone, the build method will be executed.
See what it does

```
@override
  void update(StatelessWidget newWidget) {
    super.update(newWidget);
    assert(widget == newWidget);
    _dirty = true;
    rebuild();
  }
```

When you see rebuild (), you know that you must execute build.

Actually see if （ child.widget ==We also know why const text () makes text not repeat build, because constants never change

github： shouldRebuild (https://developpaper.com/go.php?go=aHR0cHM6Ly9naXRodWIuY29tL2ZhbnRhc3k1MjUvc2hvdWxkX3JlYnVpbGQ=)

If you think it has helped you, please star

Tags: assembly (https://developpaper.com/tag/assembly/), background color (https://developpaper.com/tag/background-color/), code (https://developpaper.com/tag/code/), Leaf (https://developpaper.com/tag/leaf/), state (https://developpaper.com/tag/state/)

Recommended Today

Reverse | cs1.6 perspective through inlinehook opengl (ht…

Reverse | cs1.6 perspective through inlinehook opengl I've been wanting to get it done before, but I got it today.The principle of inlinehook is basically the same as the previous article.https://www.cnblogs.com/Mz1-rc/p/16586411.html cs1.6 can be opengl to d3d, first adjust

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

Pre: Understanding of BTREE index and hash index (https://developpaper.com/understanding-of-btree-index-

Next: Transforming layui table component to realize multiple sorting (https://developpaper.com/transforming-

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