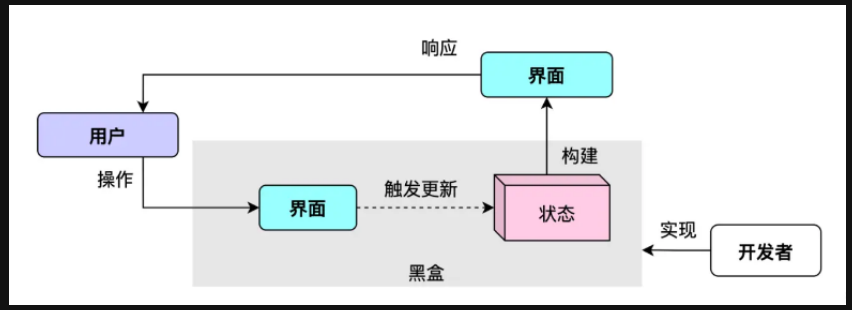
GetX 状态管理技术分享

什么是状态

用户通过操作界面，可以进行正确的逻辑处理，并得到一定的响应反馈。



开发者需要关心状态的变化。

为什么需要状态管理

Android

```
/// 展示的数量
private int mCount = 0;
/// 中间展示数字的 TextView
private TextView mTvContent;
/// 右下角按钮调用的方案
private void increase() {
  mCount++;
  mTvContent.setText(mCount);
}
```

「命令式」

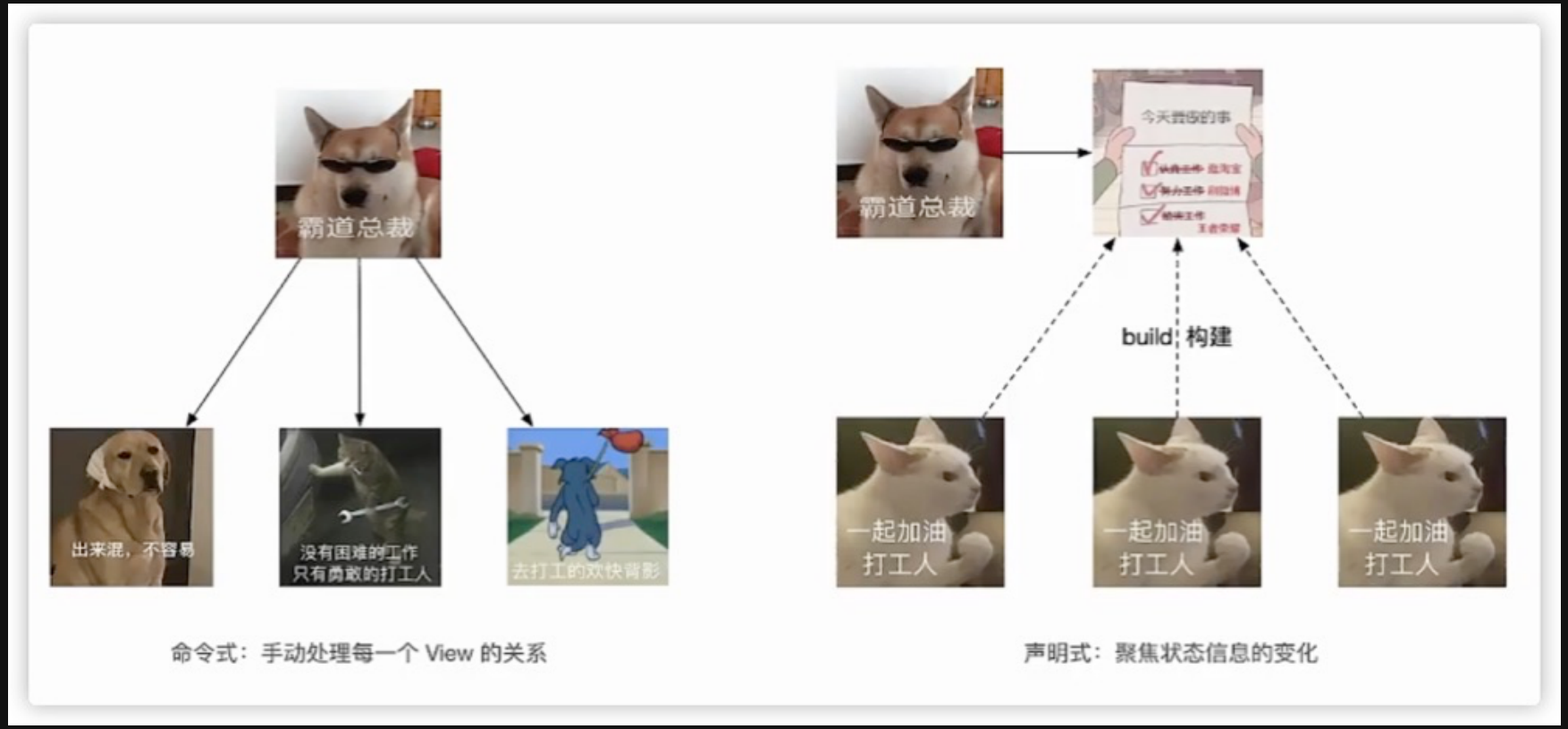
Flutter

```
int _counter = 0;
increase() {
  _counter++;
  setState((){});
}
```

UI = f(State)

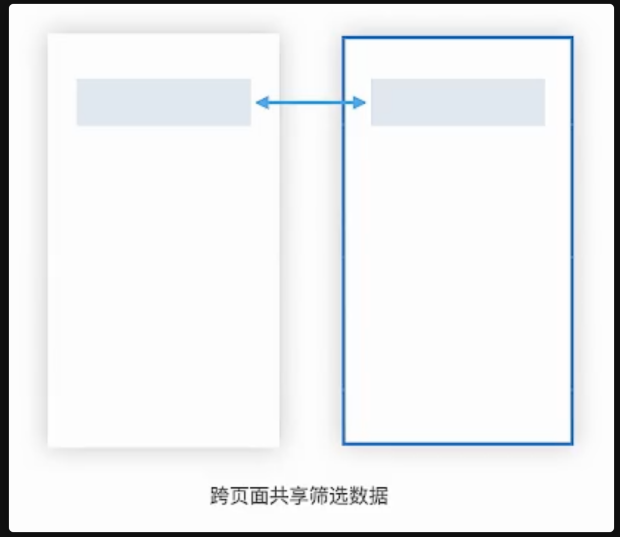
「声明式」

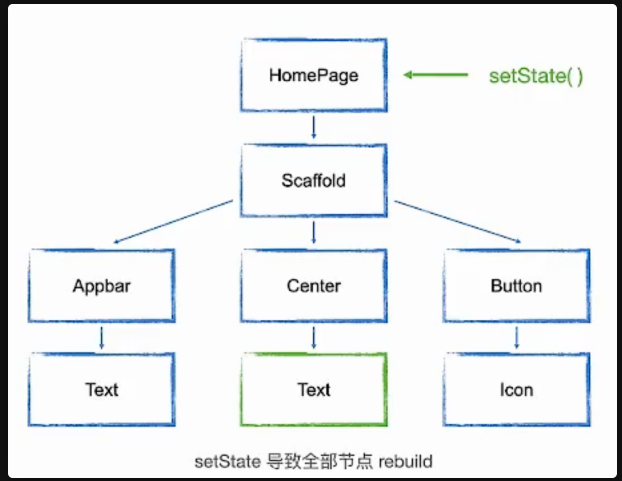
声明式开发的优点


霸道总裁
出来混，不容易
没有困难的工作
只有勇敢的打工人
去打工的欢快背影
命令式：手动处理每一个 View 的关系
霸道总裁
今天要做的事
build 构建
一起加油
打工人
一起加油
打工人
一起加油
打工人
声明式：聚焦状态信息的变化

声明式开发的问题：

1. 逻辑和页面 UI 耦合，导致无法复用/单元测试，修改混乱等
2. 难以跨组件（跨页面）访问数据
3. 无法轻松的控制刷新范围（页面 setState 的变化会导致全局页面的变化）



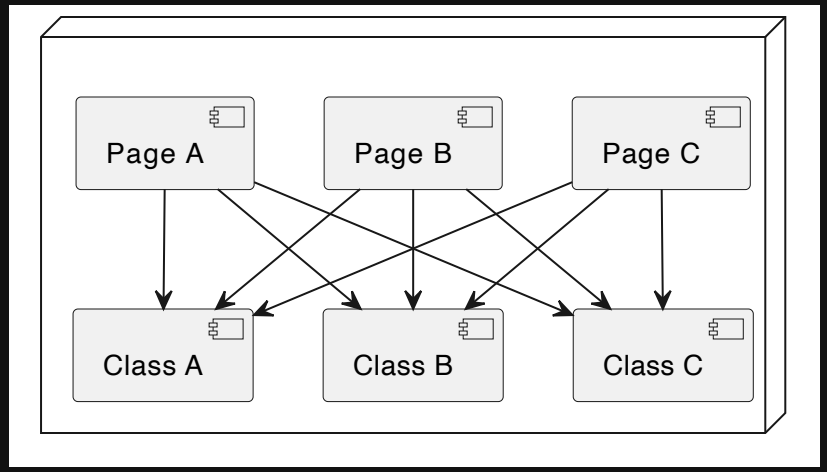


```
class _MyHomePageState extends State<MyHomePage> {
  int _counter = 0;
  void _incrementCounter() {
    setState(() {
      _counter++;
    });
  }
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: Column(
          children: <Widget>[
            const Text(
              'You have pushed the button this many times:',
            ),
            Text('$_counter',),
          ],
        ),
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: _incrementCounter,
        tooltip: 'Increment',
        child: const Icon(Icons.add),
      ), // This trailing comma makes auto-formatting nicer for build methods.
    );
  }
}
```

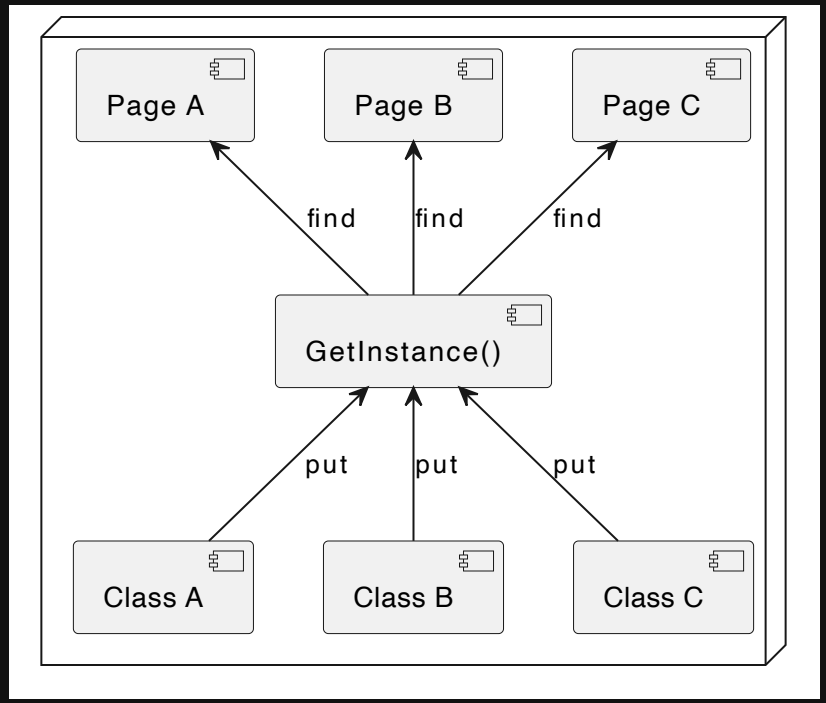
GetX 如何解决这些问题?

1. 依赖注入

GetX 依赖注入前



GetX 依赖注入后



控制器

```
class Controller extends GetxController {
  var count = 0;
  void increment() {
    count++;
    update();
  }
}
```

页面2

```
class Second extends StatelessWidget {
  final Controller ctrl = Get.find();
  @override
  Widget build(context){
     return Scaffold(body: Center(child: Text("${ctrl.coun
  }
}
```

```
class Home extends StatelessWidget {
  final controller = Get.put(Controller());
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: Column(
          children: [
            GetBuilder<Controller>(
                builder: (_) => Text(
                      'clicks: ${controller.count}',
                    )),
            ElevatedButton(
              child: Text('Next Route'),
              onPressed: () {
                Get.to(Second());
              },
            ),
          ],
        ),
      ),
      floatingActionButton: FloatingActionButton(
          child: Icon(Icons.add),
          onPressed: controller.increment(),
          ),
    );
  }
}
```

Get.put 源码解析

```
// Get.put(Controller())
S put<S>(S dependency,
    {String? tag,
    bool permanent = false,
    InstanceBuilderCallback<S>? builder}) =>
GetInstance().put<S>(dependency, tag: tag, permanent: perm

// GetInstance
S put<S>(
    S dependency, {
    String? tag,
    bool permanent = false,
    @deprecated InstanceBuilderCallback<S>? builder,
  }) {
    _insert(
        isSingleton: true,
        name: tag,
        permanent: permanent,
        builder: builder ?? (() => dependency));
    return find<S>(tag: tag);
  }


//
```

Get.find 源码解析

```
// Get.find()
S find<S>({String? tag}) => GetInstance().find<S>(tag: tag

// GetInstance
S find<S>({String? tag}) {
  final key = _getKey(S, tag);
  if (isRegistered<S>(tag: tag)) {
    final _InstanceBuilderFactory dep = _singl[key];
                                          tag);
_InstanceBuilderFactory(
  this.isSingleton,
  this.builderFunc,
  this.permanent,
  this.isInit,
  this.fenix,
  this.tag, {
  this.lateRemove,
});
      dependency = builderFunc();
    }
    return dependency!;
  } else {
    return builderFunc();
  }
}
```

2. GetX Widget

GetBuilder

```
// controller
class GetBuilderCtrl extends GetxControlle
  int counter = 0;

  increment() {
    counter++;
    update();
  }
}
// widget
class GetBuilderView extends StatelessWidg
  const GetBuilderView({super.key});

  @override
  Widget build(BuildContext context) {
    final c = Get.put(GetBuilderCtrl());
    return GetBuilder<GetBuilderCtrl>(
        builder: (_) => Text(
              'GetBuilder Counter: ${c.cou
            ));
  }
}
```

GetX

```
// controller
class GetXCtrl extends GetxController {
  RxInt counter = 0.obs;

  increment() {
    counter++;
  }
}

// widget
class GetXView extends StatelessWidget {
  const GetXView({super.key});

  @override
  Widget build(BuildContext context) {
    final c = Get.put(GetXCtrl());
    return GetX<GetXCtrl>(
        builder: (_) => Text(
              'GetXView Counter: ${c.count
            ));
  }
}
```

Obx

```
// controller
class ObxCtrl extends GetxController {
  RxInt counter = 0.obs;

  increment() {
    counter++;
  }
}

// widget
class ObxView extends StatelessWidget {
  const ObxView({super.key});

  @override
  Widget build(BuildContext context) {
    final c = Get.put(ObxCtrl());
    return Obx(() => Text(
          'ObxView Counter: ${c.counter}',
        ));
  }
}
```

三个 widget 的区别

GetBuilder

1. 我想手动决定什么时候重新构建小部件
2. 我有几个状态变量，可以作为一个组进行刷新
3. 需要使用 initState 生命周期函数
4. 没有额外性能开销
5. 使用较为复杂

GetX

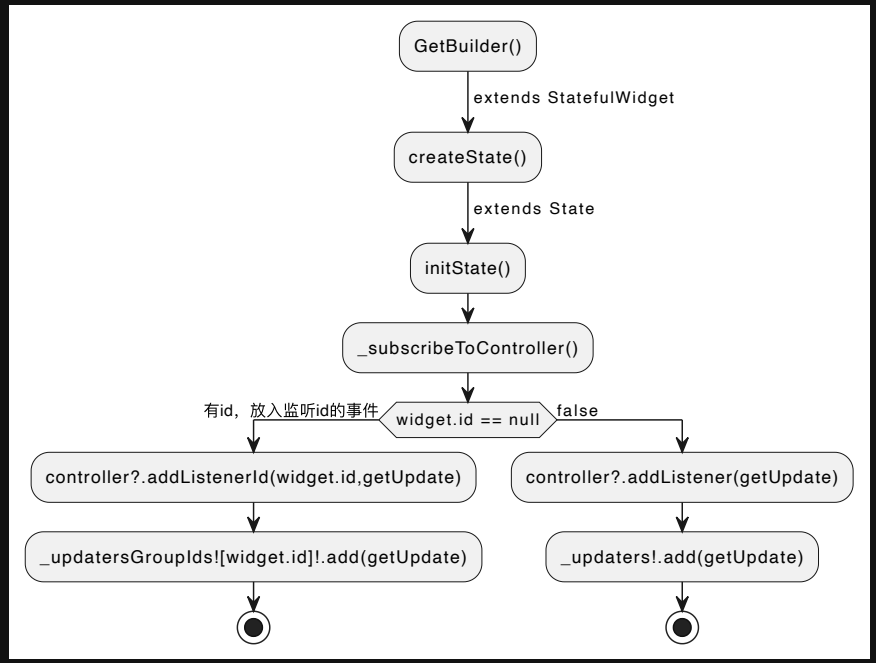
1. 在 controller 还未注册时使用
2. 明确当前 widget 内需要使用什么 controll
3. 需要使用 initState 生命周期函数
4. 需要响应式更新 widget
5. 比较耗费性能

Obx

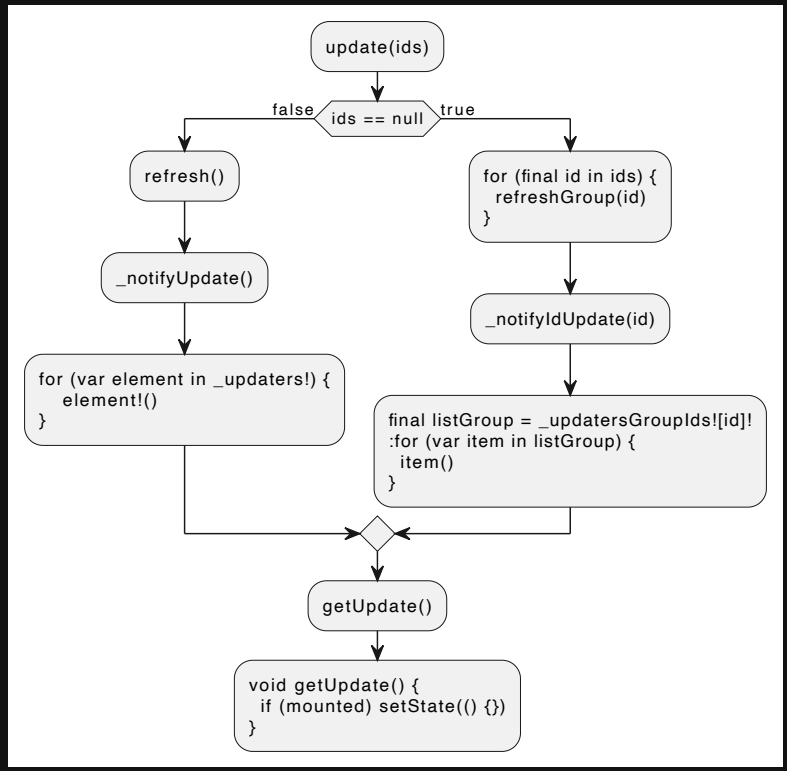
1. 当 controller 已经在其他地方注册
2. 使用上最简单
3. 需要响应式更新 widget
4. 比较耗费性能

GetBuilder 源码解析

初始化

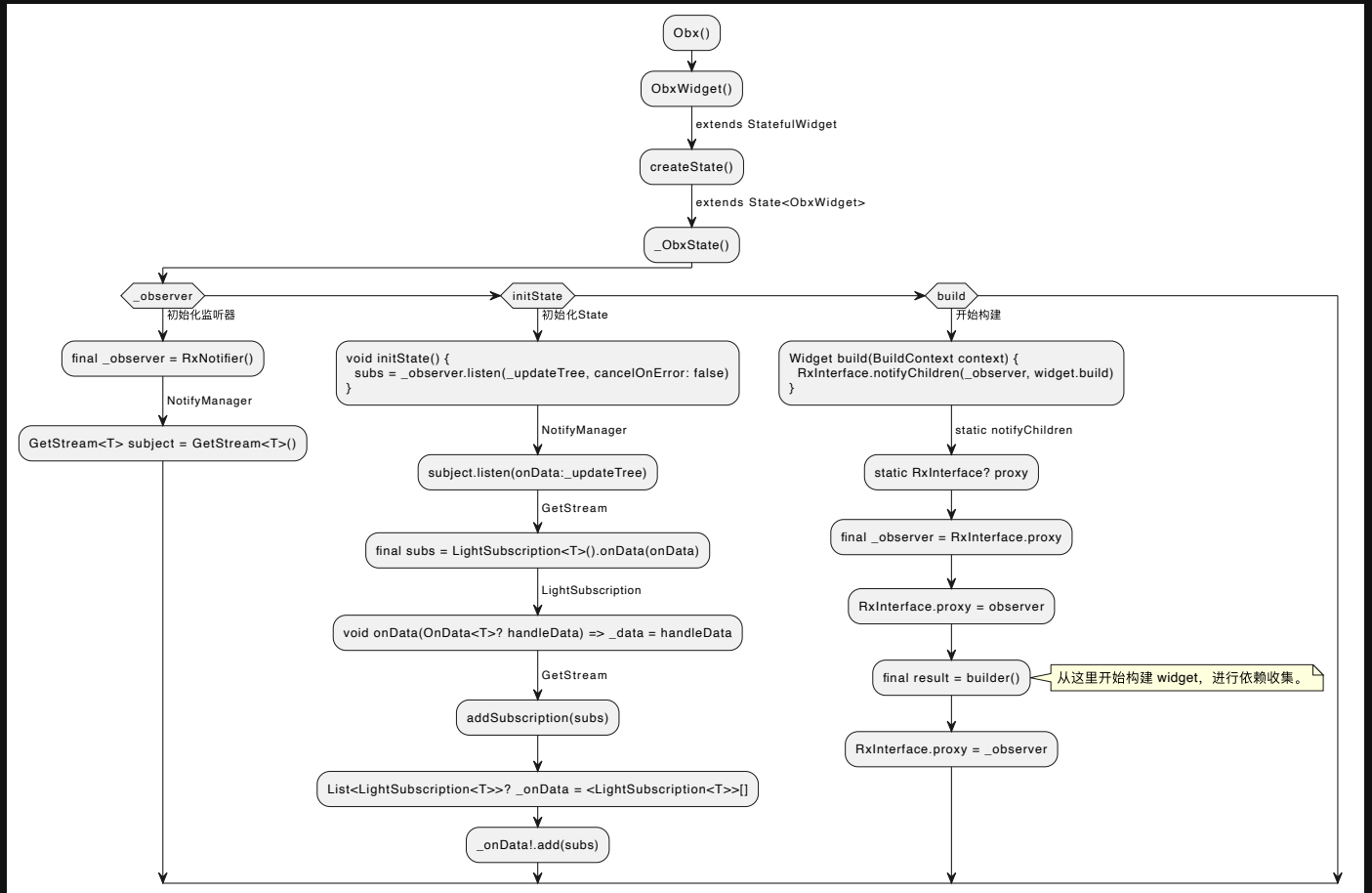


触发更新



Obx 源码解析

Obx widget初始化



Obx 源码解析

依赖收集

```
Text('${c.counter}')

mixin RxObjectMixin<T> on NotifyManager<T>
  late T _value;
  T get value {
    // RxInterface.proxy 是 Obx的_observer
    RxInterface.proxy?.addListener(subject
    return _value;
  }
}
mixin NotifyManager<T> {
  GetStream<T> subject = GetStream<T>();
  void addListener(GetStream<T> rxGetx) {
    if (!_subscriptions.containsKey(rxGetx
      // 这里是最关键的一步，Obx的subject和0.o
      // rxGetx.listen后续执行请参考上页subje
      final subs = rxGetx.listen((data) {
        if (!subject.isClosed) subject.add
      });
    }
  }
}
```

派发更新

```
counter++

mixin RxObjectMixin<T> on NotifyManager<T>
  set value(T val) {
    if (subject.isClosed) return;
    _value = val;
    subject.add(_value);
  }
}
class GetStream<T> {
  void add(T event) {
    _value = event;
    _notifyData(event);
  }
  void _notifyData(T data) {
    for (final item in _onData!) {
      item._data?.call(data);
    }
  }
  // 依赖收集添加的回调事件被执行
  final subs = rxGetx.listen((data) {
    if (!subject.isClosed) subject.add(dat
  });
}
```

Obx更新

```
// Obx()
void initState() {
  super.initState();
  subs = _observer.listen(_updateTree, can
}

class _ObxState extends State<ObxWidget> {
  void _updateTree(_) {
    if (mounted) {
      setState(() {});
    }
  }
}
```

GetX 源码解析

GetX与Obx的区别在于增加了部分参数进行使用。

```
const GetX({
  this.tag,
  required this.builder,
  this.global = true,
  this.autoRemove = true,
  this.initState,
  this.assignId = false,
  //  this.stream,
  this.dispose,
  this.didChangeDependencies,
  this.didUpdateWidget,
  this.init,
  // this.streamController
});
```