Flutter 重识 NestedScrollView

法的空间 LV.5
2021年08月17日 09:19 · 阅读 6995

「前言」

extended_nested_scroll_view 是我的第一个上传到 pub.dev 的 Flutter 组件.

一晃眼都快3年了，经历了43个版本迭代，功能稳定，代码与官方同步。

0.0.1	2.0 (dev)	Dec 1, 2018

@稀土掘金技术社区

而我最近一直筹备着对其进行重构。怎么说了，接触 Flutter 3年了，认知也与当初有所不同。我相信自己如果现在再面对 `NestedScrollView` 的问题，我应该能处理地更好。

注意：后面用到的 `SliverPinnedToBoxAdapter` 是 extended_sliver里面一个组件，你把它当作 `SliverPersistentHeader`（Pinned 为 true，minExtent = maxExtent）就好了。

「NestedScrollView 是什么」

> A scrolling view inside of which can be nested other scrolling views, with their scroll positions being intrinsically linked.

将外部滚动(Header部分)和内部滚动(Body部分)联动起来。里面滚动不了，滚动外面。外面滚动没了，滚动里面。那么 `NestedScrollView` 是如何做到的呢？

`NestedScrollView` 其实是一个 `CustomScrollView`，下面为伪代码。

```dart 复制代码
CustomScrollView(
  controller: outerController,
  slivers: [
   ...<Widget>[Header1,Header2],
  SliverFillRemaining()(
    child: PrimaryScrollController(
      controller: innerController,
      child: body,
    ),
  ),
  ],
);
```

- outerController 是 `CustomScrollView` 的 `controller`， 从层级上看，就是外部
- 这里使用了 `PrimaryScrollController` ，那么 `body` 里面的任何滚动组件，在不自定义 `controller` 的情况下，都将公用 `innerController` 。

至于为什么会这样，首先看一下每个滚动组件都有的属性 primary，如果 controller 为 null，并且是竖直方法，就默认为 true 。

```
primary = primary ?? controller == null && identical(scrollDirection, Axis.vertical),
```

然后 在 scroll_view.dart 中，如果 `primary` 为 true，就去获取 `PrimaryScrollController` 的 controller。

```dart 复制代码
final ScrollController? scrollController =
    primary ? PrimaryScrollController.of(context) : controller;
final Scrollable scrollable = Scrollable(
  dragStartBehavior: dragStartBehavior,
  axisDirection: axisDirection,
  controller: scrollController,
  physics: physics,
  scrollBehavior: scrollBehavior,
  semanticChildCount: semanticChildCount,
  restorationId: restorationId,
  viewportBuilder: (BuildContext context, ViewportOffset offset) {
    return buildViewport(context, offset, axisDirection, slivers);
  },
);
```

> 这也解释了为啥有些同学给 body 中的滚动组件设置了 controller，就会发现内外滚动不再联动了。

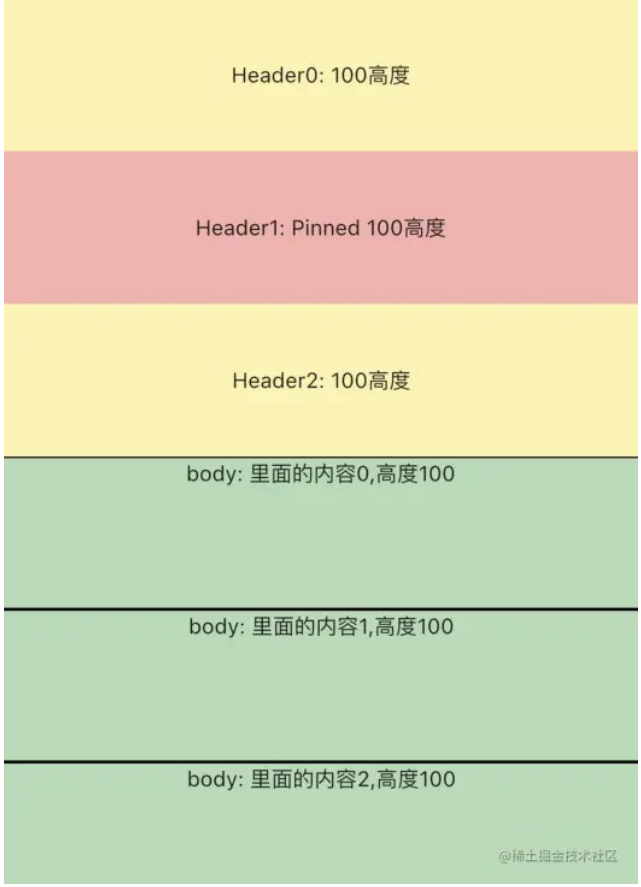
「为什么要扩展官方的」

理解了 `NestedScrollView` 是什么，那我为啥要扩展官方组件呢？

» Header 中包含多个 Pinned Sliver 时候的问题

分析

先看一个图，你觉得列表向上滚动最终的结果是什么？代码在下面。

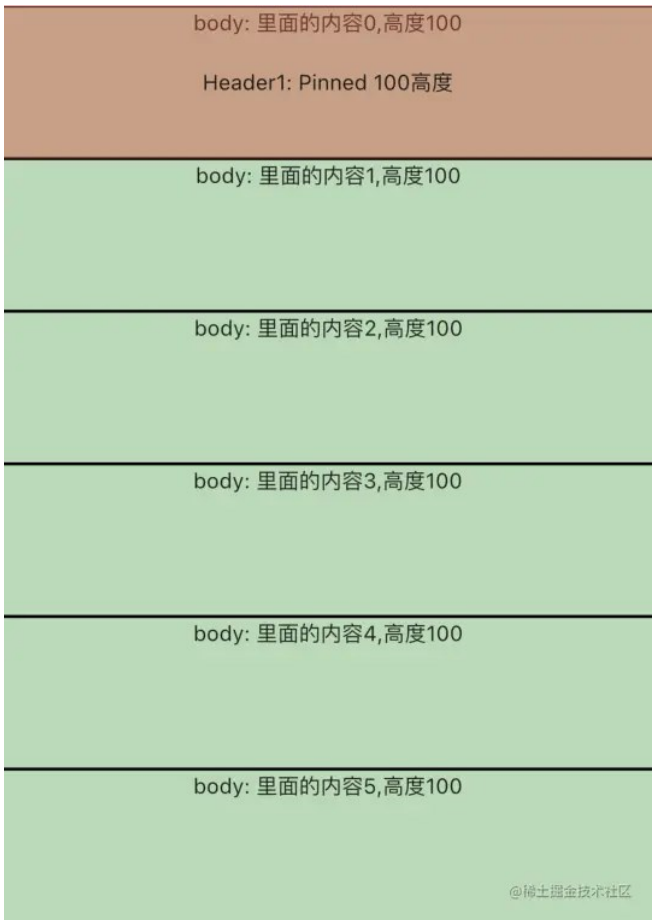


dart 复制代码

```dart
CustomScrollView(
      slivers: <Widget>[
        SliverToBoxAdapter(
          child: Container(
            alignment: Alignment.center,
            child: Text('Header: 100高度'),
            height: 100,
            color: Colors.yellow.withOpacity(0.4),
          ),
        ),
        SliverPinnedToBoxAdapter(
          child: Container(
            alignment: Alignment.center,
            child: Text('Header: Pinned 100高度'),
            height: 100,
            color: Colors.red.withOpacity(0.4),
          ),
        ),
        SliverToBoxAdapter(
          child: Container(
            alignment: Alignment.center,
            child: Text('Header: 100高度'),
            height: 100,
            color: Colors.yellow.withOpacity(0.4),
          ),
        ),
        SliverFillRemaining(
          child: Column(
            children: List.generate(
                100,
                (index) => Container(
                      alignment: Alignment.topCenter,
                      child: Text('body: 里面的内容$index,高度100'),
                      height: 100,
                      decoration: BoxDecoration(
                          color: Colors.green.withOpacity(0.4),
                          border: Border.all(
                            color: Colors.black,
                          )),
                    )),
          ),
        )
      ],
```

```
    ),
```

嗯，没错，列表的第一个 Item 会滚动到 Header1 下面。但实际上，我们通常的需求是需要列表停留在 Header1 底边。



Flutter 官方也注意到了这个问题，并且提供了 `SliverOverlapAbsorber` `SliverOverlapInjector` 来处理这个问题，

- `SliverOverlapAbsorber` 来包裹 `Pinned` 为 `true` 的 `Sliver`
- 在 body 中使用 `SliverOverlapInjector` 来占位
- 用 `NestedScrollView._absorberHandle` 来实现 `SliverOverlapAbsorber` 和 `SliverOverlapInjector` 的信息传递。

dart 复制代码

```dart
return Scaffold(
  body: NestedScrollView(
    headerSliverBuilder: (BuildContext context, bool innerBoxIsScrolled) {
      return <Widget>[
        // 监听计算高度，并且通过 NestedScrollView._absorberHandle 将
        // 自身的高度 告诉 SliverOverlapInjector
        SliverOverlapAbsorber(
          handle: NestedScrollView.sliverOverlapAbsorberHandleFor(context),
          sliver: SliverPinnedToBoxAdapter(
           child: Container(
             alignment: Alignment.center,
             child: Text('Header: Pinned 100高度'),
             height: 100,
             color: Colors.red.withOpacity(0.4),
           ),
         )
        )
      ];
    },
    body: Builder(
      builder: (BuildContext context) {
        return CustomScrollView(
          // The "controller" and "primary" members should be left
          // unset, so that the NestedScrollView can control this
          // inner scroll view.
          // If the "controller" property is set, then this scroll
          // view will not be associated with the NestedScrollView.
          slivers: <Widget>[
            // 占位，接收 SliverOverlapAbsorber 的信息
            SliverOverlapInjector(handle: NestedScrollView.sliverOverlapAbsorberHandleFor(context)),
            SliverFixedExtentList(
              itemExtent: 48.0,
              delegate: SliverChildBuilderDelegate(
                  (BuildContext context, int index) => ListTile(title: Text('Item $index')),
                childCount: 30,
              ),
            ),
          ],
        );
      }
```

```
      )
    )
  );
}
```

如果你觉得这种方法不清楚，那我简化一下，用另外的方式表达。我们也增加一个 100 的占位。不过实际操作中是不可能这样做的，这样会导致初始化的时候列表上方会留下 100 的空位。

```dart 复制代码
CustomScrollView(
      slivers: <Widget>[
        SliverToBoxAdapter(
          child: Container(
            alignment: Alignment.center,
            child: Text('Header0: 100高度'),
            height: 100,
            color: Colors.yellow.withOpacity(0.4),
          ),
        ),
        SliverPinnedToBoxAdapter(
          child: Container(
            alignment: Alignment.center,
            child: Text('Header1: Pinned 100高度'),
            height: 100,
            color: Colors.red.withOpacity(0.4),
          ),
        ),
        SliverToBoxAdapter(
          child: Container(
            alignment: Alignment.center,
            child: Text('Header2: 100高度'),
            height: 100,
            color: Colors.yellow.withOpacity(0.4),
          ),
        ),
        SliverFillRemaining(
          child: Column(
            children: <Widget>[
              // 我相当于 SliverOverlapAbsorber
              Container(
                height: 100,
              ),
              Column(
                children: List.generate(
                    100,
                    (index) => Container(
                          alignment: Alignment.topCenter,
                          child: Text('body: 里面的内容$index,高度100'),
                          height: 100,
                          decoration: BoxDecoration(
                              color: Colors.green.withOpacity(0.4),
                              border: Border.all(
                                color: Colors.black,
                              )),
                        )),
              ),
            ],
          ),
        )
      ],
    ),
```

那问题来了，如果 `NestedScrollView` 的 `Header` 中包含多个 `Pinned` 为 `true` 的 `Sliver`，那么 `SliverOverlapAbsorber` 便无能为力了，Issue 传送门。

解决

我们再来回顾 `NestedScrollView` 长什么样子的，可以看出来，这个问题应该跟 `outerController` 有关系。参照前面简单 demo 来看，只要我们让外部少滚动 100，就可以让列表停留在 Pinned Header1 底部了。

```dart 复制代码
CustomScrollView(
  controller: outerController,
  slivers: [
   ...<Widget>[Header1,Header2],
  SliverFillRemaining()(
    child: PrimaryScrollController(
      controller: innerController,
      child: body,
    ),
  ),
  ],
);
```

maxScrollExtent

我们再思考一下，是什么会影响一个滚动组件的滚动最终距离？

答案是 ScrollPosition.maxScrollExtent

知道了是什么东西影响，我们要做的就是在合适的时候修改这个值，那么如何获取时机呢？

将下面代码

```dart
@override
double get maxScrollExtent => _maxScrollExtent!;
double? _maxScrollExtent;
```

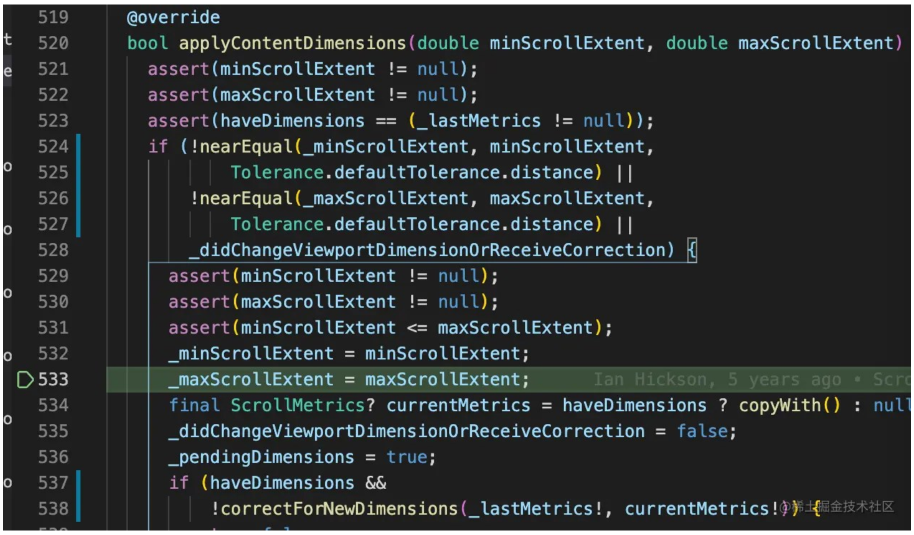
改为以下代码

```dart
@override
double get maxScrollExtent => _maxScrollExtent!;
//double? _maxScrollExtent;
double? __maxScrollExtent;
double? get _maxScrollExtent => __maxScrollExtent;
set _maxScrollExtent(double? value) {
  if (__maxScrollExtent != value) {
    __maxScrollExtent = value;
 }
}
```

这样我们就可以在 set 方法里面打上 debug 断点，看看是什么时候 `_maxScrollExtent` 被赋值的。

运行例子，得到以下 `Call Stack` 。

```
ScrollPosition._maxScrollExtent=        package:flutter/.../widgets/scroll_position.dart
ScrollPosition.applyContentDimensions   package:flutter/.../widgets/scroll_positi...
RenderViewport.performLayout            package:flutter/.../rendering/viewport.dart  1487:20
RenderObject.layout                     package:flutter/.../rendering/object.dart  1779:7
RenderProxyBoxMixin.performLayout       package:flutter/.../rendering/proxy_box.dart
RenderObject.layout                     package:flutter/.../rendering/object.dart  1779:7
RenderProxyBoxMixin.performLayout       package:flutter/.../rendering/proxy_box.dart
RenderObject.layout                     package:flutter/.../rendering/object.dart  1779:7
RenderProxyBoxMixin.performLayout       package:flutter/.../rendering/proxy_box.dart
RenderObject.layout                     package:flutter/.../rendering/object.dart  1779:7
RenderProxyBoxMixin.performLayout       package:flutter/.../rendering/proxy_box.dart
RenderObject.layout                     package:flutter/.../rendering/object.dart  1779:7
RenderProxyBoxMixin.performLayout       package:flutter/.../rendering/proxy_box.dart
RenderObject.layout                     package:flutter/.../rendering/object.dart  1779:7
RenderProxyBoxMixin.performLayout       package:flutter/.../rendering/proxy_box.dart
RenderObject.layout                     package:flutter/.../rendering/object.dart  1779:7
```

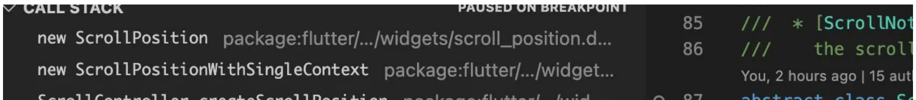


```dart
@override
bool applyContentDimensions(double minScrollExtent, double maxScrollExtent)
  assert(minScrollExtent != null);
  assert(maxScrollExtent != null);
  assert(haveDimensions == (_lastMetrics != null));
  if (!nearEqual(_minScrollExtent, minScrollExtent,
        Tolerance.defaultTolerance.distance) ||
      !nearEqual(_maxScrollExtent, maxScrollExtent,
        Tolerance.defaultTolerance.distance) ||
      _didChangeViewportDimensionOrReceiveCorrection) {
    assert(minScrollExtent != null);
    assert(maxScrollExtent != null);
    assert(minScrollExtent <= maxScrollExtent);
    _minScrollExtent = minScrollExtent;
    _maxScrollExtent = maxScrollExtent;      Ian Hickson, 5 years ago • Scr
    final ScrollMetrics? currentMetrics = haveDimensions ? copyWith() : null
    _didChangeViewportDimensionOrReceiveCorrection = false;
    _pendingDimensions = true;
    if (haveDimensions &&
        !correctForNewDimensions(_lastMetrics!, currentMetrics!)) {
```

看到这里，我们应该知道，可以通过 override `applyContentDimensions` 方法，去重新设置 `maxScrollExtent`

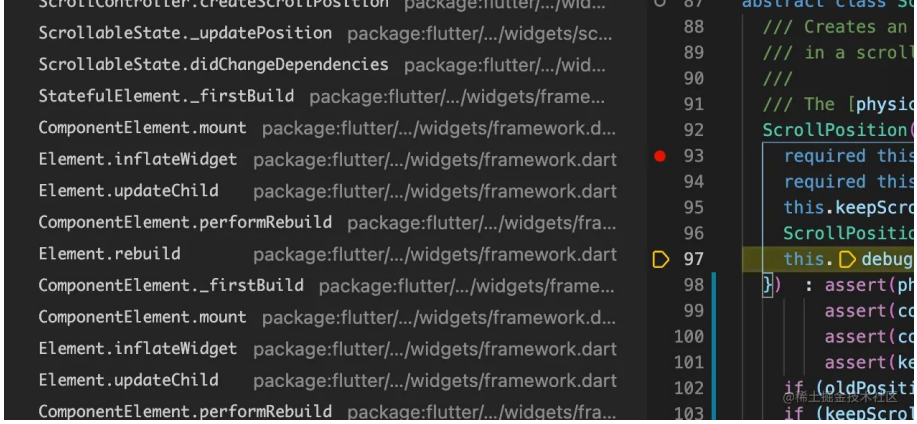
ScrollPosition

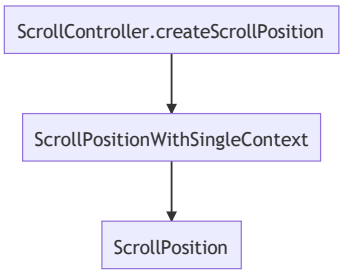
想要 override `applyContentDimensions` 就要知道 `ScrollPosition` 在什么时候创建的，继续调试，把断点打到 `ScrollPosition` 的构造上面。



```
CALL STACK                                                    PAUSED ON BREAKPOINT
  new ScrollPosition  package:flutter/.../widgets/scroll_position.d...
  new ScrollPositionWithSingleContext  package:flutter/.../widget...
```

```
///  * [ScrollNot
///    the scrol
You, 2 hours ago | 15 aut
```





可以看到如果不是特定的 `ScrollPosition`，我们平时使用的是默认的 `ScrollPositionWithSingleContext`，并且在 `ScrollController` 的 `createScrollPosition` 方法中创建。

增加下面的代码，并且给 demo 中的 `CustomScrollView` 添加 `controller` 为 `MyScrollController`，我们再次运行 demo，是不是得到了我们想要的效果呢？

```dart
class MyScrollController extends ScrollController {
  @override
  ScrollPosition createScrollPosition(ScrollPhysics physics,
      ScrollContext context, ScrollPosition oldPosition) {
    return MyScrollPosition(
      physics: physics,
      context: context,
      initialPixels: initialScrollOffset,
      keepScrollOffset: keepScrollOffset,
      oldPosition: oldPosition,
      debugLabel: debugLabel,
    );
  }
}

class MyScrollPosition extends ScrollPositionWithSingleContext {
  MyScrollPosition({
    @required ScrollPhysics physics,
    @required ScrollContext context,
    double initialPixels = 0.0,
    bool keepScrollOffset = true,
    ScrollPosition oldPosition,
    String debugLabel,
  }) : super(
          physics: physics,
          context: context,
          keepScrollOffset: keepScrollOffset,
          oldPosition: oldPosition,
          debugLabel: debugLabel,
          initialPixels: initialPixels,
        );

  @override
  bool applyContentDimensions(double minScrollExtent, double maxScrollExtent) {
    return super.applyContentDimensions(minScrollExtent, maxScrollExtent - 100);
  }
}
```

_NestedScrollPosition

对应到 `NestedScrollView` 中，可以为_NestedScrollPosition 添加以下的方法。

`pinnedHeaderSliverHeightBuilder` 回调是获取 `Header` 当中一共有哪些 `Pinned` 的 `Sliver`。

- 对于 SliverAppbar 来说，最终固定的高度应该包括 状态栏的高度 (MediaQuery.of(context).padding.top) 和 导航栏的高度 (kToolbarHeight)
- 对于 `SliverPersistentHeader`（Pinned 为 true），最终固定高度应该为 `minExtent`
- 如果有多个这种 Sliver，应该为他们最终固定的高度之和。

```dart
  @override
  bool applyContentDimensions(double minScrollExtent, double maxScrollExtent) {
```

```
    if (debugLabel == 'outer' &&
        coordinator.pinnedHeaderSliverHeightBuilder != null) {
      maxScrollExtent =
          maxScrollExtent - coordinator.pinnedHeaderSliverHeightBuilder!();
      maxScrollExtent = math.max(0.0, maxScrollExtent);
    }
    return super.applyContentDimensions(minScrollExtent, maxScrollExtent);
  }
```

» Body 中多列表滚动互相影响的问题

大家一定有这种需求，在 `TabbarView` 或者 `PageView` 中的列表，切换的时候列表的滚动位置要保留。这个使用 `AutomaticKeepAliveClientMixin`，非常简单。

但是如果把 `TabbarView` 或者 `PageView` 放到 `NestedScrollView` 的 `body` 里面的话，你滚动其中一个列表，也会发现其他的列表也会跟着改变位置。Issue 传送门

分析

先看 `NestedScrollView` 的伪代码。`NestedScrollView` 之所以能上内外联动，就是在于 `outerController` 和 `innerController` 的联动。

```dart 复制代码
    CustomScrollView(
      controller: outerController,
      slivers: [
       ...<Widget>[Header1,Header2],
      SliverFillRemaining()(
        child: PrimaryScrollController(
          controller: innerController,
          child: body,
        ),
      ),
      ],
    );
```

`innerController` 负责 `Body`，将 `Body` 中没有设置过 controller 的列表的 `ScrollPosition` 通过 `attach` 方法，加载进来。

> 当使用列表缓存的时候，切换 tab 的时候，原列表将不会 `dispose`，就不会从 controller 中 `detach` 。innerController.positions 将不止一个。而 `outerController` 和 `innerController` 的联动计算都是基于 positions 来进行的。这就是导致这个问题的原因。

具体代码体现在 github.com/flutter/flu...

```dart 复制代码
        if (innerDelta != 0.0) {
          for (final _NestedScrollPosition position in _innerPositions)
            position.applyFullDragUpdate(innerDelta);
        }
```

解决

> 不管是3年前还是现在再看这个问题，第一感觉，不就是只要找到当前 `显示` 的那个列表，只让它滚动就可以了嘛，不是很简单吗？

确实，但是那只是看起来觉得简单，毕竟这个 issue 已经 open 3年了。

老方案

1. 在 `ScrollPosition` `attach` 的时候去通过 `context` 找到这个列表所对应的标志，跟 `TabbarView` 或者 `PageView` 的 index 关联进行对比。Flutter 扩展NestedScrollView（二）列表滚动同步解决 (juejin.cn)
2. 通过计算列表的相对位置，来确定当前 `显示` 的列表。Flutter 你想知道的Widget可视区域,相对位置,大小 (juejin.cn)

总体来说，

- 1方案更准确，但是用法比较繁琐。
- 2方案受动画影响，在一些特殊的情况下会导致计算不正确。

新方案

首先我们先准备一个的 demo 重现问题。

```dart 复制代码
    NestedScrollView(
      headerSliverBuilder: (
        BuildContext buildContext,
        bool innerBoxIsScrolled,
      ) =>
          <Widget>[
        SliverToBoxAdapter(
          child: Container(
```

```
            color: Colors.red,
            height: 200,
          ),
        )
      ],
      body: Column(
        children: [
          Container(
            color: Colors.yellow,
            height: 200,
          ),
          Expanded(
            child: PageView(
              children: <Widget>[
                ListItem(
                  tag: 'Tab0',
                ),
                ListItem(
                  tag: 'Tab1',
                ),
              ],
            ),
          ),
        ],
      ),
    ),

 class ListItem extends StatefulWidget {
  const ListItem({
    Key key,
    this.tag,
  }) : super(key: key);
  final String tag;

  @override
  _ListItemState createState() => _ListItemState();
}

class _ListItemState extends State<ListItem>
    with AutomaticKeepAliveClientMixin {
  @override
  Widget build(BuildContext context) {
    super.build(context);
    return ListView.builder(
      itemBuilder: (BuildContext buildContext, int index) =>
          Center(child: Text('${widget.tag}---$index')),
      itemCount: 1000,
    );
  }

  @override
  bool get wantKeepAlive => true;
}
```

Drag

现在再看这个问题，我在思考，我自己滚动了哪个列表，我自己不知道？？

看过上一篇 Flutter 锁定行列的FlexGrid - 掘金 (juejin.cn) 的小伙伴，应该知道在拖拽列表的时候是会生成一个 `Drag` 的。那么有这个 `Drag` 的 `ScrollPosition` 不就对应正在显示的列表吗？？

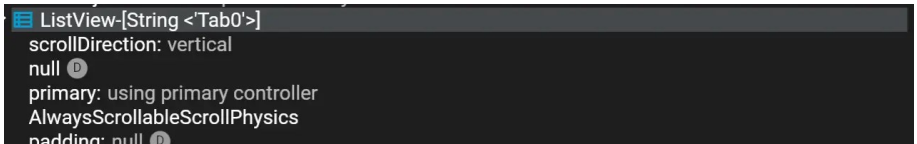
具体到代码，我们试试打日志看看，

github.com/flutter/flu...

```dart
  @override
  Drag drag(DragStartDetails details, VoidCallback dragCancelCallback) {
    print(debugLabel);
    return coordinator.drag(details, dragCancelCallback);
  }
```

理想很好，但是现实是骨感的，不管我是滚动 `Header` 还是 `Body` ，都只打印了 `outer` 。那意思是 Body 里面的手势全部被吃了？？

不着急，我们打开 `DevTools` ，看看 `ListView` 里面的 ScrollableState 的状态。(具体为啥要看这里面，可以去读读 Flutter 锁定行列的 FlexGrid (juejin.cn))



```
itemExtent: null
dependencies: [PrimaryScrollController, MediaQuery]
PrimaryScrollController
  no controller
  Scrollable
    axisDirection: down
    physics: AlwaysScrollableScrollPhysics
    restorationId: null
    dependencies: [_LocalizationsScope-[GlobalKey#11fde], UnmanagedRestorationScope, …
    state: ScrollableState#f6ada(position: _NestedScrollPosition#c00a6(inner, offset: 0.0, range: …
    _ScrollSemantics-[GlobalKey#3668f]
      renderObject: _RenderScrollSemantics#8bb83
      _ScrollableScope
        Listener
          listeners: signal
          behavior: deferToChild
          renderObject: RenderPointerListener#622d3
          RawGestureDetector-[LabeledGlobalKey<RawGestureDetectorState>#183bb]
            state: RawGestureDetectorState#59f83(gestures: <none>, behavior: opaque)
              gestures: <none>
              behavior: opaque
            _GestureSemantics
              renderObject: RenderSemanticsGestureHandler#ee8af
              Listener
```


哈哈，`gestures` 居然为 `none`，就是说 `Body` 里面没有注册手势。

github.com/flutter/flu... `setCanDrag` 方法中，我们可以看到只有 `canDrag` 等于 `false` 的时候，我们是没有注册手势的。当然也有一种可能，`setCanDrag` 也许就没有被调用过，默认的 `_gestureRecognizers` 就是空。

dart 复制代码

```dart
@override
@protected
void setCanDrag(bool canDrag) {
  if (canDrag == _lastCanDrag && (!canDrag || widget.axis == _lastAxisDirection))
    return;
  if (!canDrag) {
    _gestureRecognizers = const <Type, GestureRecognizerFactory>{};
    // Cancel the active hold/drag (if any) because the gesture recognizers
    // will soon be disposed by our RawGestureDetector, and we won't be
    // receiving pointer up events to cancel the hold/drag.
    _handleDragCancel();
  } else {
    switch (widget.axis) {
      case Axis.vertical:
        _gestureRecognizers = <Type, GestureRecognizerFactory>{
          VerticalDragGestureRecognizer: GestureRecognizerFactoryWithHandlers<VerticalDragGestureRecogni
            () => VerticalDragGestureRecognizer(),
            (VerticalDragGestureRecognizer instance) {
              instance
                ..onDown = _handleDragDown
                ..onStart = _handleDragStart
                ..onUpdate = _handleDragUpdate
                ..onEnd = _handleDragEnd
                ..onCancel = _handleDragCancel
                ..minFlingDistance = _physics?.minFlingDistance
                ..minFlingVelocity = _physics?.minFlingVelocity
                ..maxFlingVelocity = _physics?.maxFlingVelocity
                ..velocityTrackerBuilder = _configuration.velocityTrackerBuilder(context)
                ..dragStartBehavior = widget.dragStartBehavior;
            },
          ),
        };
        break;
      case Axis.horizontal:
        _gestureRecognizers = <Type, GestureRecognizerFactory>{
          HorizontalDragGestureRecognizer: GestureRecognizerFactoryWithHandlers<HorizontalDragGestureRec
            () => HorizontalDragGestureRecognizer(),
            (HorizontalDragGestureRecognizer instance) {
              instance
                ..onDown = _handleDragDown
                ..onStart = _handleDragStart
                ..onUpdate = _handleDragUpdate
                ..onEnd = _handleDragEnd
                ..onCancel = _handleDragCancel
                ..minFlingDistance = _physics?.minFlingDistance
                ..minFlingVelocity = _physics?.minFlingVelocity
                ..maxFlingVelocity = _physics?.maxFlingVelocity
                ..velocityTrackerBuilder = _configuration.velocityTrackerBuilder(context)
                ..dragStartBehavior = widget.dragStartBehavior;
            },
          ),
        };
        break;
    }
  }
  _lastCanDrag = canDrag;
  _lastAxisDirection = widget.axis;
  if (_gestureDetectorKey.currentState != null)
```

```
    _gestureDetectorKey.currentState!.replaceGestureRecognizers(_gestureRecognizers);
  }
```

我们在 `setCanDrag` 方法中打一个断点，看看调用的时机。

```
ScrollableState.setCanDrag                          package:flutter/.../widgets/scrollable.dart   543:20
ScrollPositionWithSingleContext.applyNewDimensions  package:flutter/.../widgets/scroll_pos...
ScrollPosition.applyContentDimensions               package:flutter/.../widgets/scroll_position.dart   533:7
RenderViewport.performLayout                        package:flutter/.../rendering/viewport.dart   1437:20
```

1. RenderViewport.performLayout

performLayout 方法中计算出当前 `ScrollPosition` 的最小最大值

```dart
    if (offset.applyContentDimensions(
            math.min(0.0, _minScrollExtent + mainAxisExtent * anchor),
            math.max(0.0, _maxScrollExtent - mainAxisExtent * (1.0 - anchor)),
        ))
```

2. ScrollPosition.applyContentDimensions

调用 `applyNewDimensions` 方法

```dart
  @override
  bool applyContentDimensions(double minScrollExtent, double maxScrollExtent) {
    assert(minScrollExtent != null);
    assert(maxScrollExtent != null);
    assert(haveDimensions == (_lastMetrics != null));
    if (!nearEqual(_minScrollExtent, minScrollExtent, Tolerance.defaultTolerance.distance) ||
        !nearEqual(_maxScrollExtent, maxScrollExtent, Tolerance.defaultTolerance.distance) ||
        _didChangeViewportDimensionOrReceiveCorrection) {
      assert(minScrollExtent != null);
      assert(maxScrollExtent != null);
      assert(minScrollExtent <= maxScrollExtent);
      _minScrollExtent = minScrollExtent;
      _maxScrollExtent = maxScrollExtent;
      final ScrollMetrics? currentMetrics = haveDimensions ? copyWith() : null;
      _didChangeViewportDimensionOrReceiveCorrection = false;
      _pendingDimensions = true;
      if (haveDimensions && !correctForNewDimensions(_lastMetrics!, currentMetrics!)) {
        return false;
      }
      _haveDimensions = true;
    }
    assert(haveDimensions);
    if (_pendingDimensions) {
      applyNewDimensions();
      _pendingDimensions = false;
    }
    assert(!_didChangeViewportDimensionOrReceiveCorrection, 'Use correctForNewDimensions() (and return tru
    _lastMetrics = copyWith();
    return true;
  }
```

3. ScrollPositionWithSingleContext.applyNewDimensions

不特殊定义的话，默认 `ScrollPosition` 都是 `ScrollPositionWithSingleContext` 。 `context` 是谁呢？ 当然是 `ScrollableState`

```dart
  @override
  void applyNewDimensions() {
    super.applyNewDimensions();
    context.setCanDrag(physics.shouldAcceptUserOffset(this));
  }
```

这里提了一下，平时有同学问。不满一屏幕的列表 controller 注册不触发 或者 NotificationListener 监听不触发。原因就在这里，physics.shouldAcceptUserOffset(this) 返回的是 `false` 。而我们的处理办法就是 设置 physics 为 `AlwaysScrollableScrollPhysics` ， shouldAcceptUserOffset 放

`AlwaysScrollableScrollPhysics` 的 `shouldAcceptUserOffset` 方法永远返回 `true` 。

```dart
class AlwaysScrollableScrollPhysics extends ScrollPhysics {
  /// Creates scroll physics that always lets the user scroll.
  const AlwaysScrollableScrollPhysics({ ScrollPhysics? parent }) : super(parent: parent);

  @override
  AlwaysScrollableScrollPhysics applyTo(ScrollPhysics? ancestor) {
    return AlwaysScrollableScrollPhysics(parent: buildParent(ancestor));
  }

  @override
```

```
  bool shouldAcceptUserOffset(ScrollMetrics position) => true;
}
```

4. ScrollableState.setCanDrag

最终达到这里，去根据 `canDrag` 和 `axis` (水平/垂直)

_NestedScrollCoordinator

那接下来，我们就去 `NestedScrollView` 代码里面找找看。

github.com/flutter/flu...

```dart 复制代码
@override
void applyNewDimensions() {
  super.applyNewDimensions();
  coordinator.updateCanDrag();
}
```

这里我们看到调用了 `coordinator.updateCanDrag()` 。

首先我们看看 `coordinator` 是什么？不难看出来，用来协调 `outerController` 和 `innerController` 的。

```dart 复制代码
class _NestedScrollCoordinator
    implements ScrollActivityDelegate, ScrollHoldController {
  _NestedScrollCoordinator(
    this._state,
    this._parent,
    this._onHasScrolledBodyChanged,
    this._floatHeaderSlivers,
  ) {
    final double initialScrollOffset = _parent?.initialScrollOffset ?? 0.0;
    _outerController = _NestedScrollController(
      this,
      initialScrollOffset: initialScrollOffset,
      debugLabel: 'outer',
    );
    _innerController = _NestedScrollController(
      this,
      initialScrollOffset: 0.0,
      debugLabel: 'inner',
    );
  }
```

那么我们看看 `updateCanDrag` 方法里面做了什么。

```dart 复制代码
void updateCanDrag() {
  if (!_outerPosition!.haveDimensions) return;
  double maxInnerExtent = 0.0;
  for (final _NestedScrollPosition position in _innerPositions) {
    if (!position.haveDimensions) return;
    maxInnerExtent = math.max(
      maxInnerExtent,
      position.maxScrollExtent - position.minScrollExtent,
    );
  }
  // _NestedScrollPosition.updateCanDrag
  _outerPosition!.updateCanDrag(maxInnerExtent);
}
```

`_NestedScrollPosition.updateCanDrag`

```dart 复制代码
void updateCanDrag(double totalExtent) {
  // 调用 ScrollableState 的 setCanDrag 方法
  context.setCanDrag(totalExtent > (viewportDimension - maxScrollExtent) ||
      minScrollExtent != maxScrollExtent);
}
```

知道原因之后，我们试试动手改下。

- 修改 `_NestedScrollCoordinator.updateCanDrag` 为如下:

```dart 复制代码
void updateCanDrag({_NestedScrollPosition? position}) {
  double maxInnerExtent = 0.0;

  if (position != null && position.debugLabel == 'inner') {
    if (position.haveDimensions) {
      maxInnerExtent = math.max(
        maxInnerExtent,
        position.maxScrollExtent - position.minScrollExtent,
      );
      position.updateCanDrag(maxInnerExtent);
    }
```

```
  }
  if (!_outerPosition!.haveDimensions) {
    return;
  }

  for (final _NestedScrollPosition position in _innerPositions) {
    if (!position.haveDimensions) {
      return;
    }
    maxInnerExtent = math.max(
      maxInnerExtent,
      position.maxScrollExtent - position.minScrollExtent,
    );
  }
  _outerPosition!.updateCanDrag(maxInnerExtent);
}
```

- 修改 `_NestedScrollPosition.drag` 方法为如下:

```dart
bool _isActived = false;
@override
Drag drag(DragStartDetails details, VoidCallback dragCancelCallback) {
  _isActived = true;
  return coordinator.drag(details, () {
    dragCancelCallback();
    _isActived = false;
  });
}

/// Whether is actived now
bool get isActived {
  return _isActived;
}
```

- 修改 `_NestedScrollCoordinator._innerPositions` 为如下:

```dart
Iterable<_NestedScrollPosition> get _innerPositions {
  if (_innerController.nestedPositions.length > 1) {
    final Iterable<_NestedScrollPosition> actived = _innerController
        .nestedPositions
        .where((_NestedScrollPosition element) => element.isActived);
    print('${actived.length}');
    if (actived.isNotEmpty) return actived;
  }
  return _innerController.nestedPositions;
}
```

现在再运行 demo，切换列表之后滚动看看，是否□了？结果是失望的。

1. 虽然我们在 `drag` 操作的时候，确实可以判断到谁是激活的，但是手指 up，开始惯性滑动的时候，`dragCancelCallback` 回调已经触发，`_isActived` 已经被设置为 `false` 。
2. 当我们在操作 `PageView` 上方黄色区域的时候(通常情况下，这部分可能是 `Tabbar`)，由于没有在列表上面进行 `drag` 操作，所以这个时候 `actived` 的列表为 0.

```dart
    NestedScrollView(
      headerSliverBuilder: (
        BuildContext buildContext,
        bool innerBoxIsScrolled,
      ) =>
          <Widget>[
        SliverToBoxAdapter(
          child: Container(
            color: Colors.red,
            height: 200,
          ),
        )
      ],
      body: Column(
        children: [
          Container(
            color: Colors.yellow,
            height: 200,
          ),
          Expanded(
            child: PageView(
              children: <Widget>[
                ListItem(
                  tag: 'Tab0',
                ),
                ListItem(
                  tag: 'Tab1',
                ),
              ],
            ),
          ),
```

```
        ],
      ),
    ),
```

是否可见

问题好像又走到了老地方，怎么判断一个视图是可见。

首先，我们这里能拿到最直接的就是 `_NestedScrollPosition`，我们看看这个家伙有什么东西可以利用。

```
_EfficientLengthCastIterable ([_NestedScrollPosition#a89ac(inner, offset: 6110.…
  > _listeners: LinkedList ((Instance of '_ListenerEntry', Instance of '_Listene
    _maxScrollExtent: 16411.0
    _minScrollExtent: 0.0
    _parent: null
    _pendingDimensions: false
    _pixels: 6110.887344887594
  > _semanticActions: _CompactLinkedHashSet ({SemanticsAction.scrollDown, Semant
    _viewportDimension: 623.0
  > context: ScrollableState (ScrollableState#974d4(tickers: tracking 0 tickers,
  > coordinator: _NestedScrollCoordinator (_NestedScrollCoordinator(outer=_Neste
    debugLabel: "inner"
  > isScrollingNotifier: ValueNotifier (ValueNotifier<bool>#1a76a(false))
    keepScrollOffset: true
  > physics: AlwaysScrollableScrollPhysics (AlwaysScrollableScrollPhysics -> Bou
```

一眼就看到了 `context(ScrollableState)`，是一个 `ScrollContext`，而 `ScrollableState` 实现了 `ScrollContext`。

dart 复制代码

```dart
  /// Where the scrolling is taking place.
  ///
  /// Typically implemented by [ScrollableState].
  final ScrollContext context;
```

看一眼 `ScrollContext`， `notificationContext` 和 `storageContext` 应该是相关的。

dart 复制代码

```dart
abstract class ScrollContext {
  /// The [BuildContext] that should be used when dispatching
  /// [ScrollNotification]s.
  ///
  /// This context is typically different that the context of the scrollable
  /// widget itself. For example, [Scrollable] uses a context outside the
  /// [Viewport] but inside the widgets created by
  /// [ScrollBehavior.buildOverscrollIndicator] and [ScrollBehavior.buildScrollbar].
  BuildContext? get notificationContext;

  /// The [BuildContext] that should be used when searching for a [PageStorage].
  ///
  /// This context is typically the context of the scrollable widget itself. In
  /// particular, it should involve any [GlobalKey]s that are dynamically
  /// created as part of creating the scrolling widget, since those would be
  /// different each time the widget is created.
  // TODO(goderbauer): Deprecate this when state restoration supports all features of PageStorage.
  BuildContext get storageContext;

  /// A [TickerProvider] to use when animating the scroll position.
  TickerProvider get vsync;

  /// The direction in which the widget scrolls.
  AxisDirection get axisDirection;

  /// Whether the contents of the widget should ignore [PointerEvent] inputs.
  ///
  /// Setting this value to true prevents the use from interacting with the
  /// contents of the widget with pointer events. The widget itself is still
  /// interactive.
  ///
  /// For example, if the scroll position is being driven by an animation, it
  /// might be appropriate to set this value to ignore pointer events to
  /// prevent the user from accidentally interacting with the contents of the
  /// widget as it animates. The user will still be able to touch the widget,
  /// potentially stopping the animation.
  void setIgnorePointer(bool value);

  /// Whether the user can drag the widget, for example to initiate a scroll.
  void setCanDrag(bool value);

  /// Set the [SemanticsAction]s that should be expose to the semantics tree.
  void setSemanticsActions(Set<SemanticsAction> actions);

  /// Called by the [ScrollPosition] whenever scrolling ends to persist the
  /// provided scroll `offset` for state restoration purposes.
  ///
  /// The [ScrollContext] may pass the value back to a [ScrollPosition] by
  /// calling [ScrollPosition.restoreOffset] at a later point in time or after
```

```
  /// the application has restarted to restore the scroll offset.
  void saveOffset(double offset);
}
```

再看看 `ScrollableState` 中的实现。

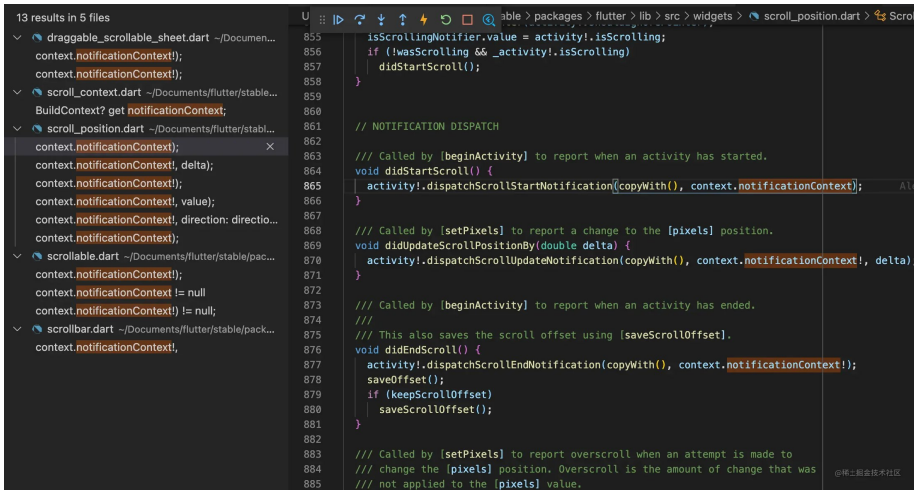
```dart
class ScrollableState extends State<Scrollable> with TickerProviderStateMixin, RestorationMixin
    implements ScrollContext {

  @override
  BuildContext? get notificationContext => _gestureDetectorKey.currentContext;

  @override
  BuildContext get storageContext => context;

}
```

- `storageContext` 其实是 `ScrollableState` 的 `context`。
- `notificationContext` 查找下引用，可以看到。



果然，谁触发的事件，当然是 `ScrollableState` 里面的 `RawGestureDetector` 。

```dart
  NotificationListener<ScrollNotification>(
    onNotification: (ScrollNotification scrollNotification) {
/// The build context of the widget that fired this notification.
///
/// This can be used to find the scrollable's render objects to determine the
/// size of the viewport, for instance.
// final BuildContext? context;
      print(scrollNotification.context);
      return false;
    },
  );
```

最终我们还是要在 `storageContext` 上面下功夫了。之前 # Flutter Sliver一生之敌 # 系列里面我们对 `Sliver` 相关知识进行过梳理。对于 `TabbarView` 或者 `PageView` 当前显示的元素，在 `RenderSliverFillViewport` 当中应该是唯一的(除非你把 `viewportFraction` 的值设置为小于 `1` 的数值)。我们可以通过 `_NestedScrollPosition` 的 `Context` 向上找到 `RenderSliverFillViewport`，看看 `RenderSliverFillViewport` 中的 child 是否为 `_NestedScrollPosition` 的 `Context` 。

- 修改 `_NestedScrollCoordinator._innerPositions` 为如下:

```dart
Iterable<_NestedScrollPosition> get _innerPositions {
  if (_innerController.nestedPositions.length > 1) {
    final Iterable<_NestedScrollPosition> actived = _innerController
        .nestedPositions
        .where((_NestedScrollPosition element) => element.isActived);
    if (actived.isEmpty) {
      for (final _NestedScrollPosition scrollPosition
          in _innerController.nestedPositions) {
        final RenderObject? renderObject =
            scrollPosition.context.storageContext.findRenderObject();

        if (renderObject == null || !renderObject.attached) {
          continue;
        }

        if (renderObjectIsVisible(renderObject, Axis.horizontal)) {
          return <_NestedScrollPosition>[scrollPosition];
        }
      }
      return _innerController.nestedPositions;
    }
```

```
    return actived;
  } else {
    return _innerController.nestedPositions;
  }
}
```

- 在 `renderObjectIsVisible` 方法中查看是否存在于 `TabbarView` 或者 `PageView` 中，并且其 `axis` 与 `ScrollPosition` 的 `axis` 相垂直。如果有的话，用 `RenderViewport` 当前的 `child` 调用 `childIsVisible` 方法验证是否包含 `ScrollPosition` 所对应的 `RenderObject`。注意，这里调用了 `renderObjectIsVisible` 因为可能有嵌套(多级)的 `TabbarView` 或者 `PageView`。

dart 复制代码

```dart
bool renderObjectIsVisible(RenderObject renderObject, Axis axis) {
  final RenderViewport? parent = findParentRenderViewport(renderObject);
  if (parent != null && parent.axis == axis) {
    for (final RenderSliver childrenInPaint
        in parent.childrenInHitTestOrder) {
      return childIsVisible(childrenInPaint, renderObject) &&
          renderObjectIsVisible(parent, axis);
    }
  }
  return true;
}
```

- 向上寻找 `RenderViewport`，我们只在 `NestedScrollView` 的 `body` 的中找，直到 `_ExtendedRenderSliverFillRemainingWithScrollable`。

dart 复制代码

```dart
RenderViewport? findParentRenderViewport(RenderObject? object) {
  if (object == null) {
    return null;
  }
  object = object.parent as RenderObject?;
  while (object != null) {
    // 只在 body 中寻找
    if (object is _ExtendedRenderSliverFillRemainingWithScrollable) {
      return null;
    }
    if (object is RenderViewport) {
      return object;
    }
    object = object.parent as RenderObject?;
  }
  return null;
}
```

- 调用 `visitChildrenForSemantics` 遍历 `children`，看是否能找到 `ScrollPosition` 所对应的 `RenderObject`

dart 复制代码

```dart
  /// Return whether renderObject is visible in parent
bool childIsVisible(
  RenderObject parent,
  RenderObject renderObject,
) {
  bool visible = false;

  // The implementation has to return the children in paint order skipping all
  // children that are not semantically relevant (e.g. because they are
  // invisible).
  parent.visitChildrenForSemantics((RenderObject child) {
    if (renderObject == child) {
      visible = true;
    } else {
      visible = childIsVisible(child, renderObject);
    }
  });
  return visible;
}
```

还有其他方案吗

其实对于 Body 中多列表滚动互相影响的问题，如果你只是要求列表保持位置的话，你完全可以利用 `PageStorageKey` 来保持滚动列表的位置。这样的话，`TabbarView` 或者 `PageView` 切换的时候，`ScrollableState` 会 `dispose`，并且从将 `ScrollPosition` 从 `innerController` 中 `detach` 掉。

dart 复制代码

```dart
@override
void dispose() {
  if (widget.controller != null) {
    widget.controller!.detach(position);
  } else {
    _fallbackScrollController?.detach(position);
    _fallbackScrollController?.dispose();
  }

  position.dispose();
```

```
    _persistedScrollOffset.dispose();
    super.dispose();
  }
```

而你需要做的是在上一层，利用比如 provider | Flutter Package (flutter-io.cn) 来保持列表数据或者其他数据状态。

```dart 复制代码
NestedScrollView(
    headerSliverBuilder: (
      BuildContext buildContext,
      bool innerBoxIsScrolled,
    ) =>
        <Widget>[
      SliverToBoxAdapter(
        child: Container(
          color: Colors.red,
          height: 200,
        ),
      )
    ],
    body: Column(
      children: <Widget>[
        Container(
          color: Colors.yellow,
          height: 200,
        ),
        Expanded(
          child: PageView(
            //controller: PageController(viewportFraction: 0.8),
            children: <Widget>[
              ListView.builder(
                //store Page state
                key: const PageStorageKey<String>('Tab0'),
                physics: const ClampingScrollPhysics(),
                itemBuilder: (BuildContext c, int i) {
                  return Container(
                    alignment: Alignment.center,
                    height: 60.0,
                    child:
                        Text(const Key('Tab0').toString() + ': ListView$i'),
                  );
                },
                itemCount: 50,
              ),
              ListView.builder(
                //store Page state
                key: const PageStorageKey<String>('Tab1'),
                physics: const ClampingScrollPhysics(),
                itemBuilder: (BuildContext c, int i) {
                  return Container(
                    alignment: Alignment.center,
                    height: 60.0,
                    child:
                        Text(const Key('Tab1').toString() + ': ListView$i'),
                  );
                },
                itemCount: 50,
              ),
            ],
          ),
        ),
      ],
    ),
  ),
```

「重构代码」

» 体力活

3年不知不觉就写了 18 个 Flutter 组件库和 3 个 Flutter 相关 工具。

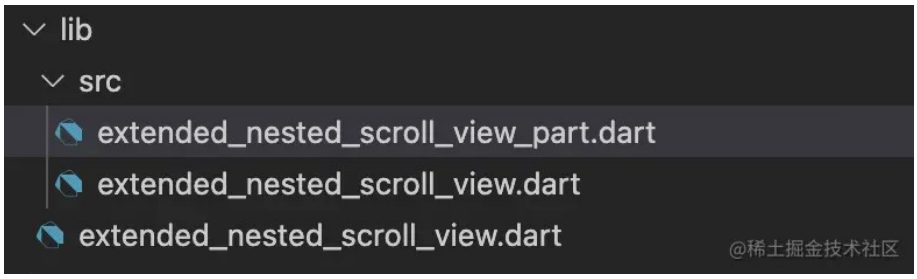
1. like_button | Flutter Package (flutter-io.cn)
2. extended_image_library | Flutter Package (pub.dev)
3. extended_nested_scroll_view | Flutter Package (flutter-io.cn)
4. extended_text | Flutter Package (flutter-io.cn)
5. extended_text_field | Flutter Package (flutter-io.cn)
6. extended_image | Flutter Package (flutter-io.cn)
7. extended_sliver | Flutter Package (flutter-io.cn)
8. pull_to_refresh_notification | Flutter Package (flutter-io.cn)

9. waterfall_flow | Flutter Package (flutter-io.cn)
10. loading_more_list | Flutter Package (flutter-io.cn)
11. extended_tabs | Flutter Package (flutter-io.cn)
12. http_client_helper | Dart Package (flutter-io.cn)
13. extended_text_library | Flutter Package (flutter-io.cn)
14. extended_list | Flutter Package (flutter-io.cn)
15. extended_list_library | Flutter Package (flutter-io.cn)
16. ff_annotation_route_library | Flutter Package (flutter-io.cn)
17. loading_more_list_library | Dart Package (flutter-io.cn)
18. ff_annotation_route | Dart Package (flutter-io.cn)
19. ff_annotation_route_core | Dart Package (flutter-io.cn)
20. flex_grid | Flutter Package (flutter-io.cn)
21. assets_generator | Dart Package (flutter-io.cn)
22. fluttercandies/JsonToDart: The tool to convert json to dart code, support Windows，Mac，Web. (github.com)

可以说每一次官方发布 `Stable` 版本，对于我来说都是一次体力活。特别是 extended_nested_scroll_view，extended_text，extended_text_field，extended_image 这 4 个库，`merge` 代码是不光是体力活，也需要认真仔细去理解新改动。

» 结构重构

这次乘着这个改动的机会，我将整个结构做了调整。



- `src/extended_nested_scroll_view.dart` 为官方源码，只做了一些必要改动。比如增加参数，替换扩展类型。最大程度的保持官方源码的结构和格式。
- `src/extended_nested_scroll_view_part.dart` 为扩展官方组件功能的部分代码。增加下面3个扩展类，实现我们相应的扩展方法。

```dart
class _ExtendedNestedScrollCoordinator extends _NestedScrollCoordinator
```

```dart
class _ExtendedNestedScrollController extends _NestedScrollController
```

```dart
class _ExtendedNestedScrollPosition extends _NestedScrollPosition
```

最后在 `src/extended_nested_scroll_view.dart` 修改初始化代码即可。以后我只需要用 `src/extended_nested_scroll_view.dart` 跟官方的代码进行 `merge` 即可。

```dart
  _NestedScrollCoordinator? _coordinator;

  @override
  void initState() {
    super.initState();
    _coordinator = _ExtendedNestedScrollCoordinator(
      this,
      widget.controller,
      _handleHasScrolledBodyChanged,
      widget.floatHeaderSlivers,
      widget.pinnedHeaderSliverHeightBuilder,
      widget.onlyOneScrollInBody,
      widget.scrollDirection,
    );
  }
```

「小糖果」

字数: 6059　行数: 1320

如果你看到这里，已经看了6000字，感谢。送上一些的技巧，希望能对你有所帮助。

» CustomScrollView center

`CustomScrollView.center` 这个属性我其实很早之前就讲过了，Flutter Sliver一生之敌 (ScrollView) (juejin.cn)。简单地来说：

- `center` 是开始绘制的地方，既绘制在 `zero scroll offset` 的地方，向前为负，向后为正。
- `center` 之前的 `Sliver` 是倒序绘制。

比如下面代码，你觉得最终的效果是什么样子的？

```dart
CustomScrollView(
    center: key,
    slivers: <Widget>[
    SliverList(),
    SliverGrid(key:key),
    ]
)
```

效果图如下，`SliverGrid` 被绘制在了开始位置。你可以向下滚动，这个时候，上面的 `SliverList` 才会展示。

`CustomScrollView.anchor` 可以控制 `center` 的位置。0 为 viewport 的 leading，1 为 viewport 的 trailing，既这个是 viewport 高度垂直(宽度水平)的占比。比如如果是 0.5，那么绘制 `SliverGrid` 的地方就会在 `viewport` 的中间位置。

通过这2个属性，我们可以创造一些有趣的效果。

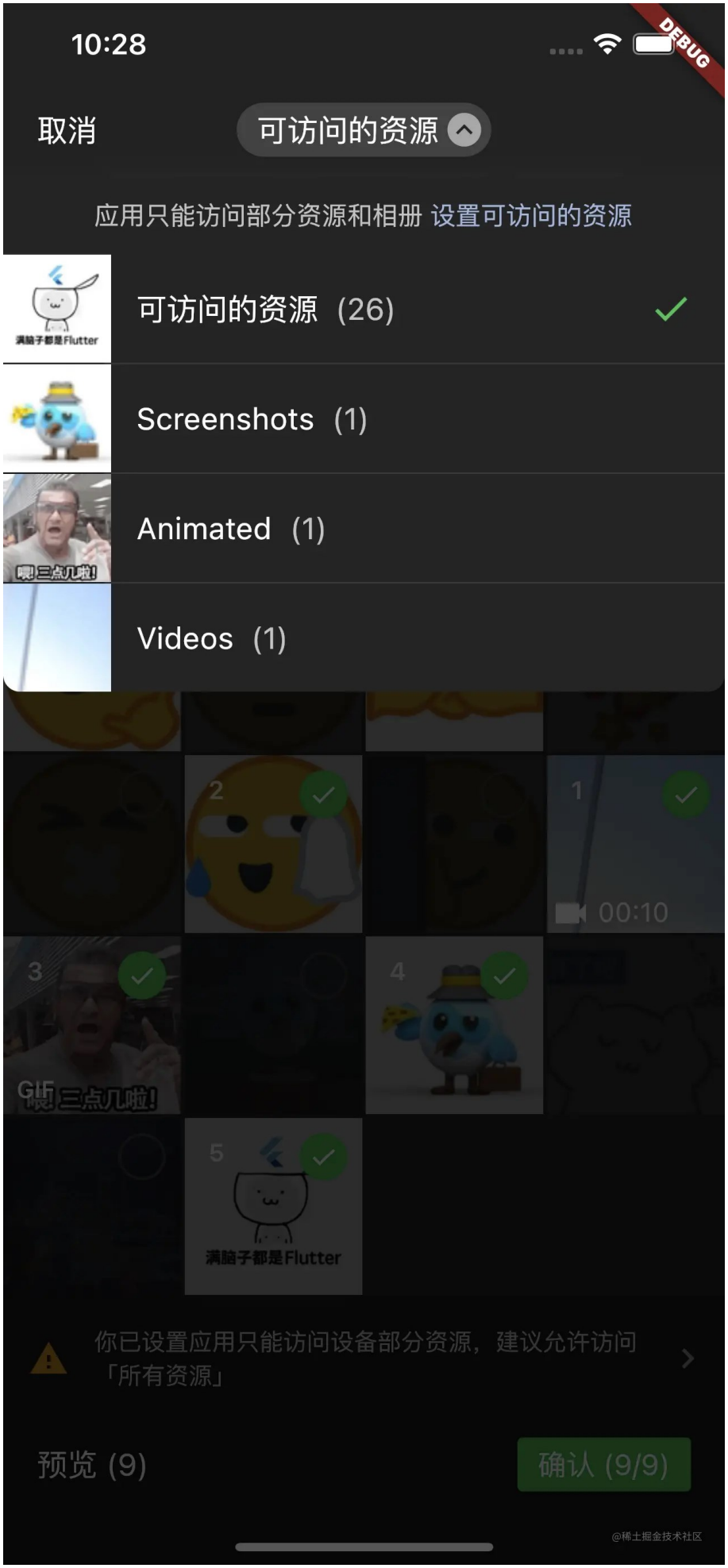
聊天列表

flutter_instant_messaging/main.dart at master · fluttercandies/flutter_instant_messaging (github.com) 一年前写的小 demo，现在移到 flutter_challenges/chat_sample.dart at main · fluttercandies/flutter_challenges (github.com) 统一维护。

ios 倒序相册

flutter_challenges/ios_photo album.dart at main · fluttercandies/flutter_challenges (github.com) 代码在此。

起源于马师傅给 wechat_assets_picker | Flutter Package (flutter-io.cn)提的需求(尾款都没有结)，要让相册查看效果跟 Ios 原生的一样。Ios 的设计果然不一样，学习(chao)就是了。



» 斗鱼首页滚动效果

flutter_challenges/float_scroll.dart at main · fluttercandies/flutter_challenges (github.com) 代码在此。

不得不再提提，`NotificationListener`，它是 `Notification` 的监听者。通过 `Notification.dispatch`，通知会沿着当前节点（BuildContext）向上传递，就跟冒泡一样，你可以在父节点使用 `NotificationListener` 来接受通知。Flutter 中经常使用到的是 `ScrollNotification`，除此之外还有 `SizeChangedLayoutNotification`、`KeepAliveNotification`、`LayoutChangedNotification` 等。你也可以自己定义一个通知。

dart 复制代码

```dart
import 'package:flutter/material.dart';
import 'package:oktoast/oktoast.dart';

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({Key key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return OKToast(
      child: MaterialApp(
        title: 'Flutter Demo',
        theme: ThemeData(
          primarySwatch: Colors.blue,
          visualDensity: VisualDensity.adaptivePlatformDensity,
        ),
        home: MyHomePage(),
      ),
    );
  }
}

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

  @override
  _MyHomePageState createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
```

```
  @override
  Widget build(BuildContext context) {
    return NotificationListener<TextNotification>(
      onNotification: (TextNotification notification) {
        showToast('星星收到了通知: ${notification.text}');
        return true;
      },
      child: Scaffold(
          appBar: AppBar(),
          body: NotificationListener<TextNotification>(
            onNotification: (TextNotification notification) {
              showToast('大宝收到了通知: ${notification.text}');
              // 如果这里改成 true, 星星就收不到信息了,
              return false;
            },
            child: Center(
              child: Builder(
                builder: (BuildContext context) {
                  return RaisedButton(
                    onPressed: () {
                      TextNotification('下班了!')..dispatch(context);
                    },
                    child: Text('点我'),
                  );
                },
              ),
            ),
          )),
    );
  }
}

class TextNotification extends Notification {
  TextNotification(this.text);
  final String text;
}
```

而我们经常使用的下拉刷新和上拉加载更多的组件也可以通过监听 `ScrollNotification` 来完成。

pull_to_refresh_notification | Flutter Package (flutter-io.cn)

loading_more_list | Flutter Package (flutter-io.cn)

» ScrollPosition.ensureVisible

要完成这个操作，应该大部分人都是会的。其实万变不离其中，通过当前对象的 `RenderObject` 去找到对应的 `RenderAbstractViewport`，然后通过 `getOffsetToReveal` 方法获取相对位置。

```dart 复制代码
/// Animates the position such that the given object is as visible as possible
/// by just scrolling this position.
///
/// See also:
///
///  * [ScrollPositionAlignmentPolicy] for the way in which `alignment` is
///    applied, and the way the given `object` is aligned.
Future<void> ensureVisible(
  RenderObject object, {
  double alignment = 0.0,
  Duration duration = Duration.zero,
  Curve curve = Curves.ease,
  ScrollPositionAlignmentPolicy alignmentPolicy = ScrollPositionAlignmentPolicy.explicit,
}) {
  assert(alignmentPolicy != null);
  assert(object.attached);
  final RenderAbstractViewport viewport = RenderAbstractViewport.of(object);
  assert(viewport != null);

  double target;
  switch (alignmentPolicy) {
    case ScrollPositionAlignmentPolicy.explicit:
      target = viewport.getOffsetToReveal(object, alignment).offset.clamp(minScrollExtent, maxScrollExte
      break;
    case ScrollPositionAlignmentPolicy.keepVisibleAtEnd:
      target = viewport.getOffsetToReveal(object, 1.0).offset.clamp(minScrollExtent, maxScrollExtent) as
      if (target < pixels) {
        target = pixels;
      }
      break;
    case ScrollPositionAlignmentPolicy.keepVisibleAtStart:
      target = viewport.getOffsetToReveal(object, 0.0).offset.clamp(minScrollExtent, maxScrollExtent) as
      if (target > pixels) {
        target = pixels;
      }
      break;
  }

  if (target == pixels)
    return Future<void>.value();

  if (duration == Duration.zero) {
    jumpTo(target);
    return Future<void>.value();
  }

  return animateTo(target, duration: duration, curve: curve);
}
```

Demo 代码地址: ensureVisible 演示 (github.com)

留个问题，当你点击 `点我跳转顶部，我是固定的` 这个按钮的时候，你猜会发生什么现象。

「Flutter 挑战」

之前跟掘金官方提过，是否可以增加 `你问我答 / 你出题我挑战` 模块，增加程序员之间的交流，程序员都是不服输的，应该会 吧？想想都刺激。我创建一个新的 FlutterChallenges qq 群 321954965 来进行交流；仓库，用来讨论和存放这些小挑战代码。平时收集一些平时有一些难度的实际场景例子，不单单只是秀技术。进群需要通过推荐或者验证，欢迎喜欢折腾自己的童鞋 。

情人节 + 七夕 这是不是个巧合 ？？

» 美团饿了么点餐页面

要求:

1. 左右2个列表能联动，整个首页上下滚动联动
2. 通用性，可成组件

如果你认真看完了 `NestedScrollView`，我想应该有办法来做这种功能了。

» 增大点击区域

增加点击区域，这应该是平时应该会遇到的需求，那么在 Flutter 中应该怎么实现呢？

原始代码地址: 增大点击区域 (github.com)

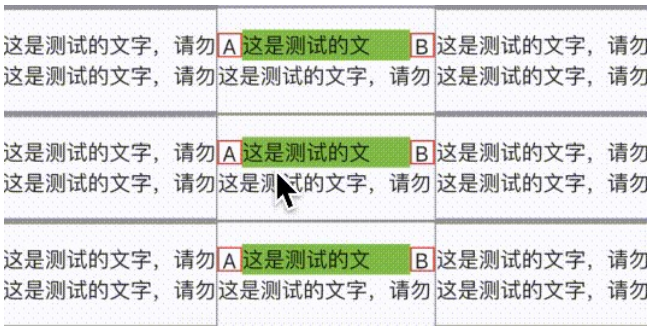
为了测试方便，请添加在 `pubspec.yaml` 中 添加财经龙大佬的 `oktoast` 。

```yaml
oktoast: any
```

要求:

1. 不要改变整个结构和尺寸。
2. 不要直接 `Stack` 把整个 `Item` 重写。
3. 通用性。

完成效果如下，扩大的范围理论上可以随意设置。



「结语」

第一次把掘金干爆，只能将文章一部分代码都移到了 gist.github.com/zmtzawqlp (突然想起来一些说什么万字文章的标题党是不是有点打脸呀，我看接近 9000 就不能再写了)。这篇写的比较多，想到了什么就写。不管是什么技术，只有深入了才能领会其中的道理。维护开源组件，确实是一件很累的事情。但是这会不断强迫你去学习，在不停更新迭代当中，你都会学习到一些平时不容易接触到的知识。积沙成塔，撸遍 `Flutter` 源码不再是梦想。

爱 `Flutter`，爱 `糖果`，欢迎加入Flutter Candies，一起生产可爱的Flutter小糖果 加入QQ群 QQ群:181398081

最最后放上 Flutter Candies 全家桶，真香。