

# ListView类



## ListView

一个可滚动的列表

线性排列的可滚动列表小部件。

[列表视图](#)是最常用的滚动窗口小部件。它显示 孩子们一个接一个滚动的方向。 十字轴的 孩子们需要填补[列表视图](#)。

如果非空, [itemExtent](#)迫使孩子们已经给定的程度 滚动的方向。 指定一个[itemExtent](#)是更有效的比让孩子决定自己的程度, 因为滚动 预知的机器可以利用孩子们的程度上保存 工作, 例如, 当滚动位置变化显著。

为构建一个有三个选项[列表视图](#):

1. 默认构造函数需要一个显式的[列表](#)的孩子。 这 构造函数适合与少量的列表视图 孩子们因为建设[列表](#)需要为每个工作 孩子可能被显示在列表视图而不是 这些孩子实际上是可见的。
2. 的[ListView.builder](#)以一个[IndexedWidgetBuilder](#),构建 孩子的需求。 这个构造函数适合列表视图 大(或无限)的儿童数量,因为建筑商 只有对那些孩子,实际上是可见的。
3. 的[ListView.custom](#)需要一个[SliverChildDelegate](#),它提供了 子模型的定制其他方面的能力。 例如, 一个[SliverChildDelegate](#)可以控制算法用于估计 孩子没有可见的大小。

控制的初始滚动抵消滚动视图, 提供一个[控制器](#)与它的[ScrollController.initialScrollOffset](#)属性集。

## 示例代码

无限的孩子们:

```
new ListView.builder(  
  padding: new EdgeInsets.all(8.0),  
  itemExtent: 20.0,  
  itemBuilder: (BuildContext context, int index) {  
    return new Text('entry $index');  
  },  
)
```

## 过渡到CustomScrollView

一个[列表视图](#)基本上是一个[CustomScrollView](#)用一个[SliverList](#)在它的[CustomScrollView.slivers](#)财产。

如果[列表视图](#)不再是足够的,例如因为滚动视图是一个列表和一个网格,或因为列表组合与一个[SliverAppBar](#)等,是直接港口代码使用[列表视图](#)使用[CustomScrollView](#)直接。

的[关键](#), [scrollDirection](#), [反向](#), [控制器](#), [主](#), [物理](#), 和[收缩包装](#)上的属性[列表视图](#)直接映射到相同命名属性[CustomScrollView](#)。

的[CustomScrollView.slivers](#)房地产应该包含一个列表[SliverList](#)或者一个[SliverFixedExtentList](#); 前如果[itemExtent](#)在[列表视图](#)是零,后者如果[itemExtent](#)没有空。

的[childrenDelegate](#)属性[列表视图](#)对应于[SliverList.delegate](#)(或[SliverFixedExtentList.delegate](#))的财产。 的[新列表视图](#)构造函数的children参数对应[childrenDelegate](#)作为一个

[SliverChildListDelegate](#)与同样的论点。 的[新ListView.builder](#)构造函数的itemBuilder和childCount参数对应[childrenDelegate](#)作为一个[SliverChildBuilderDelegate](#)的匹配参数。

的[填充](#)属性对应于有一个[SliverPadding](#)在[CustomScrollView.slivers](#)属性列表本身,而是和拥有的[SliverList](#)一个孩子的[SliverPadding](#)。

默认情况下, [列表视图](#)将自动垫列表的滚动吗四肢,以避免局部障碍物所示[MediaQuery](#)的填充。 为了避免这种行为,覆盖以零[填充](#)财产。

一旦代码被移植到使用[CustomScrollView](#),等其他裂片[SliverGrid](#)或[SliverAppBar](#),可以放的[CustomScrollView.slivers](#)列表。

### 示例代码

这里有两个简短的代码片段显示[列表视图](#)及其等价的使用[CustomScrollView](#):

```
new ListView(  
  shrinkWrap: true,  
  padding: const EdgeInsets.all(20.0),  
  children: <Widget>[  
    const Text('I\'m dedicating every day to you'),  
    const Text('Domestic life was never quite my style'),  
    const Text('When you smile, you knock me out, I fall apart'),  
    const Text('And I thought I was so smart'),  
  ],  
)  
  
new CustomScrollView(  
  shrinkWrap: true,  
  slivers: <Widget>[  
    new SliverPadding(  
      padding: const EdgeInsets.all(20.0),  
      sliver: new SliverList(  
        delegate: new SliverChildListDelegate(  
          <Widget>[
```

```

const Text('I\'m dedicating every day to you'),
const Text('Domestic life was never quite my style'),
const Text('When you smile, you knock me out, I fall apart'),
const Text('And I thought I was so smart'),
],
),
),
),
],
)

```

参见:

- [SingleChildScrollView](#),这是一个可滚动的小部件 的孩子。
- [页面浏览人数](#),这是一个滚动的列表小部件,每一个孩子 窗口的大小。
- [显示数据表格](#)可滚动,二维数组的小部件。
- [CustomScrollView](#),这是一个可滚动的小部件创建自定义 使用裂片滚动效果。
- [ListBody](#)以类似的方式,安排孩子,但是没有 滚动。
- [ScrollNotification](#)和[NotificationListener](#),可以用来观察 没有使用一个滚动的位置 [ScrollController](#)。

继承

- [对象](#)

- [Diagnosticable](#)

- [DiagnosticableTree](#)

- [小部件](#)

- [StatelessWidget](#)

- [滚动视图](#)

- [BoxScrollView](#)

- 列表视图

## 构造函数

[列表视图](#) ({[关键](#) 关键, [轴](#) scrollDirection:Axis.vertical, [bool](#) 反向:

假, [ScrollController](#) 控制器, [bool](#) 主, [ScrollPhysics](#) 物理, [bool](#) 收缩包装:

假, [EdgeInsetsGeometry](#) 填充, [双](#) itemExtent, [bool](#) addAutomaticKeepAlives:真正的, [bool](#) addRepaintBoundaries:真正的, [列表<小部件>](#) 孩子们:常量[]})

创建了一个可滚动的, 小部件从一个显式的线性数组[列表](#)。[...]

[ListView.builder](#) ({[关键](#) 关键, [轴](#) scrollDirection:Axis.vertical, [bool](#) 反向:

假, [ScrollController](#) 控制器, [bool](#) 主, [ScrollPhysics](#) 物理, [bool](#) 收缩包装:

假, [EdgeInsetsGeometry](#) 填充, [双](#) itemExtent, @

required [IndexedWidgetBuilder](#) itemBuilder, [int](#) itemCount, [bool](#) addAutomaticKeepAlives:真正的, [bool](#) addRepaintBoundaries:真正的})

创建了一个可滚动, 线性阵列上创建的小部件的需求。[...]

[ListView.custom](#) ({[关键](#) 关键, [轴](#) scrollDirection:Axis.vertical, [bool](#) 反向:

假, [ScrollController](#) 控制器, [bool](#) 主, [ScrollPhysics](#) 物理, [bool](#) 收缩包装:

假, [EdgeInsetsGeometry](#) 填充, [双](#) itemExtent, @

required [SliverChildDelegate](#) childrenDelegate})

创建了一个可滚动, 线性阵列的小部件和一个定制的子模型。[...]

## 属性

[childrenDelegate](#) → [SliverChildDelegate](#)

一个委托, 提供孩子们的ListView。[...]

最后

[itemExtent](#) → [双](#)

如果非空, 迫使孩子们有给定的程度上滚动 方向。[...]

最后

[控制器](#) → [ScrollController](#)

一个对象, 可以用来控制这个卷轴的位置 视图是滚动。[...]

最后, 继承了

[hashCode](#) → [int](#)

这个对象的哈希码。[...]

只读的, 遗传的

[关键](#) → [关键](#)

控制一个小部件替换另一个小部件在树上。[...]

最后, 继承了

[填充](#) → [EdgeInsetsGeometry](#)

空间的插图的儿童。

最后, 继承了

[物理](#) → [ScrollPhysics](#)

滚动视图应该如何响应用户输入。 [\[...\]](#)

最后, 继承了

[主](#) → [bool](#)

是否这是主要的滚动视图与父母有关PrimaryScrollController。 [\[...\]](#)

最后, 继承了

[反向](#) → [bool](#)

是否阅读方向滚动视图卷轴。 [\[...\]](#)

最后, 继承了

[runtimeType](#) → [类型](#)

一个对象的运行时类型的代表。

只读的, 遗传的

[scrollDirection](#) → [轴](#)

轴滚动视图的卷轴。 [\[...\]](#)

最后, 继承了

[收缩包装](#) → [bool](#)

是否滚动视图的程度scrollDirection应该是 决定的内容正在查看。 [\[...\]](#)

最后, 继承了

## 方法

[buildChildLayout](#) ([BuildContext](#) 上下文) → [小部件](#)

子类应该重写此方法构建布局模型。

[debugFillProperties](#) ([DiagnosticPropertiesBuilder](#) 属性) → 无效

[构建](#) ([BuildContext](#) 上下文) → [小部件](#)

描述了由这个小部件的用户界面的一部分。 [\[...\]](#)

继承了

[buildSlivers](#) ([BuildContext](#) 上下文) → [列表<小部件>](#)

建立部件放置在窗口的列表。 [\[...\]](#)

继承了

[buildViewport](#) ([BuildContext](#) 上下文, [ViewportOffset](#) 抵

消, [AxisDirection](#) axisDirection, [列表<小部件>](#) 裂成小片) → [小部件](#)

构建视窗。 [\[...\]](#)

@protected, 继承了

[createElement](#)() → [StatelessElement](#)

创建一个[StatelessElement](#)在树上来管理这个小部件的位置。 [\[...\]](#)

继承了

[debugDescribeChildren\(\)](#) → [列表<DiagnosticsNode>](#)

返回一个列表[DiagnosticsNode](#)描述该节点的对象 的孩子。 [\[...\]](#)

*@protected, 继承了*

[getDirection](#)([BuildContext](#) 上下文) → [AxisDirection](#)

返回[AxisDirection](#)滚动视图的卷轴。 [\[...\]](#)

*@protected, 继承了*

[noSuchMethod](#)([调用](#) 调用) → [动态](#)

当用户访问一个不存在的方法或属性调用。 [\[...\]](#)

*继承了*

[toDiagnosticsNode](#)([{字符串 的名字, DiagnosticsTreeStyle 风格}](#)) → [DiagnosticsNode](#)

返回一个对象被调试的调试表示 工具和[toStringDeep](#)。 [\[...\]](#)

*继承了*

[toString](#)([{DiagnosticLevel minLevel:DiagnosticLevel.debug}](#)) → [字符串](#)

返回该对象的字符串表示。

*继承了*

[toStringDeep](#)([{字符串 prefixLineOne:”, 字符](#)

[串 prefixOtherLines, DiagnosticLevel minLevel:DiagnosticLevel.debug}](#)) → [字符串](#)

返回一个字符串表示该节点及其后代。 [\[...\]](#)

*继承了*

[toStringShallow](#)([{字符串 乔伊](#)

[纳:”、“, DiagnosticLevel minLevel:DiagnosticLevel.debug}](#)) → [字符串](#)

返回一行详细描述的对象。 [\[...\]](#)

*继承了*

[toStringShort](#)() → [字符串](#)

短, 这个小部件的文本描述。

*继承了*

## 运营商

[运算符==](#)([动态 其他](#)) → [bool](#)

相等操作符。 [\[...\]](#)

*继承了*