

AppBar类



AppBar

一个Material Design应用程序栏，由工具栏和其他可能的widget（如TabBar和FlexibleSpaceBar）组成。

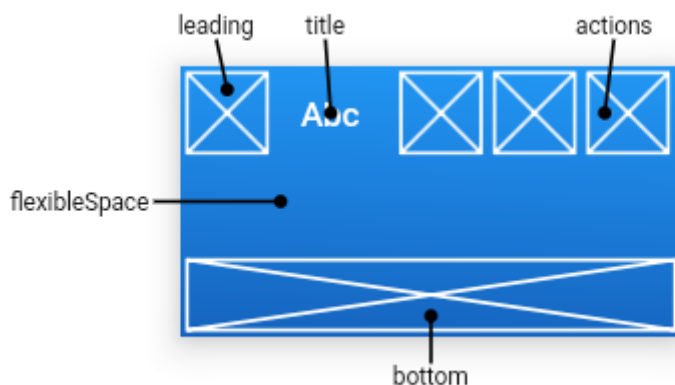
材料设计应用程序栏。

应用程序栏包含一个工具栏和潜在的其他部件，如[TabBar](#)和一个[FlexibleSpaceBar](#)。

应用酒吧通常会公开一个或多个 常见的[行动](#)与[IconButton](#) (可选), 后跟一个年代 [PopupMenuButton](#)不太常见的操作 (有时称为“溢出 菜单”)。

应用酒吧通常使用的[Scaffold.appBar](#)财产, 哪些地方 应用程序栏作为一个fixed-height部件在屏幕的顶部。 对于一个 可滚动应用酒吧, 看到的[SliverAppBar](#)嵌入一个[AppBar](#)在一片 中使用的[CustomScrollView](#)。

AppBar显示工具栏小部件, [领先的](#), [标题](#), [行动](#), 以上[底](#) (如果有的话)。 的[底](#)通常是用于[TabBar](#)。 如果 一个[flexibleSpace](#)指定小部件然后堆放在工具栏的后面 和底部的小部件。 下面的图表显示了每一个插槽 出现在工具栏从左到右书写语言时 (如。 英文):



The leading widget is in the top left, the actions are in the top right, the title is between them. The bottom is, naturally, at the bottom, and the `flexibleSpace` is behind all of them.

如果[领先的](#)小部件是省略了, 但是[AppBar](#)是在一个[脚手架](#)与 一个[抽屉](#)里, 然后将插入一个按钮打开抽屉。 否则, 如果 最近的[导航器](#)有任何以前的路线, [BackButton](#)插入代替。 这种行为可以通过设置[automaticallyImplyLeading](#)为假。 在这种情况下下一个空主要部件将中间结果/标题小部件 伸展运动开始。

示例代码

```
new AppBar(  
  title: new Text('My Fancy Dress'),  
  actions: <Widget>[  
    new IconButton(  
      icon: new Icon(Icons.playlist_play),  
      tooltip: 'Air it',  
      onPressed: _airDress,  
    ),  
    new IconButton(  
      icon: new Icon(Icons.playlist_add),  
      tooltip: 'Restitch it',  
      onPressed: _restitchDress,  
    ),  
    new IconButton(  
      icon: new Icon(Icons.playlist_add_check),  
      tooltip: 'Repair it',  
      onPressed: _repairDress,  
    ),  
  ],  
)
```

参见:

- [脚手架](#),这显示了[AppBar](#)在其[Scaffold.appBar](#)槽。
- [SliverAppBar](#),它使用[AppBar](#)提供一个灵活的应用程序栏 可以用在一个[CustomScrollView](#)。
- [TabBar](#)通常,这是放置在[底](#)槽的[AppBar](#)如果屏幕有多个页面安排在选项卡。

- [IconButton](#),这是使用[行动](#)显示应用程序栏上的按钮。
- [PopupMenuButton](#)应用程序栏上的,显示一个弹出菜单,通过[行动](#)。
- [FlexibleSpaceBar](#),这是使用[flexibleSpace](#)当应用程序栏 可以展开和折叠。
- [material.google.com/layout/structure.html # structure-toolbars](https://material.google.com/layout/structure.html#structure-toolbars)

继承

- [对象](#)

- [Diagnosticable](#)

- [DiagnosticableTree](#)

- [小部件](#)

- [StatefulWidget](#)

- [AppBar](#)

实现了

- [PreferredSizeWidget](#)

构造函数

[AppBar](#)({[关键](#) 关键, [小部件](#) 领先的, [bool](#) automaticallyImplyLeading:真正的, [小部件](#) 标题, [列表<小部件>](#) 行动, [小部件](#) flexibleSpace, [PreferredSizeWidget](#) 底, [双](#) 海拔高度:4.0, [颜色](#) 写成

backgroundColor, [亮度](#) 亮度, [IconThemeData](#) iconTheme, [TextTheme](#) textTheme, [bool](#) 主:真正的, [bool](#) centerTitle, [双](#) titleSpacing:NavigationToolbar.kMiddleSpacing, [双](#) toolbarOpacity:1.0, [双](#) bottomOpacity:1.0}))
创建一个材料设计应用程序栏。[\[...\]](#)

属性

[行动](#) → [列表](#)<[小部件](#)>

小部件显示后title小部件。[\[...\]](#)

最后

[automaticallyImPLYLeading](#) → [bool](#)

控制我们是否应该试图暗示如果零的主要部件。[\[...\]](#)

最后

[写成backgroundColor](#) → [颜色](#)

应用程序栏的颜色使用的材料。 通常这应该被设置 随着
brightness, iconTheme, textTheme。[\[...\]](#)

最后

[底](#) → [PreferredSizeWidget](#)

这个小部件出现在应用程序栏的底部。[\[...\]](#)

最后

[bottomOpacity](#) → [双](#)

如何不透明部分的应用程序栏是底部。[\[...\]](#)

最后

[亮度](#) → [亮度](#)

应用程序栏的亮度的材料。 通常这是组 与
backgroundColor, iconTheme, textTheme。[\[...\]](#)

最后

[centerTitle](#) → [bool](#)

标题是否应该为中心。[\[...\]](#)

最后

[海拔高度](#) → [双](#)

把这个程序的z坐标酒吧。 这个控件的大小 应用程序栏下面的阴影。[\[...\]](#)

最后

[flexibleSpace](#) → [小部件](#)

这个小部件是工具栏和tabbar背后堆叠。 它的高度 同应用程序栏的整体高度。 [\[...\]](#)
最后

[iconTheme](#) → [IconThemeData](#)

颜色、不透明度和用于应用程序栏图标大小。 这通常 设置与
backgroundColor, brightness, textTheme。 [\[...\]](#)

最后

[领先的](#) → [小部件](#)

一个小部件显示之前title。 [\[...\]](#)

最后

[preferredSize](#) → [大小](#)

一个高度之和的大小[kToolbarHeight](#)和[底](#)小部件的 首选的高度。 [\[...\]](#)

最后

[主](#) → [bool](#)

这个应用程序栏是否显示在屏幕的顶部。 [\[...\]](#)

最后

[textTheme](#) → [TextTheme](#)

文本的排版样式使用在应用程序栏中。 通常这是 组一起
brightness backgroundColor, iconTheme。 [\[...\]](#)

最后

[标题](#) → [小部件](#)

主要的小部件中显示appbar。 [\[...\]](#)

最后

[titleSpacing](#) → [双](#)

周围的间距title水平轴上的内容。 这个间距 即使没有应用leading内容或
actions。 如果你想要title采取一切可用的空间, 将这个值设置为0.0。 [\[...\]](#)

最后

[toolbarOpacity](#) → [双](#)

如何不透明工具栏应用程序栏的一部分。 [\[...\]](#)

最后

[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

[runtimeType](#) → [类型](#)

一个对象的运行时类型的代表。

只读的, 遗传的

方法

[createState](#)() → [_AppBarState](#)

为这个小部件创建可变状态给定树中的位置。 [\[...\]](#)

[createElement](#)() → [StatefulElement](#)

创建一个[StatefulElement](#)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

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

添加额外的属性与节点相关联。 [\[...\]](#)

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

操作

[运算符==](#) (动态 其他) → [bool](#)

相等操作符。 [\[...\]](#)

继承了