

# Text类

与单一风格的文本。

的[文本](#)小部件显示一个文本字符串,与单一的风格。 的字符串 可能打破跨多行或可能被显示在同一行根据布局的约束。

的[风格](#)参数是可选的。 省略时,将使用的文字风格 从最近的封闭[DefaultTextStyle](#)。 如果给定的样式 [TextStyle.inherit](#)财产是真的,给定的样式将合并 最接近的封闭[DefaultTextStyle](#)。 这种合并行为是有用的,例如,为了使文本在使用默认字体和大胆 大小。

使用new [TextSpan.rich](#)构造函数, [文本](#)小部件也可以 创建一个[TextSpan](#)显示文本,使用多个样式 (例如,一段有一些大胆的字)。

## 示例代码

```
new Text(  
  'Hello, $_name! How are you?',  
  textAlign: TextAlign.center,  
  overflow: TextOverflow.ellipsis,  
  style: new TextStyle(fontWeight: FontWeight.bold),  
)
```

## 交互性

为了使[文本](#)对触摸事件做出反应,包装它[GestureDetector](#)小部件 与一个[GestureDetector.onTap](#)处理程序。

在材料设计应用程序中,考虑使用[FlatButton](#)相反,或 如果不合适,至少使用一个[墨水池](#)而不是[GestureDetector](#)。

部分文本的互动,利用[RichText](#)并指定一个[TapGestureRecognizer](#)随着[TextSpan.recognizer](#)的相关部分文本。

参见:

- [RichText](#),它给你更多的控制文本样式。
- [DefaultTextStyle](#)设置默认样式[文本](#)小部件。

继承

- [对象](#)

- [Diagnosticable](#)

- [DiagnosticableTree](#)

- [小部件](#)

- [StatelessWidget](#)

- 文本

## 构造函数

[文本](#)([字符串](#) 数据, { [关键](#) 关键, [TextStyle](#) 风格, [TextAlign](#) textAlign, [TextDirection](#) textDirection, [bool](#) softWrap, [TextOverflow](#) 溢出, [双](#) textScaleFactor, [int](#) maxLines})

创建一个文本小部件。[...]

常量

[Text.rich](#)([TextSpan](#) textSpan, { [关键](#) 关键, [TextStyle](#) 风格, [TextAlign](#) textAlign, [TextDirection](#) textDirection, [bool](#) softWrap, [TextOverflow](#) 溢出, [双](#) textScaleFactor, [int](#) maxLines})

创建一个文本小部件[TextSpan](#)。

常量

## 属性

[数据](#) → [字符串](#)

要显示的文本。[...]

最后

[maxLines](#) → [int](#)

一个可选的最大跨度的文本的行数, 如果需要包装。 如果文本超过给定的行数, 表示它将被截断 来 overflow。[...]

最后

[溢出](#) → [TextOverflow](#)

应该如何处理视觉溢出。

最后

[softWrap](#) → [bool](#)

是否应该打破软换行符的文本。[...]

最后

[风格](#) → [TextStyle](#)

如果非空, 使用这个文本风格。[...]

最后

[textAlign](#) → [TextAlign](#)

水平应该如何对齐文本。

最后

[textDirection](#) → [TextDirection](#)

文本的方向性。[...]

最后

[textScaleFactor](#) → [双](#)

字体为每个逻辑像素像素的数量。[...]

最后

[textSpan](#) → [TextSpan](#)

文本显示作为一个TextSpan。[...]

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

[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。[...]

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

[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

[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