第 3 章

用 户 输 入

用户输入是大部分应用程序必不可少的环节，本章将介绍 2 种最基本的用户输入类型，分别是文本框和按钮。更复杂的用户操作，如滑动手势和拖放等，可参考第 8 章“人机交互”。

3.1 文本框

文本框可以让用户在应用程序中输入文本内容，例如搜索栏、聊天窗口的输入区域、表单中需要填写的快递信息或者登录界面的用户名和密码等地方都能找到文本框的身影。在 Flutter 框架中，最常见的文本框有 TextField 和 CupertinoTextField 这 2 个组件，分别对应安卓的 Material 风格和 iOS 的 Cupertino 风格。

3.1.1 TextField

TextField 组件是 Flutter 中最常用的文本框组件。它的基本用法非常简单，同时又支持各式各样的自定义。该组件虽然没有任何必传参数，但实战中经常需要传入 onSubmitted 参数，用于在用户完成输入时处理业务逻辑。例如，可将用户输入的内容打印至终端，代码如下：

```
TextField(
  onSubmitted: (value) => 
    print("submitted: $value"),
)
```

运行效果如图 3-1 所示。

图 3-1 默认样式的文本框

TextField 共有 5 种状态：无焦点、无焦点且错误、有焦点、有焦点且错误、禁用。通常有焦点时 TextField 需要呈现出与众不同的样式，帮助用户辨认输入的文字将被录入哪个文本框。禁用的文本框(enabled 属性为 false 时)无法被用户选中，因此不可能有焦点。另外，这里的错误不是指程序运行时抛出异常或报错，而是指 TextField 可以呈现出一种错误

的状态，主要用于提示用户可能输入有误，例如电子邮箱不符合格式等。

在默认情况下，TextField 组件遵循 Material 界面风格，自带一条下画线。当页面焦点不在文本框时下画线呈灰色，一旦获得页面焦点，下画线会自动变成程序当前主题风格的主颜色（如蓝色），并出现光标闪烁。若该设备没有物理键盘，则屏幕软键盘会同时弹出。当需要提示输入有误时，下画线则会变成红色。

1. InputDecoration

TextField 组件的大部分外观修饰可通过向 decoration 参数传入 InputDecoration 实现。而 InputDecoration 本身包罗万象，可以通过其构造函数传入大量参数，以便设置前缀、后缀、提示信息、边框、填充色及错误状态等需要显示的文本及样式。

1）前缀

InputDecoration 有 3 类前缀，如果同时设置则会依次显示，互不冲突。第 1 类是可用 icon 属性设置，在 TextField 前面插入一个图标。第 2 类是由 prefixIcon（前缀图标）和 prefixIconConstraints（前缀图标的布局约束）属性设置，用于定义显示在 icon 之后的前缀图标。第 3 类是 prefix（前缀组件）、prefixText（前缀文本）和 prefixStyle（前缀样式）属性设置，用于定义显示在 icon 和前缀图标之后的前缀文本。这里为了演示，同时使用上述 3 类前缀，代码如下：

```
TextField(
  decoration: InputDecoration(
    icon: Icon(Icons.add),                    //图标(加号)
    prefixIcon: Icon(Icons.lock),             //前缀图标(小锁)
    prefixText: "https://",                   //前缀文本内容
  ),
)
```

运行时，icon 属性设置的图标（加号）首先被显示出来，接着是 prefixIcon 属性设置的前缀图标（小锁）。这 2 个图标平时显示为灰色，并在文本框获得焦点时变成程序的主题色（如蓝色），并且在文本框获得焦点时，prefixText 中的内容也会被显示出来，运行效果如图 3-2 所示。

图 3-2 文本框前缀在有无焦点时的效果对比

上例代码中没有用到以下几个属性：首先 prefixIconConstraints 用于定义 prefixIcon 属性中图标的布局约束，例如规定最小宽度等。另外，prefix 属性和 prefixText 属性只能二选一，前者可以接收任何 Widget 类型，因此可以用于显示任意组件，而后者则直接接收 String 类型，故只支持文本。若选用 prefixText 使用文本，则可以用 prefixStyle 属性修改文本样式。

2）后缀

InputDecoration 还有 3 类设置后缀的属性。第 1 类是 counter、counterText、counterStyle

等属性，用于定义 TextField 的计数器。第 2 类是 suffixIcon(后缀图标)和 suffixIconConstraints(后缀图标的布局约束)属性，用于定义显示 TextField 末尾处的后缀图标。第 3 类是 suffix(后缀组件)、suffixText(后缀文本)和 suffixStyle(后缀样式)属性，用于定义后缀文本。例如，可同时设置这 3 类后缀，代码如下：

```
TextField(
  decoration: InputDecoration(
    counterText: "0/40",                    //计数器文本内容
    suffixIcon: Icon(Icons.visibility),     //后缀图标(眼睛)
    suffix: Icon(Icons.clear),              //后缀组件(清除图标)
  ),
)
```

运行时无论文本框是否有焦点，counterText 属性设置的计数器和 suffixIcon 属性设置的后缀图标都始终可见。当文本框获得焦点时 suffix 属性所设置的组件(清除图标)也会被渲染出来，运行效果如图 3-3 所示。

图 3-3 文本框后缀在有无焦点时的效果对比

上例代码中没有用到以下几个属性：首先是 counter 和 counterStyle，前者用于代替 counterText 传入任意组件，使其不再局限于显示文本，后者则用于定义 counterText 的文本样式。另外，suffix 和 suffixText 只能二选一。不同于前缀的例子，这里为了演示，故意选用了可以传入任意组件的版本，并传入了一个 Icon 组件。最后还有 suffixStyle 属性，可用于定义后缀文本的默认样式，以及可用于定义后缀图标布局约束的 suffixIconConstraints 属性。

3) 提示信息

Material 设计风格的文本框有 3 类提示信息，分别是 label(标签)、hint(暗示)和 helper(助手)，因此 InputDecoration 也有相应的 3 类属性，分别是：labelText、labelStyle 和 floatingLabelBehavior 属性，用于设置“标签”的文本、样式和是否自动漂浮。hintText、hintStyle 和 hintMaxLines 属性，用于设置“暗示”的文本、样式和行数，以及最后的 helperText、helperStyle 和 helperMaxLines 属性，用于设置“助手”的文本、样式和行数。这 3 类提示信息互不冲突，可以同时使用，代码如下：

```
TextField(
  decoration: InputDecoration(
    labelText: "Date of Birth",             //标签
    hintText: "yyyy-mm-dd",                 //暗示
    helperText: "Optional",                 //助手
  ),
)
```

运行时，首先 labelText 属性设置了标签，标明了该文本框的意图（Date of Birth，用于收集出生日期），并在文本框获得焦点后自动缩小并漂浮至左上角，不遮挡用户输入。此时 hintText 属性设置的文本开始出现，暗示格式要求（年-月-日），并将在用户输入任何字符后自动消失。最后，文本框的左下角始终显示 helperText，运行效果如图 3-4 所示。

图 3-4　文本框提示信息在有无焦点时的效果对比

上例代码中没有用到这 3 种提示信息对应的 style 属性或修改它们允许的最大行数，因此它们渲染的都是默认文本样式，包括字体大小和颜色等，且只占一行。另外，floatingLabelBehavior 属性可以用于定义标签（label）是否需要漂浮，默认为 FloatingLabelBehavior.auto，即自动漂浮，也可设置为 never（从不漂浮）或 always（总是漂浮）这些枚举值。

此外若文本框无焦点且支持多行，例如某个用于输入长篇内容的文本框共有 10 行，则默认情况下 labelText 会出现在垂直居中的位置，也就是第 5 行左右。若需让它漂浮在第 1 行的高度，则可通过传入 alignLabelWithHint：true 使 label 与 hint 对齐，即可实现这个效果。

4）边框

上文提到，TextField 组件共有 5 种状态，因此 InputDecoration 也有 5 个相应的定义边框样式的属性，分别为 enabledBorder（无焦点时的边框）、errorBorder（无焦点且错误）、focusedBorder（有焦点时的边框）、focusedErrorBorder（有焦点且错误）、disabledBorder（禁用时的边框）。另外它还提供简单的 border 属性，用于定义默认边框。

边框样式的选择一般有下画线（默认样式）、四周边框、粗细及颜色等，这里举例展示一些不同的边框样式，代码如下：

```
//第 3 章/text_field_decoration_border.dart

Column(
  mainAxisAlignment: MainAxisAlignment.spaceEvenly,
  children: [
    TextField(
      decoration: InputDecoration(
```

```
            border: UnderlineInputBorder(),
            helperText: "UnderlineInputBorder",
          ),
        ),
        TextField(
          decoration: InputDecoration(
            border: OutlineInputBorder(),
            helperText: "OutlineInputBorder",
          ),
        ),
        TextField(
          decoration: InputDecoration(
            border: InputBorder.none,
            helperText: "InputBorder.none",
          ),
        ),
        TextField(
          decoration: InputDecoration(
            enabledBorder: OutlineInputBorder(
              borderRadius: BorderRadius.circular(48),
              borderSide: BorderSide(
                width: 8.0,
                color: Colors.black,
              ),
            ),
            helperText: "width: 8.0, black",
          ),
        ),
      ],
    )
```

运行后一共得到4个TextField组件，从上到下分别为下画线、四周边框、无边框及自定义的黑色加粗圆边，效果如图3-5所示。

一般情况下通过border属性即可同时设置5种状态，如设置为四周边框后，TextField会继续保持原有的边框配色和粗细方案（例如有错误时显示为红色等），但若像上例中第4个TextField组件那样完全自定义颜色和粗细，则必须通过enabledBorder等共5个属性分别设置相应情况下的样式，否则其他情况依然会显示默认样式。

另外，TextField组件的默认修饰样式只有一条下画线作为边框。若不想有任何边框及其他修饰，则可以直接传入decoration：null，清空一切默认的样式。

5）错误状态

当用户输入的内容不符合格式要求或者有误时，开发者可以通过传入一个errorText（错误提示文字）字符串将TextField组件设置为错误状态。这样做时，组件修饰中原本helperText的位置会被替换为errorText的内容，并默认显示为红色。同时该组件的边框

图 3-5　文本框修饰边框的效果

会根据此刻是否有焦点，采用 errorBorder 或 focusedErrorBorder 二者之一，或在没有设置这些属性时将默认的蓝色下画线改为红色下画线。

例如，这里使用 Column 展示 3 个 TextField 组件，从上到下分别为无错误状态、有错误状态及设置了 errorBorder 的有错误状态，代码如下：

```
//第 3 章/text_field_decoration_error.dart

Column(
  mainAxisAlignment: MainAxisAlignment.spaceEvenly,
  children: [
```

```
      TextField(
        decoration: InputDecoration(
          errorText: null,                          //错误提示为空,即无错误状态
          helperText: "Helper Text",
        ),
      ),
      TextField(
        decoration: InputDecoration(
          errorText: "This field cannot be left blank",
          helperText: "Helper Text",
        ),
      ),
      TextField(
        decoration: InputDecoration(
          errorText: "This field cannot be left blank",
          errorBorder: OutlineInputBorder(
            borderSide: BorderSide(
              width: 8.0,
              color: Colors.red,
            ),
          ),
        ),
      ),
    ],
  )
```

运行效果如图 3-6 所示。

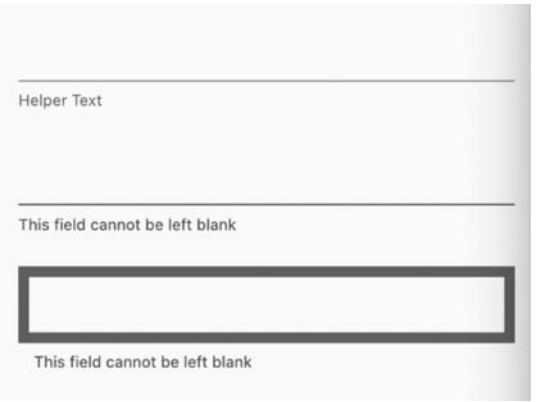

图 3-6　文本框后缀在有无焦点时的效果对比

另外,errorStyle 和 errorMaxLines 参数可分别设置错误提示文字的样式及最大行数,这与提示信息(如 helperText 等属性)的相应参数用法一致,在此不再赘述。

6) 填充色

InputDecoration 支持为文本框填充背景色。如有需求,则可先将 filled 属性设置为

true，启用填充色，再通过 fillColor 属性设置一种颜色。此外若程序运行在有鼠标的设备上，则开发者还可以再通过 hoverColor 属性传入另一种颜色，用于当用户将鼠标指针停留在 TextField 组件上时叠加渲染的另一层颜色。例如可通过 fillColor 传入黑色填充，再通过 hoverColor 传入半透明的白色，代码如下：

```
TextField(
  decoration: InputDecoration(
    filled: true,
    fillColor: Colors.black,
    hoverColor: Colors.white.withOpacity(0.5),
  ),
)
```

当程序运行在桌面计算机的浏览器上时，除了会显示填充的黑色外，还会在鼠标光标停留时额外在黑色的基础上叠加透明度为 50%的白色，最终混色出灰色，效果如图 3-7 所示。

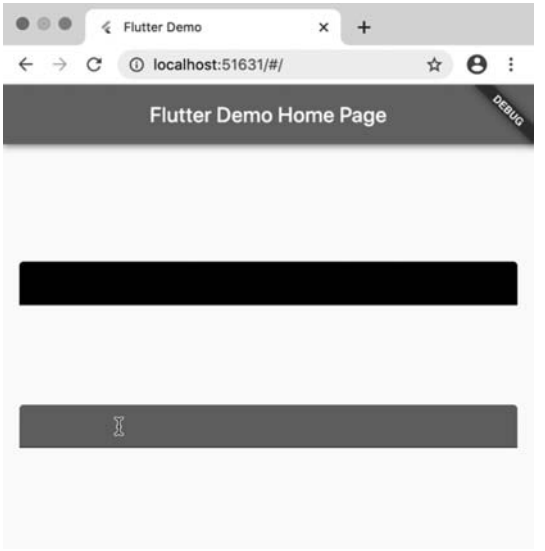

图 3-7　文本框填充色的效果

若程序运行在没有鼠标的设备（如手机）上，则会一直显示填充的黑色，而不会出现 hoverColor 的效果。另外，InputDecoration 原先还支持 focusColor 属性，用于在获得鼠标光标时添加额外混色，但因为不符合 Material 设计要求（该设计于 2020 年 2 月被移除[①]），因此该属性没有效果。

7）InputDecoration 的其他属性

除了上面详细讲解的这些属性外，InputDecoration 还有以下几个属性：contentPadding 用

① https://github.com/flutter/flutter/pull/33062

于设置边框与内容之间的留白；isDense 用于设置是否紧凑，紧凑的文本框会尽量节约垂直方向的空间；isCollapsed 用于设置是否折叠，折叠的文本框尺寸更小，且没有足够的空间显示 labelText 等内容。

此外，程序在运行时可随时通过 enabled 属性设置启用或禁用文本框。禁用时用户不可以选中该文本框，若禁用前已选中，则禁用时焦点会被强制移开，图标和边框等修饰物在默认情况下也会自动变成灰色，已经录入的内容会保留但不可再修改，直至再次启用。

2. TextField 样式

除了使用 InputDecoration 属性传入各式修饰之外，TextField 还有大量用于设置自身样式的属性，如文本样式、组件尺寸、允许的最大行数甚至光标颜色等都可以被自定义。

1）文本样式

TextField 组件中的文本样式主要通过 style 属性传入一个 TextStyle 类的值设置，这与 Text 组件中的同名属性用法相同，可自定义字体、字号、粗细、颜色、渲染特效等。同时 TextField 组件也与 Text 组件一样，还可用 textAlign 和 textDirection 参数设置文本对齐方式与阅读顺序。对此不熟悉的读者可以参考第 2 章“文字与图片”中关于 Text 组件的相关介绍和代码示例。

2）多行显示

较长的文本，如用于撰写邮件的正文部分或留言区评论等内容的输入框，一般需要 TextField 支持多行显示。这里可以通过 maxLines 属性开启多行显示并设置最大行数。该属性默认值不是 null，而是 1（最多为 1 行），此时 TextField 只会单行显示，且不允许用户换行。当传入其他值时，如设置 maxLines：5，允许最大 5 行，则表示 TextField 组件的尺寸（高度）最多可有 5 行文字那么高，但并不防止用户输入超过 5 行的内容。当内容较多时，单行显示的 TextField 支持水平（横向）滚动，而多行显示的则支持垂直（纵向）滚动。

界面设计时如有需要，还可以通过 minLines 设置最小行数，最终限制 TextField 的尺寸高度在最小行数与最大行数之间，随着内容长度的变化，限制在一定范围内变动。这里再次强调：maxLines 属性的默认值是 1，而不是 null。若无须限制最大行数，则可通过手动传入 maxLines：null 实现。

若需直接固定组件尺寸，则可选择传入 expands：true 将 TextField 强制拉伸至父级组件的尺寸，再通过在父级插入 Container 或 SizedBox 等组件设置尺寸约束，如 300×300 单位等。当启用 expands 时，minLines 和 maxLines 必须为 null。另外，当组件尺寸较大但文本内容较短时，也可通过 textAlignVertical 属性设置垂直对齐方式，如 TextAlignVertical.center 设置垂直居中等。

当文本内容超出多行显示的 TextField 的最大行数时，垂直滚动的具体行为可由 scrollController、scrollPadding 和 scrollPhysics 来定义。这与 ListView 等支持滚动的组件中的同名属性用法相似，不熟悉的读者可参考第 5 章“分页呈现”中关于 ListView 组件的介绍。

3）光标样式

TextField 在有焦点时会有闪烁的光标，如需隐藏光标，则可通过传入 showCursor：false 实现。

默认情况下光标为程序的主题色，如蓝色。若需设置光标的样式，则可以通过 cursorColor、cursorWidth 及 cursorRadius 这 3 个属性分别修改光标的颜色、粗细及圆角半径，代码如下：

```
TextField(
  cursorColor: Colors.black,
  cursorWidth: 8.0,
  cursorRadius: Radius.circular(4.0),
)
```

运行后可以得到一个较粗的黑色光标，且有圆角效果，如图 3-8 所示。实战中若光标设置得不够粗，则可能不易观察到圆角的效果。

图 3-8　自定义光标的效果

另外，TextField 组件还支持 mouseCursor 参数，用于在有鼠标的设备上定义鼠标指针悬浮在 TextField 时的鼠标光标，默认值为 SystemMouseCursors.text，即 I 字形鼠标光标。

4）密文

当需要用户输入密码时，开发者可通过 obscureText 属性决定是否将内容遮挡。在传入属性值 true 开启内容遮挡后，文本内容会被替换成 obscuringCharacter 属性中的字符，默认为“U+2022 BULLET”字符，即实心小圆圈。若需自定义遮挡字符，可在 obscuringCharacter 传入长度为 1 的字符串，如"*"等。

5）最大长度

开发者可通过 maxLength 属性设置 TextField 可允许的最长字符数，例如传入 maxLength：80 表示用户最多应输入 80 个字符。在设置最大长度后，TextField 的右下角会自动出现一个计数器，用于显示当前已输入的字符数和最大长度，渲染出如“5/80”的效果。

若无须限制最大长度却又想保留自动计数器的功能，则可以通过传入 maxLength：－1（或者传入 maxLength：TextField.noMaxLength 常量，增加可读性，其值为－1）实现，运行时 TextField 只会渲染用户已输入的字符数，如“5”，而没有分母。

反之，若需要限制最大长度但又需隐藏计数器，或者自定义计数器的样式，则可以通过 buildCounter 属性覆盖默认的计数器 build 方法。这个回传函数会将用户当前已输入的字符数量和所允许的最大字符数量等信息作为参数传给开发者，并期待得到一个 Widget 类

型的返回值以便用于最终渲染。默认情况下该 build 方法会返回一个 Text 组件，以达成默认时如“5/80”的效果，因此这里直接覆盖返回一个空的 Container 或 SizedBox 组件即可使计数器消失。

这里使用 Column 容器一并展示这 3 种计数器，代码如下：

```
//第 3 章/text_field_counter.dart

Column(
  children: [
    TextField(
      maxLength: 80,
    ),
    TextField(
      maxLength: TextField.noMaxLength,
    ),
    TextField(
      maxLength: 80,
      buildCounter: (
        BuildContext context, {
        required int currentLength,
        required int? maxLength,
        required bool isFocused,
      }) {
        return Text('$ {maxLength! - currentLength}');
      },
    ),
  ],
)
```

上述代码从上到下依次展示了 3 种计数器，分别为显示分子与分母的默认样式、不显示分母的无限长度样式，以及通过 buildCounter 自定义样式。自定义时，这里直接利用传入的 maxLength(最大长度)减去 currentLength(当前长度)实现“剩余多少个字符”的效果，如图 3-9 所示。

图 3-9　TextField 组件的计数器效果

有些读者也许记得前面介绍过的 decoration 属性的 InputDecoration 类中，也有 counter 和 counterText 参数可用于设置计数器。InputDecoration 中的计数器并不能自动计数，而这里的却可以。若两边都设置，InputDecoration 中的修饰属性则有优先权，覆盖这里的自动计数器。

最后，当用户输入的字符达到最大长度限制时，默认情况下就不可以再继续输入文字了。若需要允许用户超出长度后依然输入，则可通过传入 maxLengthEnforced：false，使 TextField 不强行停止用户输入。此时，默认的计数器会用红色显示一个假分数，如“81/80”，并将整个 TextField 设置为错误状态，即默认的红边框且加粗等。该错误状态的具体边框样式可由之前介绍过的 InputDecoration 中的 errorBorder 及 focusedErrorBorder 属性设置。

3. 内容过滤

如需检查或清洗用户输入的内容，例如在填写电话号码的输入框中不应该出现字母或汉字，或需要过滤敏感词，则可以借助 inputFormatter 属性直接完成，从而避免亲自编写代码。该属性不但支持一个 TextInputFormatter 类的列表，用于自定义允许或禁止的字符串，而且还支持直接传入正规表达式(RegExp)作为筛选条件。

实战中，最常使用的是 BlacklistingTextInputFormatter 和 WhitelistingTextInputFormatter 这 2 个继承于 TextInputFormatter 的类，分别对应“黑名单”和“白名单”。在 2020 年 8 月更新的 Flutter 版本 v1.20.0 中，为响应“计算机术语应尽量避免不必要的种族歧视”的号召，将这 2 类更改合并为 FilteringTextInputFormatter 类，并提供了 2 个构造函数，分别是 allow(允许)和 deny(禁止)。这 2 个构造函数都可以接收一个字符串或正规表达式作为筛选条件，并支持可选的 replacementString 参数，用于将筛选出的词语替换为任意其他词。由于 inputFormatters 属性接收的是列表类型，因此这 2 类也可以混搭使用。

这里为了演示，假设不允许用户输入任何元音字母，并将所有用户输入的"zzz"字符串替换为"Zzz..."，可以使用 2 个禁止列表完成，代码如下：

```
import 'package:flutter/services.dart';

//...

TextField(
  inputFormatters: [
    FilteringTextInputFormatter.deny(
      "zzz",
      replacementString: "Zzz...",
    ),
    FilteringTextInputFormatter.deny(
      RegExp(r"[aeiou]"),
      replacementString: "*",
    ),
  ],
)
```

上述代码首先定义不允许用户输入连续 3 个字母 z，即"zzz"字符串，若触发则替换为"Zzz..."文本，同时再通过正规表达式定义阻止条件为 aeiou 中的任意字符，并替换为一个星号。过滤前与过滤后的文本对比效果如图 3-10 所示。

4. 选择与交互菜单

1）选择框的高度

通常在安卓和 iOS 系统中，用户可以通过手指触摸选择文本框中的内容，然而 2018 年一位国外的开发者提出当 TextField 显示阿拉伯语时可能会出现不太完美的文本选择的视觉效果。笔者测试后遗憾地发现中英文混搭时也会发生类似情况，具体表现为高亮部分的高度不一致，如图 3-11 所示。

图 3-10　用 inputFormatter 参数实现内容过滤

图 3-11　中英文混搭时选择框的高度不一致

Flutter 框架于 2020 年 2 月[①]新增了 2 个属性用于方便开发者修复这个视觉效果，分别为 selectionHeightStyle 和 selectionWidthStyle。对于中英文混搭时选择框的高度不一致的情况，可以通过传入 selectionHeightStyle：BoxHeightStyle.max 解决。

2）交互菜单

在移动设备上，用户通常可以通过触摸手势呼叫出交互菜单，从而进行复制粘贴等操作。若需要禁止此类操作，则可以通过传入 enableInteractiveSelection：false 彻底关闭交互菜单。

如需关闭部分功能，则可以通过 toolbarOptions 参数单独设定每项菜单是否开启。默认情况下交互菜单共有 4 项，分别对应 ToolbarOptions 的 4 个参数：copy（复制）、cut（剪切）、paste（粘贴）和 selectAll（全选）。这并不表示默认情况下系统一定会弹出这 4 个选项。Flutter 会在程序运行时根据当前状态，智能弹出合理的菜单，例如剪切板为空时就不会出现粘贴菜单，又如文本框为密文时（通过 obscureText 属性设置）则不会出现剪切和复制菜单，以免用户将密码复制到别的明文框中查看内容。

以下代码展示 enableInteractiveSelection 和 toolbarOptions 属性的默认值，即全部开启：

```
TextField(
  enableInteractiveSelection: true,
  toolbarOptions: ToolbarOptions(
```

① https://github.com/flutter/flutter/pull/48917

```
      copy: true,
      cut: true,
      paste: true,
      selectAll: true,
    ),
  )
```

如需关闭其中的若干项，则只需将上述代码中对应的 true 改为 false。

5. 屏幕软键盘

1）键盘类型与样式

当设备没有物理键盘且用户需要输入文字时，TextField 会自动将系统的屏幕软键盘显示出来。这里开发者可以通过 keyboardType 设置合适的键盘类型，需要传入的值为 TextInputType 类，可用的值包括 number 数字、decimal 小数、multiline 多行、datetime 日期、emailAddress 电子邮箱、url 网址、name 姓名、phone 电话、streetAddress 地址等。例如，当传入 TextInputType.number 时，弹出的软键盘为数字九宫格的样式。

针对 iOS 设备还可以通过 keyboardAppearance 属性设置是否使用夜间模式的深色键盘。安卓系统允许用户对软键盘深度自定义，因此系统会忽略这里的设置，继续尊重用户在系统设置里选择的样式或自行安装的第三方键盘软件。

另外，开发者还可通过 textInputAction 属性设置键盘右下角的“回车键”需要改成什么键，值为 TextInputAction 类，如 send 发送、search 搜索、join 加入、go 访问、previous 上一个、next 下一个、done 结束等，但要注意并不是每个系统都支持所有值，例如 iOS 暂不支持 previous，安卓暂不支持 join 等。开发的过程中需要注意测试所有目标系统，确保键盘类型可用。

例如可同时使用上述 3 项功能，将软键盘设置为暗色（只对 iOS 有效），类型为输入电子邮件，并将右下角改为“搜索”键，代码如下：

```
TextField(
  keyboardAppearance: Brightness.dark,
  keyboardType: TextInputType.emailAddress,
  textInputAction: TextInputAction.search,
)
```

上述代码运行在 iOS 时可观察到暗色键盘，空格键右边有@按键，并且右下角显示 search 字样，如图 3-12(a)所示。当运行在安卓系统时，可观察到空格键左侧添加了@按键，并且右下角显示搜索图标，如图 3-12(b)所示。

2）自动纠错与补全

除了键盘样式外，TextField 组件还支持通过 autocorrect 属性设置是否启用自动纠错功能。当运行在安卓系统时，还额外支持 enableSuggestions 属性单独启用或禁用自动补全功能，但在 iOS 上自动补全只能随着自动纠错功能一并开启或者同时关闭。若开启自动补

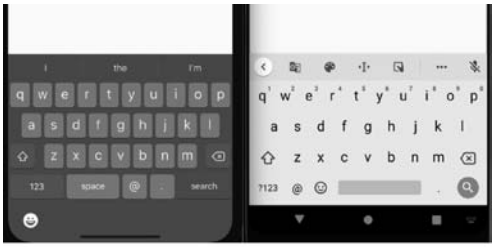

图 3-12　自定义键盘样式在 iOS 和安卓的效果对比

全，则安卓系统还可借助 autofillHints 传入一个任意数量的候选词列表。其他系统暂只支持一个候选词，因此 Flutter 会将候选列表的第一项转达给如 iOS 或网页浏览器。另外，在 iOS 11 及之后的版本上还有 smartQuotesType 和 smartDashesType 参数可供设置，分别决定是否自动将引号配对及是否将多个横行字符(---)转换为一条长长的横线。

3）自动大写字母

开发者可通过 textCapitalization 属性控制软键盘是否应自动处理大写字母。通常在使用拉丁字母的语言(如英语、法语、西班牙语等)中，每句话的开头及部分专有名词都需要使用大写字母。

默认情况下软键盘不会开启自动大写(但部分软键盘的自动纠错功能可能也会导致某些字母大写)。如需改变设置，则可以传入一个 TextCapitalization 类的值，例如设置 textCapitalization：TextCapitalization. words 可使每个单词的首字母均为大写。这里可供选择的值分别有 sentences(每句话开头)、words(每个单词开头)、characters(每个字母)及 none(无)。

最后值得一提的是，一般系统软键盘不会在弹出后突然改变样式或行为，所以这些属性值若发生变动，则通常需要等软键盘下次弹出时才会生效。

6. 事件

1）onSubmitted

当用户完成输入时，具体来讲是当用户单击屏幕软键盘上的“完成”等表示结束的按钮时，软键盘会被收起，同时 onSubmitted 事件会被触发，而用户输入的文字内容将作为该事件的参数传给开发者。

如需要在输入尚未完成的情况下实时监测文本框中内容的一举一动，则可使用 onChanged 事件。

2）onChanged

每当用户对 TextField 的内容做出改动，如添加或删除文字时，onChangd 事件都会被触发，并将最新的文本内容作为参数传入，但若用户没有直接修改 TextField，而是通过单击某个“清空”或“提交”按钮，从而导致文本内容发生变动的，则不会触发 onChanged 事件。这种情况下开发者应直接在相关按钮事件中处理业务逻辑，而不需要依赖 onChanged 事件。

如需监听更详细的用户行为，如光标位置的移动或选择区的变化等，则可以通过控制器的 TextEditingController.addListener 方法添加监听者实现（下文 7. 控制器中的选择区域会介绍具体用法）。

3）onEditingComplete

当用户完成输入时，onEditingComplete 事件会在 onSubmitted 事件之前触发。它的主要作用是处理与焦点相关的逻辑。

默认情况下，当用户单击软键盘上的 done、go、send、search 等表示结束的按键时（详见 textInputAction 属性），TextField 会自动放弃焦点并收起软键盘，但若软键盘的那个按键被修改成了 previous、next 或 join 等不表示结束的操作，TextField 则不会自动放弃焦点。开发者可通过设置 onEditingComplete 回传函数改变这一默认行为。

例如，在即时通信软件中，开发者通常会通过 textInputAction 属性将软键盘右下角修改为“发送”按键，但用户发送完一条短消息后很可能希望立即开始编辑下一条消息，因此不应该放弃焦点。此时可通过向 onEditingComplete 传入一个空函数来覆盖默认行为，代码如下：

```
TextField(
  onSubmitted: (value) => _send(value),
  onEditingComplete: () {},
  textInputAction: TextInputAction.send,
)
```

这样当用户单击软键盘上的“发送”按键后，TextField 依然会保持焦点。

4）onTap

每当用户单击一次 TextField 组件，它的 onTap 事件就会被触发一次，但双击操作的第 2 下单击不算是单独事件，因此不会触发，另外当 TextField 被禁用时（enabled 属性）该事件也不会触发。

由于 TextField 内部使用了 GestureDetector 监听用户手势（以便适时获取焦点、移动光标、弹出菜单等），因此它不适合再被嵌套 GestureDetector 组件，以免造成冲突，因此 TextField 在这里提供了 onTap 事件，方便有需要的开发者处理相关业务逻辑。对于 GestureDetector 组件不熟悉的读者可参考第 8 章“人机交互”中的相关内容。

7. 控制器

TextField 组件可以通过 controller 参数接收一个 TextEditingController 类的控制器。通过控制器，开发者可以更直接地控制 TextField 内的文本内容及选择区域的状态，例如设置文本初始值，或实现清空文本、移动光标、全选等功能。使用完毕后，需适时调用 dispose 方法释放资源。

1）初始化

控制器声明代码不建议放在 build 方法中，否则 Flutter 每次重绘都会重新初始化。实战中常见做法是在 State（组件状态类）中定义私有变量 _controller，并初始化为一个新的

TextEditingController控制器，以及最终在dispose方法中释放资源，完整代码如下：

```
//第3章/text_field_controller.dart

import 'package:flutter/material.dart';

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: MyHomePage(),
    );
  }
}

class MyHomePage extends StatefulWidget {
  @override
  _MyHomePageState createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
  TextEditingController _controller = TextEditingController();          //初始化

  @override
  void dispose() {
    _controller.dispose();                                              //释放资源
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text("TextEditingController Demo")),
      body: TextField(
        controller: _controller,                                        //在TextField组件
里使用控制器
      ),
    );
  }
}
```

声明并初始化控制器后，直接将其通过controller属性传给TextField组件即可。

2）文本内容

TextField组件的内容可通过与之关联的控制器的text属性随时访问并修改。例如，

以下代码先将当前文本的内容输出，再替换成 hello 字样：

```
print(_controller.text);
_controller.text = "hello";
```

如需清空文本内容，则可直接调用控制器的 clear 方法。

如需设置文本的初始内容，使页面刚加载时 TextField 中内容不为空，除了可以通过 text 属性修改外，还可以直接将初始内容传给 TextEditingController 的构造函数，代码如下：

```
TextEditingController(text: "这是 TextField 的初始内容")
```

3）选择区域

开发者可通过控制器的 selection 属性随时获取和修改 TextField 的文本选择情况。例如，某文本框中有 hello 字样，且被用户全选时，执行 print(_controller.selection)就会得到以下输出：

```
TextSelection(baseOffset: 0, extentOffset: 5,
affinity: TextAffinity.downstream, isDirectional: false)
```

这里可以观察到，输出内容中 baseOffset 为选择区域的起始位置，extentOffset 为选择区域的结束位置，两者之差就是选择的字符数量。若起始位置与结束位置一致，则表示当前没有选中任何文字，且光标正在该位置闪烁。如需修改选择区域，则应同样传入这 2 个值，但需注意选择区域的起始和结束位置不应超过文本的总长度，否则会发生运行时错误。另外，若只想移动光标而不想选择任何文字，则可以使用 collapsed 语法糖，少传一次重复的值。例如，将光标移动到第 0 个字符前，即 TextField 最起始的位置，以下 2 种写法都可以实现：

```
//两行代码效果一样，只需选择一行使用
_controller.selection = TextSelection(baseOffset: 0, extentOffset: 0);
_controller.selection = TextSelection.collapsed(offset: 0);
```

若需检查选择区域或光标位置是否合法，则可以使用控制器的 isSelectionWithinTextBounds 方法。例如，这里先判断合法性，若可行（当文字长度足够时）则选择从第 20 位至第 25 位共 5 个字符，代码如下：

```
final selection = TextSelection(baseOffset: 20, extentOffset: 25);
if (_controller.isSelectionWithinTextBounds(selection)) {        //判断合法性
  _controller.selection = selection;
}
```

4）事件监听

TextEditingController 还支持 addListener 方法，允许开发者添加回传函数，并在

TextField 组件的文字内容或选择区域发生变化时调用，代码如下：

```
_controller.addListener(() {
  print("现在的值: ${_controller.value}");
});
```

这里的_controller.value 属性包括了文字内容和选择区域，若只需获得其中一部分信息，则可直接通过其 text 或 selection 属性读取。

8. 其他属性

TextField 组件除了上述的大量属性外，还有以下几个比较容易理解的属性，这里简单概括。

如需禁用文本框，则可通过将 enabled 属性设置为 false 实现。这与之前介绍的 InputDecoration 中的 enabled 属性效果一致，若两者发生冲突则优先采用这里的属性。

如需使文本框进入“只读”模式，则可通过 readOnly 传入 true 实现。只读模式下文本框的内容不可被修改，包括不可以被交互菜单或键盘快捷键剪切或粘贴，但文字内容仍可被选中。

默认情况下，文本框不会自动获取焦点，即用户必须先单击某个文本框后才会弹出屏幕软键盘。若通过 autofocus：true 将页面上的某个 TextField 设置为自动获取焦点，则每当该组件出现时，若没有其他组件正在占用焦点，该文本框将自动获取焦点，并弹出软键盘。合理使用该属性可为用户免去一次单击。如需要在程序运行时更精确地控制焦点，则可由 focusNode 属性传入一个 FocusNode 实现。

3.1.2 CupertinoTextField

虽然 TextField 组件功能强大，但其 Material 设计风格偶尔会显得与 iOS 系统界面有些格格不入。实际上，Flutter 也自带了相应的 CupertinoTextField 组件，专门按照 iOS 的 Cupertino 风格设计。除了显示风格外，它的大部分功能与属性都和 TextField 组件保持高度一致，因此本节的重点在于介绍二者的差异。若读者对 TextField 组件还不熟悉，建议先阅读 3.1.1 节介绍的 TextField 的内容。

1. 装饰

虽然 CupertinoTextField 组件也有 decoration 属性，但实际上这里需要传入的值为 BoxDecoration 类，而不是 TextField 组件同名属性所支持的 InputDecoration 类。相比之下，BoxDecoration 缺少大量属性。其中一部分，如负责渲染前缀的 prefix 属性等，将直接转移至 CupertinoTextField 组件自身的属性中，而另一部分缺少的属性所对应的功能则不再支持，这是 2 个组件的显著区别之一。

1) placeholder 属性

根据 iOS 设计风格，文本框只支持内容空白时显示一个占位符，一旦用户开始输入就会消失。这样就舍去了 Material 风格的 label(标签)、hint(暗示)和 helper(助手)这 3 种修

饰方式。占位符可用 placeholder 及 placeholderStyle 属性定义，分别设置占位符的内容和样式，代码如下：

```
CupertinoTextField(
  placeholder: "Enter your name",
)
```

运行时，空白的文本框会显示 placeholder 内容，并在用户开始输入后消失，如图 3-13 所示。

图 3-13 空白的文本框会显示 placeholder 内容

2）前缀、后缀与清除按钮

CupertinoTextField 组件支持通过 prefix 和 suffix 属性添加前缀与后缀。这与 TextField 组件的 InputDecoration 中的同名属性一致，可以接收任意组件，非常灵活。同时，前缀与后缀还分别有 prefixMode 和 suffixMode 属性，用于设置它们何时可见。默认情况为 OverlayVisibilityMode.always（永远可见），即只要前缀或后缀属性不为空，就会被渲染出来。此外，该属性还有 editing（仅编辑时可见）、notEditing（仅不编辑时可见）及 never（永不可见）这些值可供选择。

相对于 TextField 组件，这里的 CupertinoTextField 组件还额外添加了一个 clearButtonMode 属性，用于设置文本框末尾的“清除按钮”何时可见。默认情况是 OverlayVisibilityMode.never（永不可见），实战中可根据需要修改这个值，如改为 editing（仅在编辑时可见），则每当文本框的内容不为空且没有后缀时，末尾处就会出现一个用于清空内容的小按钮。清除按钮只可以在没有后缀时出现。

例如，这里定义了一个"https://"字样的 Text 组件，作为永远可见的前缀，又定义了一个五角星图标，作为后缀，仅不编辑时可见（文本框内容为空时才能看到），最后定义了清除按钮（仅在编辑时可见），代码如下：

```
CupertinoTextField(
  prefix: Text("https://"),
  prefixMode: OverlayVisibilityMode.always,
  suffix: Icon(Icons.star_border),
  suffixMode: OverlayVisibilityMode.notEditing,
  clearButtonMode: OverlayVisibilityMode.editing,
)
```

程序运行时首先可以观察到"https://"字样的前缀，接着若文本框内容为空则显示后缀五角星，否则显示清除按钮，如图 3-14 所示。

图 3-14 前缀、后缀与清除按钮的显示效果

如果开发者认为前缀或后缀的组件与 CupertinoTextField 组件边框的留白不足，则可以直接通过插入 Padding 组件增加留白。对此不熟悉的读者可参考第 6 章有关 Padding 组件

的内容。

3）padding 属性

3.1.1 节介绍的 TextField 组件的 InputDecoration 中有 contentPadding 属性，用于设置边框与内容之间的留白。这里的 padding 属性作用与之类似，但却会在有前缀与后缀时，将留白插入文字内容与前后缀之间，而不是前后缀与边框之间。

例如，这里分别将 TextField 组件和 CupertinoTextField 组件设置 16 单位的留白，并设置前后缀装饰，代码如下：

```
Column(
  children: [
    TextField(
      decoration: InputDecoration(
        contentPadding: EdgeInsets.all(16),
        prefix: Icon(Icons.person),
        suffix: Icon(Icons.star),
      ),
    ),
    CupertinoTextField(
      padding: EdgeInsets.all(16),
      prefix: Icon(Icons.person),
      suffix: Icon(Icons.star),
    ),
  ],
)
```

运行后可观察到 TextField 组件与 CupertinoTextField 组件中不同的留白效果，如图 3-15 所示。

图 3-15 TextField 组件与 CupertinoTextField 组件留白效果对比

4）decoration 属性

3.1.1 节介绍的 TextField 中的 decoration 属性支持设置前缀、后缀、提示信息、填充色、边框及错误状态下的文本及样式等众多功能，然而这些设计不完全符合 iOS 风格，因此大部分都被删减。这里的 decoration 属性只接受一个普通的 BoxDecoration 类，主要用于自定义背景填充色、圆角边、阴影及渐变等。例如，可设置一个由黑到白再到黑的渐变色修饰，并添加圆角和阴影效果，代码如下：

```
CupertinoTextField(
  decoration: BoxDecoration(
      gradient: LinearGradient(
        colors: [Colors.black, Colors.white, Colors.black],
      ),
      borderRadius: BorderRadius.circular(24.0),
```

```
        boxShadow: [
          BoxShadow(offset: Offset(4, 4), blurRadius: 4),
        ]),
    padding: EdgeInsets.all(24),
  )
```

虽然 BoxDecoration 提供了这些功能，但一定要注意不可滥用，否则运行效果会非常不符合 iOS 的风格，如图 3-16 所示。

图 3-16 修饰后的文本框

实际上这里用于接收 BoxDecoration 类型的 decoration 属性，本质与 Container 组件或 DecoratedBox 组件的同名属性无异。对此不熟悉的读者也可参考第 14 章“渲染与特效”中关于 DecoratedBox 的内容。

2. 其他属性

除了上述的区别外，CupertinoTextField 组件还有大量与 TextField 组件相同的属性，在此按照字母顺序列出：autoCorrect、autofillHints、autoFocus、controller、cursorColor、cursorRadius、cursorWidth、enabled、enableInteractiveSelection、enableSuggestions、expands、focusNode、inputFormatters、keyboardAppearance、keyboardType、maxLength、maxLengthEnforced、maxLines、minLines、obscureText、obscuringCharacter、onChanged、onEditingComplete、onSubmitted、onTap、readyOnly、scrollController、scrollPadding、scrollPhysics、selectionHeightStyle、selectionWidthStyle、showCursor、smartDashesType、smartQuotesType、style、textAlign、textAlignVertical、textCapitalization、textInputAction、toolbarOptions。

关于这些属性的具体用法及代码示范，可参考 3.1.1 节 TextField 中的相关内容。

3.2 按钮

在 Flutter 2.0 中，最常用的按钮主要有 ElevatedButton 和 TextButton 这 2 个组件，分别对应凸起的和扁平的 Material 风格按钮。另外本章也会介绍 CupertinoButton 组件，用于渲染符合 iOS 风格的按钮。除了这 3 个按钮组件外，Flutter 中还有众多支持用户选择与触碰的小组件，如 DropdownButton(下拉按钮)、IconButton(图标按钮)、CupertinoSlider(滑块)、Switcher(开关)等，将在第 11 章“风格组件”中简单介绍。

3.2.1 ElevatedButton

凸起按钮可为相对扁平的界面增添层次感，并让用户清晰地感受到这个按钮的存在，具有引导用户单击的功效，但若在已经凸起的平面上(如弹出的对话框或 Material 卡片等)再使用凸起按钮，则会使布局显得凌乱，因此一般在此类场景推荐使用扁平的按钮，将在 3.2.2 节讨论。

ElevatedButton 组件最早是于 2020 年 7 月的“按钮大重构”时提出的[①]新组件之一，主要用于替代旧的 RaisedButton 组件。该组件最先被命名为 ContainedButton，随后得到广大开发者的反馈后迅速更名[②]为 ElevatedButton，并于 2020 年 9 月 29 日随 Flutter 1.22.0 正式发布。与旧的 RaisedButton 组件相比，这个新的 ElevatedButton 组件更符合最新的 Material 界面设计要求，如默认用主题色填充等。

按钮组件可通过 child 属性接收一个用于渲染的子组件，以及通过可选的 onPressed 或 onLongPress 属性接收回传函数，分别在用户单击与长按时触发。若这 2 个回传函数都为 null，则按钮会自动变成“禁用”状态，即呈现出灰色的视觉效果且不接受用户单击。

简单地传入 Text 组件作为 child，就可以实现一个默认样式的凸起按钮，代码如下：

```
ElevatedButton(
  child: Text("Click me"),
  onPressed: () => print("用户单击了按钮"),
  onLongPress: () => print("用户长按了按钮"),
)
```

凸起按钮属于 Material 风格按钮，默认使用程序的主题色(如蓝色)，正常状态下配有阴影，以呈现轻微凸起效果。当用户单击时，按钮会提高阴影强度，以增加凸起效果，并同时触发 Material 风格的水波纹效果，如图 3-17 所示。单击完毕后按钮会恢复原来的阴影，并根据用户单击的时长，触发 onPressed 单击事件或 onLongPress 长按事件。

图 3-17　默认样式的凸起按钮及水波纹效果

1. 图标按钮

除了普通的构造函数外，凸起按钮还提供 ElevatedButton.icon()命名构造函数，用于在按钮上添加图标。该函数不再有 child 参数，取而代之的是 icon 和 label 这 2 个参数，可分别接收一个组件，用于渲染图标和标签内容，代码如下：

```
ElevatedButton.icon(
  icon: Icon(Icons.star),                //图标
  label: Text("Click me"),               //标签
  onPressed: () => print("用户单击了按钮"),
  onLongPress: () => print("用户长按了按钮"),
)
```

① https://github.com/flutter/flutter/pull/59702
② https://github.com/flutter/flutter/pull/61262

运行效果如图 3-18 所示。

图 3-18 默认样式的有图标的凸起按钮

值得一提的是，这里的 icon 和 label 属性并不一定要求传入相应的 Icon 组件和 Text 组件。实际上，ElevatedButton.icon 函数背后的工作原理也只是简单地将 icon 与 label 嵌入 Row 容器中，并在二者之间插入一个宽度为 8 单位的 SizedBox 组件以留白。其相关源代码如下：

```
Row(
  mainAxisSize: MainAxisSize.min,
  children: [icon, SizedBox(width: gap), label],
)
```

因此开发者也可以直接使用普通构造函数，通过向 child 属性传入任意组件，或嵌套 Row、Column、Stack 等布局容器，随心所欲地设置按钮内容。

2. ButtonStyle

ElevatedButton 组件的外观样式可通过向 style 参数传入 ButtonStyle 定制，而 ButtonStyle 类的构造函数支持大量参数，可设置不同状态下的字体、颜色、凸起高度甚至按钮形状等。

1）MaterialStateProperty

ButtonStyle 的大部分属性都很好理解，例如可通过 textStyle 设置文本样式、通过 backgroundColor 设置背景色、通过 foregroundColor 设置前景色、通过 overlayColor 设置当按钮被按下时的叠加色、通过 shadowColor 设置阴影的颜色、通过 elevation 设置凸起的高度、通过 padding 设置留白、通过 minimumSize 设置最小尺寸、通过 side 设置边框、通过 shape 设置形状及通过 mouseCursor 设置当鼠标指针滑过时的光标样式等。

但需要注意的是，设置上述这些属性时并不可以直接传入属性所对应的类型。例如，设置 textStyle 时不可以直接传入 TextStyle 类，设置 backgroundColor 时也不可以直接传入 Color 类。实际上，它们都需要接收一个名为 MaterialStateProperty 的类型，以便在不同的 MaterialState 下分别指定不同值。这里的 MaterialState 是指按钮的交互状态，如 pressed（被按下）、focused（有焦点）、dragged（被拖动）、disabled（禁用）等。ButtonStyle 属性接收了 MaterialStateProperty 类，就可以自动根据当前的交互状态采用对应的属性值。

实战中，推荐使用 MaterialStateProperty.resolveWith()方法，判断传入的合集（Set 数据结构）中是否含有特定的交互状态，并依此返回相应的属性值。例如，可在按钮被单击时改变样式，代码如下：

```
//第 3 章/elevated_button_style_resolve.dart

ElevatedButton.icon(
  icon: Icon(Icons.star),
```

```
    label: Text("Click me"),
    onPressed: () => print("用户单击了按钮"),
    style: ButtonStyle(
      backgroundColor: MaterialStateProperty.resolveWith((states) {
        //若按钮处于被单击的状态,则返回红色,否则返回蓝色
        if (states.contains(MaterialState.pressed)) {
          return Colors.red;
        }
        return Colors.blue;
      }),
      textStyle: MaterialStateProperty.resolveWith((states) {
        //若按钮处于被单击的状态,则返回 40 号字,否则返回 20 号字
        if (states.contains(MaterialState.pressed)) {
          return TextStyle(fontSize: 40);
        }
        return TextStyle(fontSize: 20);
      }),
    ),
  )
```

若无须单独对各种交互状态设置不同的值,则可以通过 MaterialStateProperty.all()方法统一设置所有交互状态,代码如下:

```
  ElevatedButton(
    child: Text("Click me"),
    onPressed: () => print("用户单击了按钮"),
    style: ButtonStyle(
      backgroundColor: MaterialStateProperty.all(Colors.red),
      textStyle: MaterialStateProperty.all(TextStyle(fontSize: 24)),
    ),
  )
```

这样就可以将按钮的样式始终设置为红色背景 24 号字,一成不变。

2) 其他样式属性

ButtonStyle 中也有不受交互状态影响的属性,也就不需要传入 MaterialStateProperty 类。

开启反馈 enableFeedback 属性,用于设置当按钮被单击时是否需要产生相应的、除视觉效果之外的反馈。如在安卓系统上,长按按钮会触发设备轻微震动,或在有些设备上单击按钮时会发出哒哒声等。默认开启,这里直接传入属性值 false 即可关闭。

动画时长 animationDuration 属性,用于设置按钮的形状或凸起高度等视觉效果发生改变时的过渡动画的时长,例如传入 Duration(seconds: 1),即可设置为 1s。

视觉密度 visualDensity 属性,可以设置按钮的总体留白情况,默认为 VisualDensity.standard(标准),即比较宽松。除此之外,VisualDensity 类还提供 comfortable 和 compact 值,依次更为紧凑。

触碰检测区 tapTargetSize 属性,用于设置按钮的最小触碰区。默认情况下为

MaterialTapTargetSize.padded，即当按钮的尺寸过小时，会自动保证用户可单击的区域不小于48×48逻辑像素。这是Material设计推荐的最小按钮，否则可能不方便用户单击。若需强制将按钮触碰检测区缩小至按钮本身尺寸，则可将该属性设置为MaterialTapTargetSize.shrinkWrap。当按钮本身尺寸已经超过48×48逻辑像素的最小阈值时该属性不起作用。

3. styleFrom

若需自定义按钮样式，除了通过上面介绍的创建ButtonStyle类的方式外，ElevatedButton还提供了styleFrom方法，方便开发者以凸起按钮的默认样式为起点，进行部分样式的修改。

例如可将按钮的阴影颜色设置为红色，代码如下：

```
ElevatedButton(
  child: Text("Click me"),
  onPressed: () => print("用户单击了按钮"),
  style: ElevatedButton.styleFrom(
    shadowColor: Colors.red,
  ),
)
```

这里可以看到，整段代码没有用到MaterialStateProperty，而是向shadowColor属性直接传入了更加方便且可读性更高的Color类。

大部分ButtonStyle的属性都可以通过这种方式直接传入，而styleFrom方法的内部实际上还会调用MaterialStateProperty.all，将传入的值应用于所有交互状态，但与ButtonStyle稍微不同的是，styleFrom方式缺少backgroundColor和foregroundColor这2个直接设置背景色和前景色的属性，取而代之的是primary、onPrimary及onSurface这3个新属性。

正常交互状态下，primary与onPrimary属性对应backgroundColor与foregroundColor，分别定义按钮的填充底色与前景色(文字图标等内容的颜色)。

但在按钮被禁用时，onSurface属性则负责同时定义按钮的前景和背景色。例如传入onSurface：Colors.red并将onPressed设置为null禁用按钮，即可观察到按钮呈现出红色前景文字与淡红色背景填充，无法单独设置。从这里可以看出，styleFrom主要是为了简化常见的开发场景。若需深度定制，开发者应直接使用ButtonStyle配合MaterialStateProperty，单独控制每个交互状态的样式。

3.2.2 TextButton

TextButton组件是一个符合Material设计风格的扁平按钮，常用于工具栏或菜单中，避免由多个凸起按钮的边框与阴影所造成的视觉拥挤。同时一般建议在已经凸起的平面(如弹出的对话框等)使用扁平按钮，避免再次凸起，破坏布局的整体性。由于没有边框，设

计时需要注意让用户能清晰辨认出这是一个可供单击的按钮，而不是普通文字。

1. 基础用法

使用 TextButton 组件时一般通过 child 属性传入一个 Text 组件，再通过 onPressed 或 onLongPress 回传函数监听按钮的单击或长按事件。在没有被单击时，扁平按钮在视觉上与 Text 组件除默认颜色外并无差异，但当用户按下按钮时则会出现填充色及水波纹效果，代码如下：

```
TextButton(
    child: Text("Click me"),
    onPressed: () => print("Hi"),
)
```

运行后会得到一个写有 Click me 字样的扁平按钮，默认为蓝色字样。与凸起按钮一样，扁平按钮也属于 Material 风格。用户单击时会由手指触碰的区域为起点，出现淡色水波纹效果并渐渐扩散直至填满整个按钮，如图 3-19 所示。用户单击完毕后水波纹会消失，并根据用户单击的时长，触发 onPressed 单击事件或 onLongPress 长按事件。

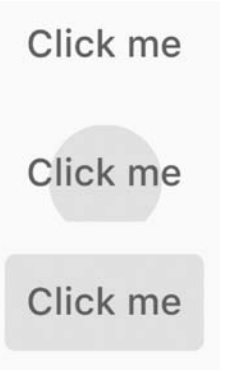

图 3-19 扁平按钮被单击时的水波纹效果

TextButton 组件较新，它与 3.2.1 节介绍的 ElevatedButton 同时出现于 2020 年 9 月 29 日正式发布的 Flutter 1.22.0 版本中，目标是分别取代旧的 FlatButton 和 RaisedButton 组件。这 2 个新按钮的用法高度一致，建议不熟悉的读者直接阅读 3.2.1 节介绍 ElevatedButton 组件的相关内容。

2. styleFrom

在 3.2.1 节介绍的 ElevatedButton.styleFrom 中，primary 是指按钮的背景色，即填充底色，而 onPrimary 才是前景的文字或图标的颜色，但扁平按钮默认为没有背景色，因此在这里的 TextButton.styleFrom 中，primary 则直接指按钮的前景色，即文字或图标等子组件的颜色，而 onPrimary 属性不存在。

类似地，之前介绍的凸起按钮中 onSurface 属性会同时定义禁用后按钮的前景与背景，但由于扁平按钮默认没有背景，因此这里的 onSurface 属性也只负责定义按钮被禁用时的前景色。

若需为扁平按钮添加背景填充，则可以使用 TextButton.styleFrom 中特有的 backgroundColor 属性统一设置所有交互状态。若需深度订制，则开发者可直接使用 ButtonStyle 配合 MaterialStateProperty，分别控制每个交互状态的样式。

3.2.3 CupertinoButton

CupertinoButton 是一个用于渲染符合 iOS 风格的按钮组件。它提供 CupertinoButton()和

CupertinoButton.filled()这 2 种构造函数，分别对应无填充与有填充色的按钮，代码如下：

```
Column(
  children: [
    CupertinoButton(
      child: Text("Cupertino Default"),
      onPressed: () {},
    ),
    CupertinoButton.filled(
      child: Text("Cupertino Filled"),
      onPressed: () {},
    ),
  ],
)
```

无填充时文字为程序主题色(如蓝色)，有填充色时则填充色为主题色，如图 3-20 所示。

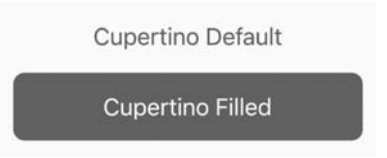

图 3-20 无填充与有填充色的按钮默认样式

Cupertino 风格的按钮没有水波纹效果。当用户单击时按钮会呈现半透明状，并在单击结束时恢复。该按钮不支持长按，因此无论用户按下按钮后保持多久，松开后都会触发 onPressed 回传函数。

除了 child 和 onPress 属性外，CupertinoButton 组件的其他属性都用于设置按钮样式，这里逐一介绍。

1) padding

CupertinoButton 按钮的内容与边框之间的留白可由 padding 属性设置。默认情况下，无填充的按钮四周都有 16 单位的留白，而有填充的按钮则在内容的左后分别有 64 单位的留白，上下有 14 单位的留白。这些默认留白数值是 Flutter 团队在 XCode 开发环境中通过测量 iOS 12 的原生按钮所得。在 CupertinoButton 中的相关源代码如下：

```
const EdgeInsets _kButtonPadding = EdgeInsets.all(16.0);
const EdgeInsets _kBackgroundButtonPadding = EdgeInsets.symmetric(
  vertical: 14.0,
  horizontal: 64.0,
);
```

当屏幕空间不足且按钮文字较长时，开发者可适当降低按钮的留白，否则可能最终渲染出的效果并不好，如图 3-21 所示。

2) borderRadius

默认情况下 CupertinoButton 组件有 8 逻辑像素的圆角，可以通过 borderRadius 属性修改。例如传入 borderRadius：BorderRadius.circular(16.0)可将圆角边增至 16 单位。

3）pressedOpacity

当用户单击按钮时，CupertinoButton 会出现半透明效果，默认不透明度值为 0.4，即 40%不透明，如图 3-22 所示。

图 3-21　文字较长时，默认留白效果不佳

图 3-22　Cupertino 按钮被单击的效果

如有必要，则这里可传入一个 0.0～1.0 的值修改按钮被按下时的不透明程度。

4）color

在使用默认构造函数时，color 属性可用于设置按钮的填充色，然而使用 CupertinoButton.filled()构造函数时，按钮已经自动填充了程序的主题色，因此没有该属性。换言之，使用 filled 构造函数就相当于将 color 属性设置为程序的主题色，如蓝色。

5）disabledColor

这是当按钮被禁用(onPressed 属性为 null)时的按钮颜色，默认为淡灰色。一般不需要改动，否则若颜色过于鲜艳可能视觉上没有“无法单击”的感觉，易对用户造成困扰，从而影响用户体验。

6）minSize

最小尺寸只当按钮内容较少而导致按钮尺寸不够大时生效，主要用于保证用户可以轻易地单击该按钮。根据苹果官方发布的界面设计指南，按钮尺寸至少为 44×44 单位[①]，即该属性的默认值，因此一般不建议修改该属性。

① https://developer.apple.com/design/human-interface-guidelines/ios/visual-design/adaptivity-and-layout