BeeWare Documentation

发行版本 *0.1.dev155+g6381907*

Russell Keith-Magee

2024 年 05 月 13 日

Contents

1 什么是 BeeWare？ 3

2 让我们开始吧！ 5

2.1 教程 0 - 准备就绪！ 5
2.2 教程 1 - 您的第一个应用程序 8
2.3 教程 2 - 使之更有趣 12
2.4 教程 3 - 包装以便分发 18
2.5 教程 4 - 更新应用程序 26
2.6 教程 5 - 移动操作 30
2.7 教程 6 - 上网！ 40
2.8 教程 7 - 启动这个（第三）派对 44
2.9 教程 8 - 使其光滑 52
2.10 教程 9 - 测试时间 54
2.11 教程 10 - 制作属于自己的应用程序 63

用 Python 编写，随处运行。

欢迎来到 BeeWare！在本教程中，我们将使用 Python 构建一个图形用户界面，并将其部署为桌面应用程序、移动应用程序和单页 Web 应用程序。我们还将探讨如何使用 BeeWare 工具执行应用程序开发者需要做的一些常见任务，例如测试你的应用。

本页中文文档基于 BeeWare 的英文文档翻译，已经由人工审核！

本页中文文档基于 BeeWare 的英文文档翻译，已经由人工审核。

如果您想帮助改进翻译，请联系我们！我们在 Discord 中有一个 `#translations` 频道；在那儿介绍自己，我们会邀请您加入翻译团队。

CHAPTER 1

什么是 BeeWare？

BeeWare 并不是一个单独的产品、工具或库 (library)，它是一系列工具和库的集合：每个工具和库都能协同工作，帮助您编写跨平台、具有本地图形用户界面的 Python 应用程序。它包括:

- Toga，一个跨平台的 widget (小部件，控件) 工具包;
- Briefcase，一个用于将 Python 项目打包为可分发的成品，可以发送给最终用户的工具;
- 库（如 Rubicon ObjC），用于访问平台原生库的库;
- 预编译的 Python 构建版本，可在官方 Python 安装程序不可用的平台上使用。

在这个教程中，我们将使用所有这些工具，但作为用户，你只需要与前两个（Toga 和 Briefcase）互动。然而，每个工具也可以单独使用 - 例如，你可以使用 Briefcase 部署应用程序，而不使用 Toga 作为 GUI 工具包。

BeeWare 套件可用于：macOS、Windows、Linux（使用 GTK）；在移动平台如 Android 和 iOS；以及 Web 上。

CHAPTER 2

让我们开始吧！

准备好亲身体验 BeeWare 了吗？*让我们用 Python 构建一个跨平台应用程序！*

2.1 教程 0 - 准备就绪！

在构建第一个 BeeWare 应用程序之前，我们必须确保已具备运行 BeeWare 的所有先决条件。

2.1.1 安装 Python

首先，我们需要一个正常工作的 Python 解释器。

macOS

Linux

Windows

如果您使用的是 macOS，Xcode 或命令行开发工具中会包含最新版本的 Python。要检查是否已安装，请运行以下命令：

```
$ python3 --version
```

如果已安装 Python，则会看到其版本号。否则，系统会提示您安装命令行开发工具。

如果您使用的是 Windows 系统，可以从 Python 网站 <https://www.python.org/downloads>‘_ 获取官方安装程序。您可以使用 Python 3.8 及以后的任何稳定版本。我们建议您不要使用 alphas、beta 和候选发布版，除非您 ** 了解自己在做什么。

如果您使用的是 Linux，您将使用系统软件包管理器（Debian/Ubuntu/Mint 上的 `apt`，Fedora 上的 `dnf`，或 Arch 上的 `pacman`）安装 Python。

您应该确保系统 Python 是 Python 3.8 或更新版本；如果不是（例如，Ubuntu 18.04 附带 Python 3.6），您需要将 Linux 发行版升级到更新版本。

目前对 Raspberry Pi 的支持有限。

如果您使用的是 Windows 系统，可以从 Python 网站 <https://www.python.org/downloads>‘_ 获取官方安装程序。您可以使用 Python 3.8 及以后的任何稳定版本。我们建议您不要使用 alphas、beta 和候选发布版，除非您 ** 了解自己在做什么。

其他 Python 发行版

安装 Python 有很多不同的方法。可以通过 homebrew 安装 Python。您可以使用 pyenv 来管理同一台机器上的多个 Python 安装。Windows 用户可以从 Windows 应用商店安装 Python。数据科学背景的用户可能想使用 Anaconda 或 Miniconda。

如果您使用的是 macOS 或 Windows 操作系统，如何安装 Python 并不重要，重要的是您能从操作系统的命令提示符/终端应用程序中运行 `python3`，并获得一个正常工作的 Python 解释器。

如果您使用的是 Linux，则应使用操作系统提供的系统 Python。您可以使用非系统 Python 完成本教程的 * 大部分内容，但无法将应用程序打包发布给他人。

2.1.2 安装依赖项

接下来，安装操作系统所需的其他依赖项：

macOS

Linux

Windows

在 macOS 上构建 BeeWare 应用程序需要：

- 版本控制系统 *Git*。它包含在您安装的 Xcode 或命令行开发工具中。

为支持本地开发，您需要安装一些系统软件包。所需的软件包列表因发行版而异：

Ubuntu 20.04+ / Debian 10+

```
$ sudo apt update
$ sudo apt install git build-essential pkg-config python3-dev python3-venv␣
↪libgirepository1.0-dev libcairo2-dev gir1.2-gtk-3.0 libcanberra-gtk3-module
```

费多拉

```
$ sudo dnf install git gcc make pkg-config rpm-build python3-devel gobject-
↪introspection-devel cairo-gobject-devel gtk3 libcanberra-gtk3
```

Arch、Manjaro

```
$ sudo pacman -Syu git base-devel pkgconf python3 gobject-introspection cairo gtk3␣
↪libcanberra
```

OpenSUSE Tumbleweed

```
$ sudo zypper install git patterns-devel-base-devel_basis pkgconf-pkg-config python3-
↪devel gobject-introspection-devel cairo-devel gtk3 'typelib(Gtk)=3.0' libcanberra-
↪gtk3-module
```

在 Windows 上构建 BeeWare 应用程序需要：

- **Git**，一个版本控制系统。你可以从 git-scm.org 下载 Git。

安装这些工具后，应确保重新启动任何终端会话。Windows 只在安装完成后才会显示新安装的工具终端。

2.1.3 建立虚拟环境

我们现在要创建一个虚拟环境–一个“沙箱”，用来将本教程的工作与我们的主 Python 安装隔离开来。如果我们将软件包安装到虚拟环境中，我们的主 Python 安装（以及计算机上的任何其他 Python 项目）将不会受到影响。如果我们把虚拟环境弄得一团糟，我们可以简单地删除它，然后重新开始，不会影响计算机上的任何其他 Python 项目，也不需要重新安装 Python。

macOS

Linux

Windows

```
$ mkdir beeware-tutorial
$ cd beeware-tutorial
$ python3 -m venv beeware-venv
$ source beeware-venv/bin/activate
```


```
$ mkdir beeware-tutorial
$ cd beeware-tutorial
$ python3 -m venv beeware-venv
$ source beeware-venv/bin/activate
```


```
C:\...>md beeware-tutorial
C:\...>cd beeware-tutorial
C:\...>py -m venv beeware-venv
C:\...>beeware-venv\Scripts\activate
```

运行 PowerShell 脚本时出现的错误

如果使用 PowerShell 时收到错误信息：:

```
File C:\...\beeware-tutorial\beeware-venv\Scripts\activate.ps1 cannot be loaded␣
↪because running scripts is disabled on this system.
```

您的 Windows 账户没有运行脚本的权限。要解决这个问题:

1. 以管理员身份运行 Windows PowerShell。
2. 运行 `set-executionpolicy RemoteSigned`
3. 选择 `Y` 更改执行策略。

完成后，您就可以在原始 PowerShell 会话（或同一目录下的新会话）中重新运行 `beeware-venv\scripts\activate.ps1`。

如果这样做成功了，您的提示符现在应该有所改变–它应该带有 `(beeware-venv)` 前缀。这样，您就可以知道当前正处于 BeeWare 虚拟环境中。无论何时运行本教程，都应确保虚拟环境已激活。如果没有激活，请重新执行最后一条命令（`activate` 命令）以重新激活虚拟环境。

替代虚拟环境

如果您使用的是 Anaconda 或 miniconda，那么您可能对使用 conda 环境更为熟悉。您可能还听说过 `virtualenv`，它是 Python 内置的 `venv` 模块的前身。就像 Python 的安装一样，如果你使用的是 macOS 或 Windows，那么如何创建虚拟环境并不重要，只要有一个就够了。如果在 Linux 上，则应坚持使用 `venv` 和系统 Python。

2.1.4 下一步

现在我们已经设置好了环境。我们准备创建第一个 *BeeWare 应用程序*。

2.2 教程 1 - 您的第一个应用程序

我们准备好创建第一个应用程序了。

2.2.1 安装 BeeWare 工具

首先，我们需要安装 **Briefcase**。Briefcase 是一款 BeeWare 工具，可用于打包应用程序，以便分发给最终用户，但也可用于引导新项目。确保您在*Tutorial 0* 中创建的 `beeware-tutorial` 目录中，并激活 `beeware-venv` 虚拟环境，然后运行：

macOS

Linux

Windows

```
(beeware-venv) $ python -m pip install briefcase
```

```
(beeware-venv) $ python -m pip install briefcase
```

安装过程中可能出现的错误

如果在安装过程中出现错误，几乎可以肯定是因为某些系统要求尚未安装。请确保您已安装了所有平台先决条件。

```
(beeware-venv) C:\...>python -m pip install briefcase
```

安装过程中可能出现的错误

请务必使用 `python -m pip`，而不是简单的 `pip`。公文包需要确保它拥有最新版本的 ``pip` 和 `setuptools`，而裸调用的 `pip` 无法自我更新。如果你想了解更多，Brett Cannon 有一篇关于这个问题的详细博文。

BeeWare 工具之一是 **Briefcase**。公文包可用于打包应用程序，以便分发给最终用户，但也可用于引导新项目。

2.2.2 引导新项目

让我们开始第一个 BeeWare 项目！我们将使用 Briefcase `new` 命令创建一个名为 **Hello World** 的应用程序。在命令提示符下运行以下命令:

macOS

Linux

Windows

```
(beeware-venv) $ briefcase new
```

```
(beeware-venv) $ briefcase new
```

```
(beeware-venv) C:\...>briefcase new
```

公文包会要求我们提供新应用程序的一些详细信息。在本教程中，请使用以下内容:

- **正式名称** - 接受默认值：你好，世界。
- **App Name** - 接受默认值：`helloworld`。
- **捆绑** - 如果您拥有自己的域名，请按相反顺序输入该域名。(例如，如果您拥有域名 “cupcakes.com”，则输入 `com.cupcakes` 作为捆绑)。如果您没有自己的域名，请接受默认的捆绑包（`com.example`）。
- **项目名称** - 接受默认值：你好世界。
- **Description** - 接受默认值（或者，如果您想发挥自己的创造力，也可以提出自己的描述!）。
- **作者** - 在此处输入您自己的姓名。
- **作者电子邮件** - 输入您自己的电子邮件地址。这将用于配置文件、帮助文本以及向应用程序商店提交应用程序时需要电子邮件的任何地方。
- **URL** - 应用程序登陆页面的 URL。同样，如果您拥有自己的域名，请输入该域名的 URL（包括 `https://`）。否则，只需接受默认 URL (`https://example.com/helloworld`)。此 URL 不需要实际存在（暂时）；只有在您将应用程序发布到应用程序商店时才会使用。
- **许可证** - 接受默认许可证（BSD）。但这不会影响本教程的任何操作，因此，如果您对许可证选择有特别强烈的意见，请随意选择其他许可证。
- **图形用户界面框架** - 接受默认选项 Toga（BeeWare 自己的图形用户界面工具包）。

然后，Briefcase 会生成一个项目骨架供你使用。如果你已经按照本教程学习到目前为止，并接受了所述的默认设置，那么你的文件系统应该是这样的::

```
beeware-tutorial/
    beeware-venv/
        ...
    helloworld/
        CHANGELOG
        LICENSE
        README.rst
        pyproject.toml
        src/
            helloworld/
                resources/
                    helloworld.icns
                    helloworld.ico
                    helloworld.png
```

(续下页)

(接上页)

```
                __init__.py
                __main__.py
                app.py
        tests/
            __init__.py
            helloworld.py
            test_app.py
```

该骨架实际上是一个功能完备的应用程序，无需添加任何其他内容。rc “文件夹包含应用程序的所有代码，” tests “文件夹包含初始测试套件，” pyproject.toml “文件描述了如何打包发布应用程序。如果用编辑器打开 `pyproject.toml`，就会看到刚才提供给 Briefcase 的配置详细信息。

现在我们有了一个存根应用程序，可以使用 Briefcase 来运行该应用程序。

2.2.3 在开发者模式下运行应用程序

移动到 `helloworld` 项目目录，告诉公文包以开发者（或 `dev`）模式启动项目：

macOS

Linux

Windows

```
(beeware-venv) $ cd helloworld
(beeware-venv) $ briefcase dev

[hello-world] Installing requirements...
...

[helloworld] Starting in dev mode...
=============================================================================
```

```
(beeware-venv) $ cd helloworld
(beeware-venv) $ briefcase dev

[hello-world] Installing requirements...
...

[helloworld] Starting in dev mode...
=============================================================================
```

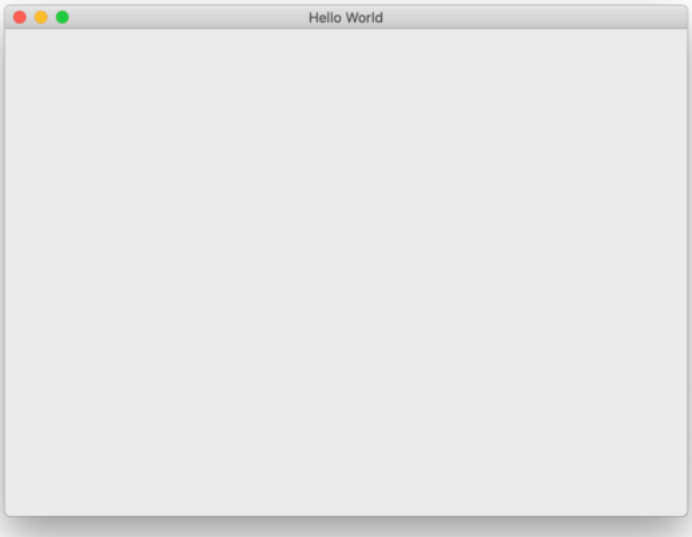
```
(beeware-venv) C:\...>cd helloworld
(beeware-venv) C:\...>briefcase dev

[hello-world] Installing requirements...
...

[helloworld] Starting in dev mode...
=============================================================================
```

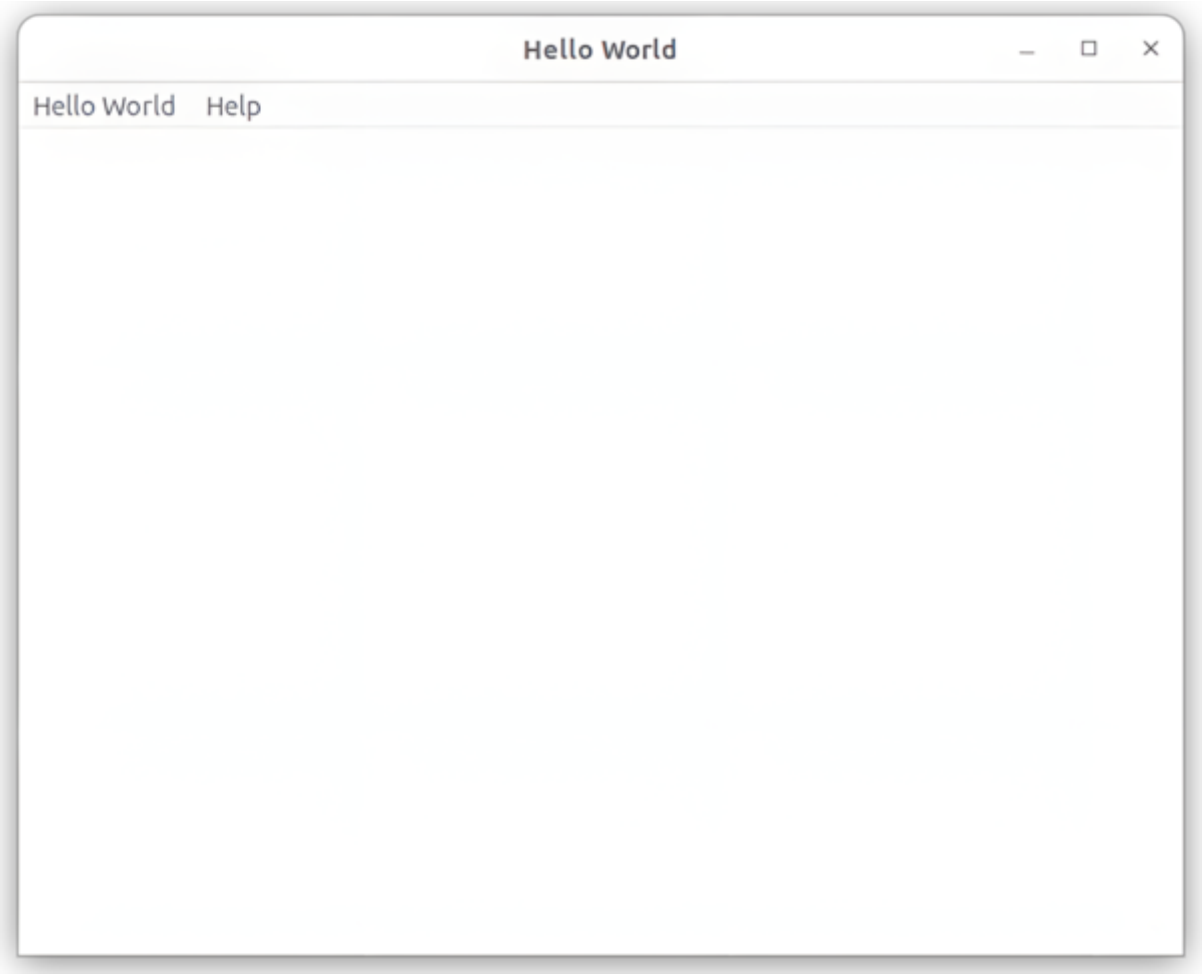
这将打开一个图形用户界面 (GUI) 窗口：

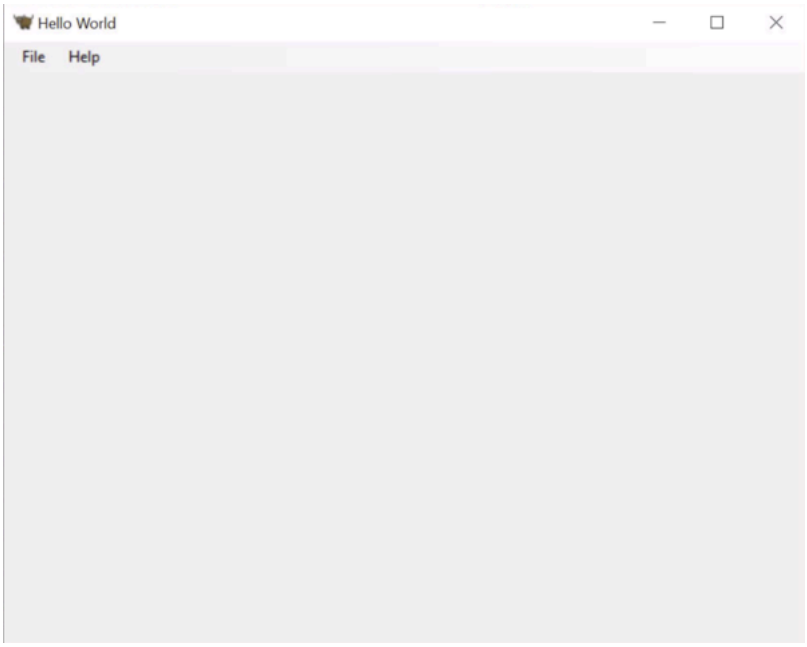
macOS

Linux

Windows







按下关闭按钮（或从应用程序菜单中选择“退出”），就大功告成了！恭喜 - 你刚刚用 Python 编写了一个独立的本地应用程序！

2.2.4 下一步

现在，我们有了一个在开发者模式下运行的应用程序。现在我们可以添加一些自己的逻辑，让应用程序做一些更有趣的事情。在*Tutorial 2* 中，我们将为应用程序添加一个更有用的用户界面。

2.3 教程 2 - 使之更有趣

在*Tutorial 1* 中，我们生成了一个可以运行的基础项目，但我们自己并没有编写任何代码。让我们看看为我们生成了什么。

2.3.1 生成的内容

在 `src/helloworld` 目录中，你应该看到 3 个文件：__init__.py“、__main__.py“ 和 *app.py*‘。

__init__.py‘将 `helloworld` 目录标记为可导入的 Python 模块。这是一个空文件；它的存在告诉 Python 解释器 helloworld 目录定义了一个模块。

`__main__.py` 将 `helloworld` 模块标记为一种特殊的模块 - 可执行模块。如果你尝试使用 `python -m helloworld` 试图运行 `helloworld` 模块，Python 将从 “__main__.py“ 文件开始执行。`__main__.py` 的内容相对简单:

```
from helloworld.app import main

if __name__ == '__main__':
    main().main_loop()
```

也就是说，它从 `helloworld` 应用程序中导入 `main` 方法；如果它作为入口点执行，则调用 main() 方法，并启动应用程序的主循环。主循环是 GUI 应用程序监听用户输入（如鼠标点击和键盘按下）的方式。

更有趣的文件是 app.py - 它包含创建我们应用程序窗口的逻辑:

```
import toga
from toga.style import Pack
from toga.style.pack import COLUMN, ROW

class HelloWorld(toga.App):
    def startup(self):
        main_box = toga.Box()

        self.main_window = toga.MainWindow(title=self.formal_name)
        self.main_window.content = main_box
        self.main_window.show()

def main():
    return HelloWorld()
```

让我们逐行查看:

```
import toga
from toga.style import Pack
from toga.style.pack import COLUMN, ROW
```

首先，我们导入 `toga` 小部件工具包，以及一些与样式相关的实用类和常量。目前我们的代码还没有使用这些——但我们很快就会使用它们。

然后，我们定义了一个类:

```
class HelloWorld(toga.App):
```

每个 Toga 应用程序都有一个 `toga.App` 实例，代表应用程序的运行实体。应用程序最终可能会管理多个窗口；但是对于简单的应用程序来说，可能只有一个主窗口。

接下来，我们定义一个 `startup()` 方法 (startup 意为启动):

```
def startup(self):
    main_box = toga.Box()
```

startup 方法的第一件事是定义一个主盒子 (main box)。Toga 的布局方案类似于 HTML。你通过构造一系列盒子 (box) 来构建应用程序，每个盒子包含其他盒子或实际的小部件 (widgets)。然后，你对这些盒子应用样式 (styles)，以定义它们将如何消耗可用的窗口空间 (window space)。

在这个应用程序中，我们定义了一个单独的空盒子 (我们没有放任何东西进去)。

接下来，我们定义一个窗口，并将这个空盒子放入其中:

```
self.main_window = toga.MainWindow(title=self.formal_name)
```

这将创建一个 toga.MainWindow 的实例，它的标题 (title) 将与应用程序的名称 (self.formal_name) 匹配。主窗口是 Toga 中的一种特殊窗口——它是与应用程序的生命周期 (life cycle) 密切绑定的窗口。当主窗口关闭时，应用程序退出。主窗口也是具有应用程序菜单的窗口（如果你在像 Windows 这样的平台上，菜单栏是窗口的一部分；如果你没有进行任何菜单栏的增删操作，你将看到默认的 file (文件) 和 help (帮助) 这两个菜单栏选项）

然后，我们将空盒子作为主窗口的内容，并指示应用程序显示我们的窗口:

```
self.main_window.content = main_box
self.main_window.show()
```

最后，我们定义一个 `main()` 方法。它将创建应用程序的实例::

```
def main():
    return HelloWorld()
```

这个 main() 方法由 __main__.py 导入并调用。它创建并返回我们的 HelloWorld 应用程序的实例。

这是最简单的可能的 Toga 应用程序。接下来让我们在应用程序中加入一些自己的内容，使应用程序做一些有趣的事情。

2.3.2 添加一些我们自己的内容

修改 src/helloworld/app.py 中的 HelloWorld 类，使其看起来像这样::

```
class HelloWorld(toga.App):
    def startup(self):
        main_box = toga.Box(style=Pack(direction=COLUMN))

        name_label = toga.Label(
            "Your name: ",
            style=Pack(padding=(0, 5))
        )
        self.name_input = toga.TextInput(style=Pack(flex=1))

        name_box = toga.Box(style=Pack(direction=ROW, padding=5))
        name_box.add(name_label)
        name_box.add(self.name_input)

        button = toga.Button(
            "Say Hello!",
            on_press=self.say_hello,
            style=Pack(padding=5)
        )

        main_box.add(name_box)
        main_box.add(button)

        self.main_window = toga.MainWindow(title=self.formal_name)
        self.main_window.content = main_box
        self.main_window.show()

    def say_hello(self, widget):
        print(f"Hello, {self.name_input.value}")
```

备注： 不要删除 app.py 文件顶部的导入 (import)，也不要删除底部的 main()。您只需更新 HelloWorld 类。

让我们详细看看有哪些变化。

我们仍然在创建一个主盒子；然而，现在我们正在应用一个样式:

```
main_box = toga.Box(style=Pack(direction=COLUMN))
```

Toga 的内置布局系统称为 “Pack”（包）。它的行为很像 CSS (Cascading Style Sheets 层叠样式表)。你可以在一个层次结构中定义对象–在 HTML 中，对象是 (division 块级容器)、 (inline span 内联容器) 和其他 DOM 元素 (Document Object Model 文档对象模型)；在 Toga 中，对象是部件 (widgets) 和盒子 (boxes)。然后，您可以为各个元素指定样式。在本例中，我们表示这是一个 COLUMN (垂直) 框，也就是说，它是一个将占用所有可用宽度 (width) 的框，并会随着内容的添加而扩大高度 (height)，但会尽量使高度更短。

接下来，我们定义了一些小部件:

```
name_label = toga.Label(
    "Your name: ",
    style=Pack(padding=(0, 5))
)
self.name_input = toga.TextInput(style=Pack(flex=1))
```

在这里，我们定义了一个标签 (toga.Label) 和一个文本输入框 (toga.TextInput)。这两个小部件都有相关的样式；标签左右各有 5px 的填充，上下没有填充 (padding=(0, 5)。文本输入框被标记为灵活的 (flex=1)——也就是说，它将吸收其布局横向方向上所有可用的空间。

文本输入框被分配为类的实例变量 (self.name_input)。这使我们能够轻松访问小部件 (widget) 实例 - 这是我们马上就会使用的东西。

接下来，我们定义了一个盒子来容纳这两个小部件:

```
name_box = toga.Box(style=Pack(direction=ROW, padding=5))
name_box.add(name_label)
name_box.add(self.name_input)
```

name_box 就像主盒子一样；然而，这次它是一个 ROW (水平) 盒子。这意味着内容将水平添加，并且它会尽量使其宽度尽可能窄 (以确保屏幕横向方向能容纳下整个水平盒子)。盒子也有一些空白填充 (以提高可读性)——四周各为 5px。

现在我们定义了一个按钮:

```
button = toga.Button(
    "Say Hello!",
    on_press=self.say_hello,
    style=Pack(padding=5)
)
```

按钮的四周也有 5px 的填充。我们还定义了一个 * 处理程序 * (handler)——当按钮被按下时要调用的方法 (on_press=self.say_hello)。

然后，我们将名称盒子和按钮添加到主盒子中:

```
main_box.add(name_box)
main_box.add(button)
```

这完成了我们的布局；其余的 startup 方法与以前一样 - 定义一个 MainWindow，并将主盒子指定为窗口的内容:

```
self.main_window = toga.MainWindow(title=self.formal_name)
self.main_window.content = main_box
self.main_window.show()
```

我们需要做的最后一件事是定义按钮的处理器。处理器可以是任何方法、生成器或异步协程；它接受生成事件的小部件 (widget) 作为参数，并且每当按下按钮时就会被调用:

```
def say_hello(self, widget):
    print(f"Hello, {self.name_input.value}")
```

方法的主体是一个简单的打印语句——然而，它会使用名称输入的当前值 (self.name_input.value)，并使用该内容作为打印的文本。

现在我们已经做了这些更改，我们可以通过再次启动应用程序来看看它们的样子。和以前一样，我们将使用开发者模式:

macOS

Linux

Windows

```
(beeware-venv) $ briefcase dev

[helloworld] Starting in dev mode...
========================================================================
```

```
(beeware-venv) $ briefcase dev

[helloworld] Starting in dev mode...
========================================================================
```

```
(beeware-venv) C:\...>briefcase dev

[helloworld] Starting in dev mode...
========================================================================
```

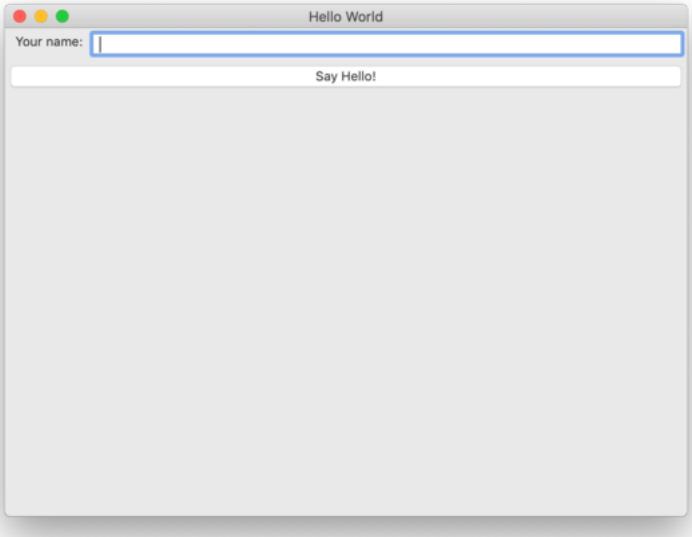
你会注意到，这次它 * 没有 * 安装依赖项。Briefcase 可以检测到应用程序已经运行过，为了节省时间，它只会运行应用程序。如果你在应用程序中添加了新的依赖项，你可以在运行 `briefcase dev` 时通过 `-r` 选项来确保它们被安装。

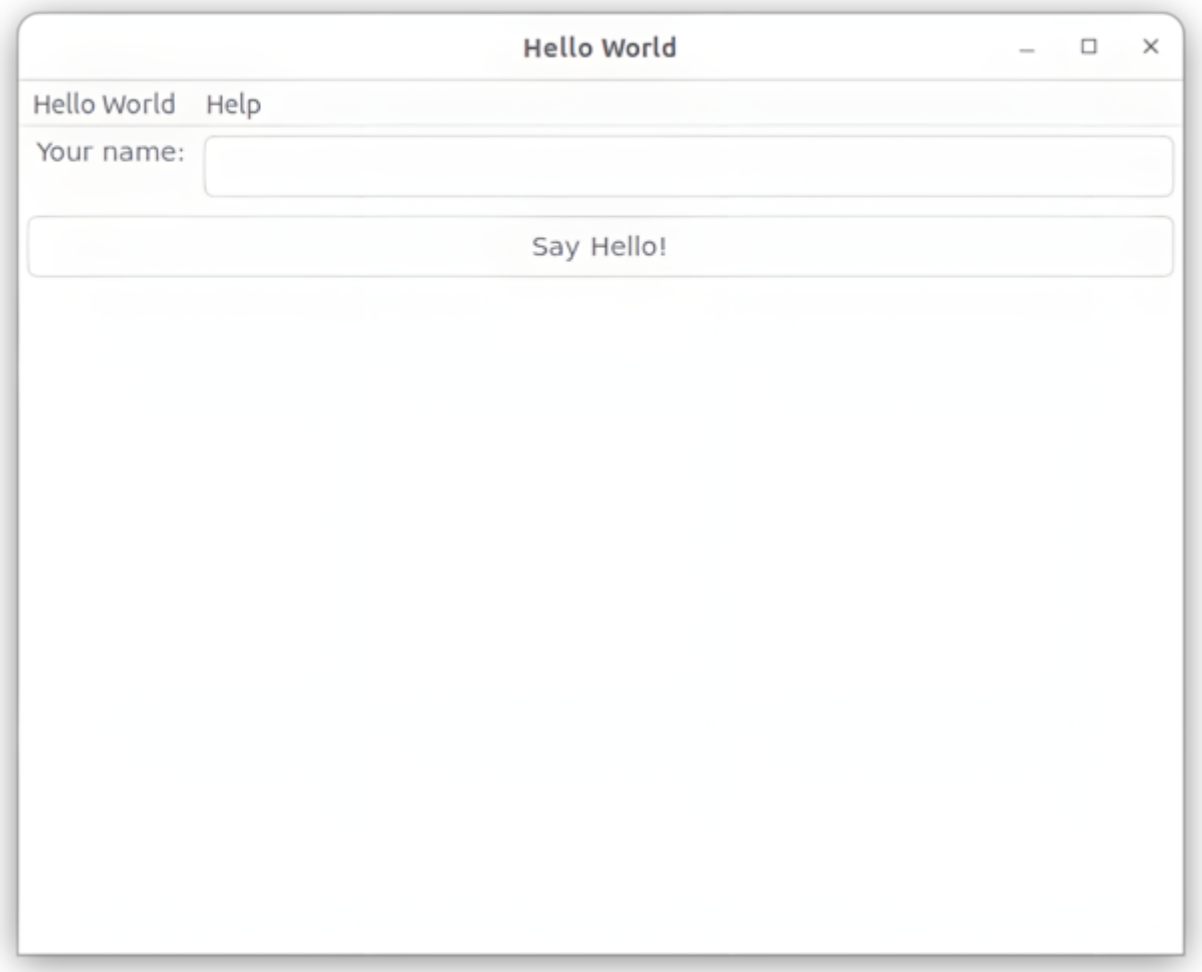
这将打开一个图形用户界面 (GUI) 窗口：

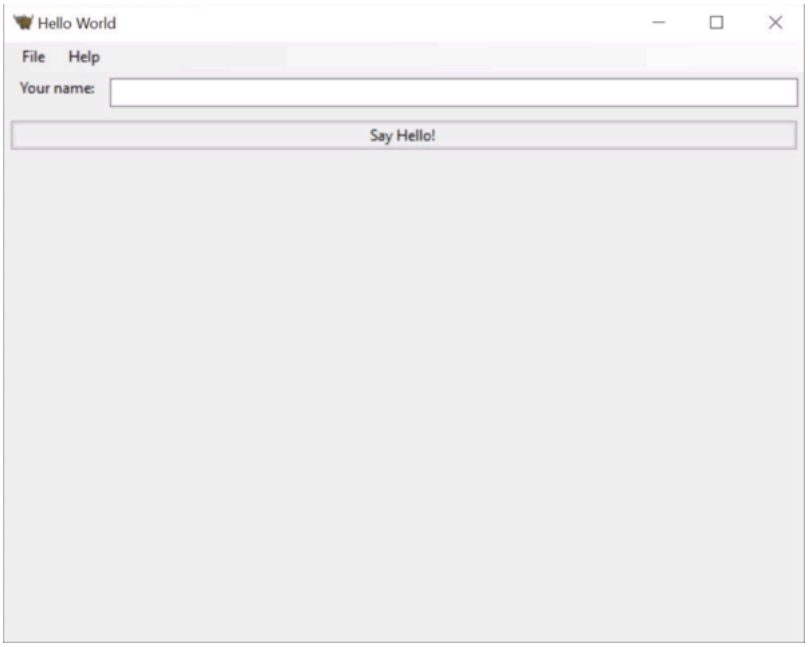
macOS

Linux

Windows







如果在文本框中输入名称并按下 GUI 按钮，就会在启动程序的控制台中看到输出结果。

2.3.3 下一步

我们现在有了一个做一些更有趣的事情的应用程序。但它只能在我们自己的电脑上运行。让我们将这个应用程序打包以供分发。在*Tutorial 3* 中，我们将把应用程序打包成一个独立的安装程序，我们可以发送给朋友、客户，或上传到应用商店。

2.4 教程 3 - 包装以便分发

到目前为止，我们一直在“开发者模式“下运行应用程序。这让我们可以轻松地在本地运行应用程序，但我们真正想要的是能将我们的应用程序提供给其他人。

但是，我们不想教用户如何安装 Python、创建虚拟环境、克隆 git 仓库并在开发者模式下运行 Briefcase。我们宁可只给他们一个安装程序，让应用程序能够正常运行。

公文包可用于打包您的应用程序，以便以这种方式发布。

2.4.1 创建应用程序脚手架

由于这是第一次打包应用程序，我们需要创建一些配置文件和其他脚手架来支持打包过程。在 `helloworld` 目录中，运行:

macOS

Linux

Windows

```
(beeware-venv) $ briefcase create

[helloworld] Generating application template...
Using app template: https://github.com/beeware/briefcase-macOS-app-template.git,␣
↪branch v0.3.14
...

[helloworld] Installing support package...
...

[helloworld] Installing application code...
Installing src/helloworld... done

[helloworld] Installing requirements...
...

[helloworld] Installing application resources...
...

[helloworld] Removing unneeded app content...
...

[helloworld] Created build/helloworld/macos/app
```

```
(beeware-venv) $ briefcase create

[helloworld] Finalizing application configuration...
Targeting ubuntu:jammy (Vendor base debian)
```

(续下页)

(接上页)

```
Determining glibc version... done

Targeting glibc 2.35
Targeting Python3.10

[helloworld] Generating application template...
Using app template: https://github.com/beeware/briefcase-linux-AppImage-template.git,␣
↪branch v0.3.14
...

[helloworld] Installing support package...
No support package required.

[helloworld] Installing application code...
Installing src/helloworld... done

[helloworld] Installing requirements...
...

[helloworld] Installing application resources...
...

[helloworld] Removing unneeded app content...
...

[helloworld] Created build/helloworld/linux/ubuntu/jammy
```

```
(beeware-venv) C:\...>briefcase create

[helloworld] Generating application template...
Using app template: https://github.com/beeware/briefcase-windows-app-template.git,␣
↪branch v0.3.14
...

[helloworld] Installing support package...
...

[helloworld] Installing application code...
Installing src/helloworld... done

[helloworld] Installing requirements...
...

[helloworld] Installing application resources...
...

[helloworld] Created build\helloworld\windows\app
```

您可能刚刚在终端上看到了几页内容……那么刚刚发生了什么？公文包做了以下工作：

1. 它 ** 生成一个应用程序模板 **。除了实际应用程序的代码之外，构建本机安装程序还需要许多文件和配置。这些额外的脚手架对于同一平台上的每个应用程序几乎都是一样的，除了正在构建的实际应用程序的名称–因此，Briefcase 为其支持的每个平台提供了一个应用程序模板。这一步将推出模板，替换应用程序名称、捆绑 ID 和配置文件中的其他属性，以支持正在构建的平台。

   如果您对 Briefcase 提供的模板不满意，可以提供自己的模板。不过，在使用 Briefcase 的默认模板获得更多经验之前，您可能不想这样做。

2. 它 ** 下载并安装了一个支持包 **。公文包所采用的打包方法被描述为 “最简单可行的方法”–它将一个完整、独立的 Python 解释器作为其构建的每个应用程序的一部分。这种打包方式在空间利用上略显不足–如果有 5 个应用程序打包到 Briefcase 中，那么就会有 5 份 Python 解释器副本。但是，这种方法保证了每个应用程序都是完全独立的，使用的 Python 都是已知的能与应用程序一起工作的特定版本。

   同样，Briefcase 为每个平台提供了默认的支持包；如果需要，您可以提供自己的支持包，并将其作为构建过程的一部分。如果您需要启用 Python 解释器中的特定选项，或者如果您想从标准库中剥离运行时不需要的模块，您可能需要这样做。

   公文包维护支持包的本地缓存，因此一旦您下载了特定的支持包，该缓存副本将用于未来的构建。

3. 它 ** 安装应用程序要求 **。您的应用程序可以指定运行时所需的任何第三方模块。这些模块将使用 `pip` 安装到应用程序的安装程序中。
4. 它 ** 安装你的应用程序代码 **。您的应用程序将有自己的代码和资源（如运行时需要的图像）；这些文件会被复制到安装程序中。
5. 最后，它还会添加安装程序本身所需的其他资源。这包括需要附加到最终应用程序的图标和闪屏图像等。

完成上述操作后，如果查看项目目录，就会发现一个与平台（” macOS”、” linux “或 “windows”）相对应的目录，其中包含附加文件。这就是应用程序的特定平台打包配置。

2.4.2 构建您的应用程序

现在可以编译应用程序了。此步骤将执行二进制编译，这是应用程序在目标平台上可执行所必需的。

macOS

Linux

Windows

```
(beeware-venv) $ briefcase build

[helloworld] Adhoc signing app...
...
Signing build/helloworld/macos/app/Hello World.app
------------------------------------------------------------ 100.0% • 00:07

[helloworld] Built build/helloworld/macos/app/Hello World.app
```

在 macOS 上，” *build*“ 命令不需要 * 编译 * 任何内容，但需要对二进制文件的内容进行签名，以便执行。这种签名是一种 * 临时 * 签名–它只能在 * 你 * 的机器上工作；如果你想将应用程序发布给其他人，就需要提供完整的签名。

```
(beeware-venv) $ briefcase build

[helloworld] Finalizing application configuration...
Targeting ubuntu:jammy (Vendor base debian)
Determining glibc version... done

Targeting glibc 2.35
Targeting Python3.10

[helloworld] Building application...
Build bootstrap binary...
make: Entering directory '/home/brutus/beeware-tutorial/helloworld/build/linux/ubuntu/
↪jammy/bootstrap'
```

(续下页)

(接上页)

```
...
make: Leaving directory '/home/brutus/beeware-tutorial/helloworld/build/linux/ubuntu/
↪jammy/bootstrap'
Building bootstrap binary... done
Installing license... done
Installing changelog... done
Installing man page... done
Update file permissions...
...
Updating file permissions... done
Stripping binary... done

[helloworld] Built build/helloworld/linux/ubuntu/jammy/helloworld-0.0.1/usr/bin/
↪helloworld
```

此步骤完成后，*build*‘文件夹将包含一个 *'helloworld-0.0.1*‘文件夹，其中包含一个 *Linux '*/usr“ 文件系统的镜像。这个文件系统镜像将包含一个 “bin“文件夹，里面有一个 “helloworld“二进制文件，以及支持二进制文件所需的 “lib“和 “share“文件夹。

```
(beeware-venv) C:\...>briefcase build
Setting stub app details... done

[helloworld] Built build\helloworld\windows\app\src\Hello World.exe
```

在 Windows 上，`build` 命令不需要编译 * 任何东西，但它确实需要写入一些元数据，以便应用程序知道它的名称、版本等。

触发杀毒软件

由于这些元数据是在 `create` 命令中直接写入从模板推出的预编译二进制文件的，因此可能会触发机器上运行的杀毒软件，从而阻止元数据的写入。在这种情况下，请指示杀毒软件允许工具（名为 `rcedit-x64.exe`）运行，并重新运行上述命令。

2.4.3 运行应用程序

现在您可以使用公文包运行应用程序了：

macOS

Linux

Windows

```
(beeware-venv) $ briefcase run

[helloworld] Starting app...
=============================================================================
Configuring isolated Python...
Pre-initializing Python runtime...
PythonHome: /Users/brutus/beeware-tutorial/helloworld/macOS/app/Hello World/Hello␣
↪World.app/Contents/Resources/support/python-stdlib
PYTHONPATH:
- /Users/brutus/beeware-tutorial/helloworld/macOS/app/Hello World/Hello World.app/
↪Contents/Resources/support/python311.zip
```

(续下页)

(接上页)

```
- /Users/brutus/beeware-tutorial/helloworld/macOS/app/Hello World/Hello World.app/
↪Contents/Resources/support/python-stdlib
- /Users/brutus/beeware-tutorial/helloworld/macOS/app/Hello World/Hello World.app/
↪Contents/Resources/support/python-stdlib/lib-dynload
- /Users/brutus/beeware-tutorial/helloworld/macOS/app/Hello World/Hello World.app/
↪Contents/Resources/app_packages
- /Users/brutus/beeware-tutorial/helloworld/macOS/app/Hello World/Hello World.app/
↪Contents/Resources/app
Configure argc/argv...
Initializing Python runtime...
Installing Python NSLog handler...
Running app module: helloworld
--------------------------------------------------------------------------------
```

```
(beeware-venv) $ briefcase run

[helloworld] Finalizing application configuration...
Targeting ubuntu:jammy (Vendor base debian)
Determining glibc version... done

Targeting glibc 2.35
Targeting Python3.10

[helloworld] Starting app...
================================================================================
Install path: /home/brutus/beeware-tutorial/helloworld/build/helloworld/linux/ubuntu/
↪jammy/helloworld-0.0.1/usr
Pre-initializing Python runtime...
PYTHONPATH:
- /usr/lib/python3.10
- /usr/lib/python3.10/lib-dynload
- /home/brutus/beeware-tutorial/helloworld/build/helloworld/linux/ubuntu/jammy/
↪helloworld-0.0.1/usr/lib/helloworld/app
- /home/brutus/beeware-tutorial/helloworld/build/helloworld/linux/ubuntu/jammy/
↪helloworld-0.0.1/usr/lib/helloworld/app_packages
Configure argc/argv...
Initializing Python runtime...
Running app module: helloworld
--------------------------------------------------------------------------------
```

```
(beeware-venv) C:\...>briefcase run

[helloworld] Starting app...

================================================================================
Log started: 2023-04-23 04:47:45Z
PreInitializing Python runtime...
PythonHome: C:\Users\brutus\beeware-tutorial\helloworld\windows\app\Hello World\src
PYTHONPATH:
- C:\Users\brutus\beeware-tutorial\helloworld\windows\app\Hello World\src\python39.zip
- C:\Users\brutus\beeware-tutorial\helloworld\windows\app\Hello World\src
- C:\Users\brutus\beeware-tutorial\helloworld\windows\app\Hello World\src\app_packages
- C:\Users\brutus\beeware-tutorial\helloworld\windows\app\Hello World\src\app
Configure argc/argv...
Initializing Python runtime...
Running app module: helloworld
```

(续下页)

(接上页)

```
------------------------------------------------------------------------
```

这将使用 `build` 命令的输出开始运行本地应用程序。

你可能会注意到应用程序运行时的外观有一些细微差别。例如，操作系统显示的图标和名称可能与在开发者模式下运行时略有不同。这也是因为您使用的是打包的应用程序，而不仅仅是运行 Python 代码。从操作系统的角度来看，您现在运行的是“一个应用程序”，而不是“一个 Python 程序”，这也反映在应用程序的显示方式上。

2.4.4 创建安装程序

现在，您可以使用 `package` 命令打包应用程序，以便发布。打包命令会执行将脚手架项目转换为最终可发布产品所需的编译工作。根据平台的不同，这可能涉及编译安装程序、执行代码签名或执行其他发布前任务。

macOS

Linux

Windows

```
(beeware-venv) $ briefcase package --adhoc-sign

[helloworld] Signing app...

************************************************************************
** WARNING: Signing with an ad-hoc identity                           **
************************************************************************

    This app is being signed with an ad-hoc identity. The resulting
    app will run on this computer, but will not run on anyone else's
    computer.

    To generate an app that can be distributed to others, you must
    obtain an application distribution certificate from Apple, and
    select the developer identity associated with that certificate
    when running 'briefcase package'.

************************************************************************

Signing app with ad-hoc identity...
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 100.0% • 00:07

[helloworld] Building DMG...
Building dist/Hello World-0.0.1.dmg

[helloworld] Packaged dist/Hello World-0.0.1.dmg
```

dist “文件夹将包含一个名为“Hello World-0.0.1.dmg “的文件。如果在 Finder 中找到该文件，并双击其图标，就会加载 DMG，从而获得 Hello World 应用程序的副本，并链接到“应用程序“文件夹以方便安装。将应用程序文件拖入“应用程序”，就安装好了应用程序。将 DMG 文件发送给朋友，他们应该也能完成同样的操作。

在本例中，我们使用了”–adhoc-sign “选项–也就是说，我们使用 *ad hoc* 凭据（仅在你的机器上有效的临时凭据）来签署我们的应用程序。我们这样做是为了让教程简单明了。设置代码签名身份有点麻烦，而且只有当你打算将应用程序发布给他人时才 * 需要。如果我们要发布一个真正的应用程序供他人使用，我们就需要指定真实的凭据。

当您准备发布真正的应用程序时，请查看关于“设置 macOS 代码签名身份 <https://briefcase.readthedocs.io/en/latest/how-to/code-signing/macOS.html>‘__”的公文包操作指南。

软件包步骤的输出会因 Linux 发行版的不同而略有不同。如果你使用的是 Debian 衍生发行版，你会看到:

```
(beeware-venv) $ briefcase package

[helloworld] Finalizing application configuration...
Targeting ubuntu:jammy (Vendor base debian)
Determining glibc version... done

Targeting glibc 2.35
Targeting Python3.10

[helloworld] Building .deb package...
Write Debian package control file... done

dpkg-deb: building package 'helloworld' in 'helloworld-0.0.1.deb'.
Building Debian package... done

[helloworld] Packaged dist/helloworld_0.0.1-1~ubuntu-jammy_amd64.deb
```

dist “文件夹将包含生成的”.deb “文件。

如果你使用的是基于 RHEL 的发行版，你就会看到:

```
(beeware-venv) $ briefcase package

[helloworld] Finalizing application configuration...
Targeting fedora:36 (Vendor base rhel)
Determining glibc version... done

Targeting glibc 2.35
Targeting Python3.10

[helloworld] Building .rpm package...
Generating rpmbuild layout... done

Write RPM spec file... done

Building source archive... done

Executing(%prep): /bin/sh -e /var/tmp/rpm-tmp.Kav9H7
+ umask 022
...
+ exit 0
Building RPM package... done

[helloworld] Packaged dist/helloworld-0.0.1-1.fc36.x86_64.rpm
```

dist “文件夹将包含生成的”.rpm “文件。

如果你使用的是基于 Arch 的发行版，你就会看到:

```
(beeware-venv) $ briefcase package

[helloworld] Finalizing application configuration...
Targeting arch:rolling (Vendor base arch)
Determining glibc version... done
```

(续下页)

(接上页)

```
Targeting glibc 2.37
Targeting Python3.10

[helloworld] Building .pkg.tar.zst package...
...
Building Arch package... done

[helloworld] Packaged dist/helloworld-0.0.1-1-x86_64.pkg.tar.zst
```

dist “文件夹将包含生成的”.pkg.tar.zst “文件。

目前不支持其他 Linux 发行版的打包。

如果你想为你正在使用的 Linux 发行版之外的其他发行版构建软件包，Briefcase 也能提供帮助，但你需要安装 Docker。

Docker Engine <https://docs.docker.com/engine/install>‘__ 的官方安装程序适用于一系列 Unix 发行版。请按照您所在平台的说明进行操作；不过，请确保不要在“无根“模式下安装 Docker。

安装好 Docker 后，你就可以启动 Linux 容器了，例如:

```
$ docker run -it ubuntu:22.04
```

将显示 Ubuntu 22.04 Docker 容器中的 Unix 提示（类似于 `root@84444e31cff9:/#`）。键入 Ctrl-D 退出 Docker 并返回本地 shell。

一旦安装了 Docker，只要将 Docker 镜像作为参数传递，就可以使用 Briefcase 为 Briefcase 支持的任何 Linux 发行版构建软件包。例如，要为 Ubuntu 22.04 (Jammy) 构建一个 DEB 包，无论你使用的是哪种操作系统，你都可以运行:

```
$ briefcase package --target ubuntu:jammy
```

这将下载所选操作系统的 Docker 镜像，创建一个能够运行 Briefcase 构建的容器，并在镜像中构建应用程序软件包。完成后，*dist*‘文件夹将包含目标 Linux 发行版的软件包。

```
(beeware-venv) C:\...>briefcase package

************************************************************************
** WARNING: No signing identity provided                              **
************************************************************************

    Briefcase will not sign the app. To provide a signing identity,
    use the `--identity` option; or, to explicitly disable signing,
    use `--adhoc-sign`.

************************************************************************

[helloworld] Building MSI...
Compiling application manifest...
Compiling... done

Compiling application installer...
helloworld.wxs
helloworld-manifest.wxs
Compiling... done

Linking application installer...
Linking... done
```

(续下页)

(接上页)

```
[helloworld] Packaged dist\Hello_World-0.0.1.msi
```

在本例中，我们使用了”–adhoc-sign “选项–也就是说，我们使用 *ad hoc* 凭据（仅在你的机器上有效的临时凭据）来签署我们的应用程序。我们这样做是为了让教程简单明了。设置代码签名身份有点麻烦，而且只有当你打算将应用程序发布给他人时才 * 需要。如果我们要发布一个真正的应用程序供他人使用，我们就需要指定真实的凭据。

当您准备发布真正的应用程序时，请查看关于“设置 macOS 代码签名身份 <https://briefcase.readthedocs.io/en/latest/how-to/code-signing/macOS.html>‘__” 的公文包操作指南。

此步骤完成后，” dist “文件夹中将包含一个名为“Hello_World-0.0.1.msi “的文件。如果双击运行该安装程序，就会进入熟悉的 Windows 安装过程。安装完成后，开始菜单中将出现“Hello World “条目。

2.4.5 下一步

现在，我们已将应用程序打包发布到桌面平台上。但是，当我们需要更新应用程序中的代码时该怎么办？如何将这些更新添加到打包的应用程序中？请访问*Tutorial 4* 了解详情…

2.5 教程 4 - 更新应用程序

在上一教程中，我们将应用程序打包为本地应用程序。如果您面对的是一个真实世界的应用程序，这并不是故事的结束–您可能会进行一些测试，发现一些问题，并需要进行一些修改。即使您的应用程序是完美的，您最终还是希望发布改进后的第二版应用程序。

那么，在更改代码时如何更新已安装的应用程序?

2.5.1 更新应用程序代码

目前，我们的应用程序会在您按下按钮时打印到控制台。但是，图形用户界面应用程序实际上不应该使用控制台进行输出。它们需要使用对话框与用户交流。

让我们添加一个对话框来说“你好”，而不是写入控制台。修改 `say_hello` 回调，使它看起来像这样：:

```
def say_hello(self, widget):
    self.main_window.info_dialog(
        f"Hello, {self.name_input.value}",
        "Hi there!"
    )
```

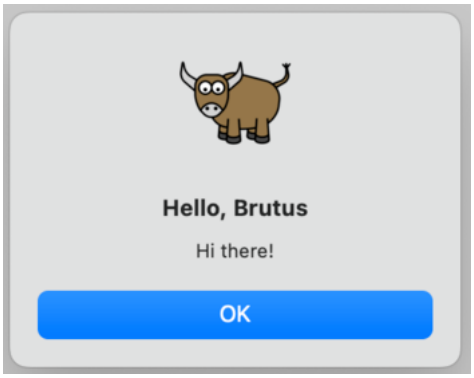
这将指示 Toga 在按下按钮时打开一个模态对话框。

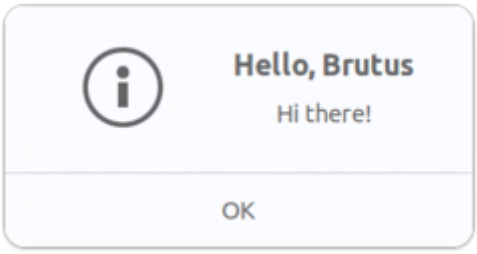
如果运行 `briefcase dev`，输入名称并按下按钮，就会看到新的对话框：

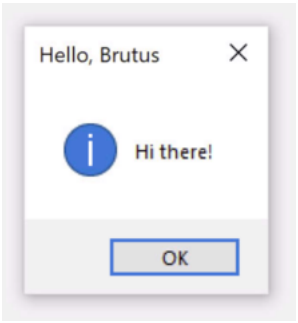
macOS

Linux

Windows







但是，如果运行 “briefcase run“，对话框就不会出现。

为什么会这样？简易开发包 “通过就地运行您的代码来运行–它试图为您的代码提供尽可能逼真的运行环境，但并不提供或使用任何平台基础架构来将您的代码包装成应用程序。打包应用程序的部分过程包括将代码复制到应用程序捆绑包中，目前，您的应用程序中仍有旧代码。

因此，我们需要告诉公文包更新应用程序，复制新版本的代码。我们可以删除旧的平台目录，然后从头开始。不过，Briefcase 提供了一种更简单的方法–您可以更新现有捆绑应用程序的代码：

macOS

Linux

Windows

```
(beeware-venv) $ briefcase update

[helloworld] Updating application code...
Installing src/helloworld... done

[helloworld] Removing unneeded app content...
Removing unneeded app bundle content... done

[helloworld] Application updated.
```

```
(beeware-venv) $ briefcase update

[helloworld] Updating application code...
Installing src/helloworld... done

[helloworld] Removing unneeded app content...
Removing unneeded app bundle content... done

[helloworld] Removing unneeded app content...
Removing unneeded app bundle content... done

[helloworld] Application updated.
```

```
(beeware-venv) C:\...>briefcase update

[helloworld] Updating application code...
Installing src/helloworld... done

[helloworld] Removing unneeded app content...
Removing unneeded app bundle content... done

[helloworld] Application updated.
```

如果 Briefcase 找不到脚手架模板，它会自动调用 `create` 生成一个新的脚手架。

更新了安装程序代码后，我们就可以运行 `briefcase build` 来重新编译应用程序，运行 `briefcase run` 来运行更新后的应用程序，并运行 `briefcase package` 来重新打包应用程序以便分发。

(macOS 用户请记住，正如*Tutorial 3* 中所述，我们建议在运行 `briefcase package` 时使用 `--adhoc-sign` 标志，以避免设置代码签名身份的复杂性，并使教程尽可能简单)。

2.5.2 更新和运行一步到位

如果要快速迭代代码更改，很可能需要更改代码、更新应用程序并立即重新运行应用程序。对于大多数情况来说，开发人员模式 (`briefcase dev`) 是进行这种快速迭代的最简单方法；但是，如果您要测试应用程序作为本地二进制文件运行的方式，或查找一个只有在应用程序打包时才会出现的错误，您可能需要反复调用 `briefcase run`。为了简化更新和运行打包应用程序的过程，Briefcase 提供了支持这种使用模式的快捷方式 - `run`` 命令上的 ``-u`` （或 ``--update`）选项。

让我们尝试做另一个改动。您可能已经注意到，如果您不在文本输入框中输入姓名，对话框就会显示 “Hello, “。让我们再次修改 `say_hello` 函数，以处理这种边缘情况。

在文件顶部，在导入和 “类 HelloWorld “定义之间，添加一个实用程序方法，以根据所提供的名称值生成适当的问候语：:

```
def greeting(name):
    if name:
        return f"Hello, {name}"
    else:
        return "Hello, stranger"
```

然后，修改 `say_hello` 回调，以使用这个新的实用程序方法：:

```
def say_hello(self, widget):
    self.main_window.info_dialog(
        greeting(self.name_input.value),
```

(续下页)

(接上页)

```
        "Hi there!",
    )
```

在开发模式下运行应用程序（使用 `briefcase dev`），以确认新逻辑是否有效；然后使用一条命令更新、构建和运行应用程序：

macOS

Linux

Windows

```
(beeware-venv) $ briefcase run -u

[helloworld] Updating application code...
Installing src/helloworld... done

[helloworld] Removing unneeded app content...
Removing unneeded app bundle content... done

[helloworld] Application updated.

[helloworld] Building application...
...

[helloworld] Built build/helloworld/macos/app/Hello World.app

[helloworld] Starting app...
```

```
(beeware-venv) $ briefcase run -u

[helloworld] Finalizing application configuration...
Targeting ubuntu:jammy (Vendor base debian)
Determining glibc version... done

Targeting glibc 2.35
Targeting Python3.10

[helloworld] Updating application code...
Installing src/helloworld... done

[helloworld] Removing unneeded app content...
Removing unneeded app bundle content... done

[helloworld] Application updated.

[helloworld] Building application...
...

[helloworld] Built build/helloworld/linux/ubuntu/jammy/helloworld-0.0.1/usr/bin/
↪helloworld

[helloworld] Starting app...
```

```
(beeware-venv) C:\...>briefcase run -u

[helloworld] Updating application code...
```

(续下页)

(接上页)

```
Installing src/helloworld... done

[helloworld] Removing unneeded app content...
Removing unneeded app bundle content... done

[helloworld] Application updated.

[helloworld] Starting app...
```

package 命令也接受 `-u` 参数，因此如果你修改了应用程序代码并希望立即重新打包，可以运行 `briefcase package -u`。

2.5.3 下一步

现在，我们的应用程序已经打包，可以在桌面平台上发布，我们也可以更新应用程序中的代码。

但是移动应用程序呢？在*Tutorial 5* 中，我们将把应用程序转换为移动应用程序，并将其部署到设备模拟器和手机上。

2.6 教程 5 - 移动操作

到目前为止，我们一直在台式机上运行和测试应用程序。不过，BeeWare 同样支持移动平台，我们编写的应用程序也可以部署到您的移动设备上！

iOS　iOS 应用程序只能在 macOS 上编译。

Let's build our app for iOS!

安卓　安卓应用程序可在 macOS、Windows 或 Linux 上编译。

Let's build our app for Android!

2.6.1 教程 5 - 移动：iOS

要编译 iOS 应用程序，我们需要使用 Xcode，它可从 MacOS 应用商店 <https://apps.apple.com/au/app/xcode/id497799835?mt=12>‘__ 免费获取。

安装好 Xcode 后，我们就可以将应用程序作为 iOS 应用程序进行部署。

将应用程序部署到 iOS 的过程与部署桌面应用程序的过程非常相似。首先，运行 `create` 命令，但这次我们指定要创建一个 iOS 应用程序：

```
(beeware-venv) $ briefcase create iOS

[helloworld] Generating application template...
Using app template: https://github.com/beeware/briefcase-iOS-Xcode-template.git,␣
↪branch v0.3.14
...

[helloworld] Installing support package...
...

[helloworld] Installing application code...
```

(续下页)

(接上页)

```
Installing src/helloworld... done

[helloworld] Installing requirements...
...

[helloworld] Installing application resources...
...

[helloworld] Removing unneeded app content...
...

[helloworld] Created build/helloworld/ios/xcode
```

完成此操作后，我们将拥有一个包含 Xcode 项目、支持库和应用程序所需应用程序代码的“build/helloworld/ios/xcode“目录。

然后，您可以使用 `briefcase build iOS` 来编译应用程序：

```
(beeware-venv) $ briefcase build iOS

[helloworld] Updating app metadata...
Setting main module... done

[helloworld] Building Xcode project...
...
Building... done

[helloworld] Built build/helloworld/ios/xcode/build/Debug-iphonesimulator/Hello World.
↪app
```

现在我们可以使用 `briefcase run iOS` 运行应用程序了。系统会提示你选择要编译的设备；如果你安装了多个 iOS SDK 版本的模拟器，系统可能还会询问你要针对哪个 iOS 版本。所显示的选项可能与此输出中显示的选项不同，至少设备列表可能不同。就我们而言，选择哪个模拟器并不重要。

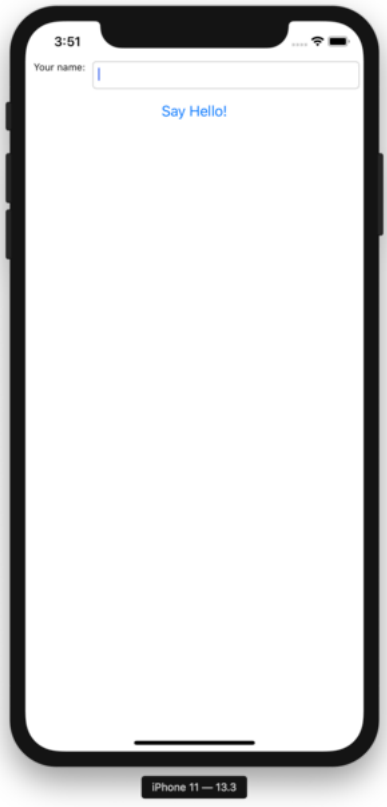
```
(beeware-venv) $ briefcase run iOS

Select simulator device:

   1) iPad (10th generation)
   2) iPad Air (5th generation)
   3) iPad Pro (11-inch) (4th generation)
   4) iPad Pro (12.9-inch) (6th generation)
   5) iPad mini (6th generation)
   6) iPhone 14
   7) iPhone 14 Plus
   8) iPhone 14 Pro
   9) iPhone 14 Pro Max
   10) iPhone SE (3rd generation)

>  10

In the future, you could specify this device by running:

    $ briefcase run iOS -d "iPhone SE (3rd generation)::iOS 16.2"

or:
```

(续下页)

(接上页)

```
    $ briefcase run iOS -d 2614A2DD-574F-4C1F-9F1E-478F32DE282E

[helloworld] Starting app on an iPhone SE (3rd generation) running iOS 16.2 (device␣
↪UDID 2614A2DD-574F-4C1F-9F1E-478F32DE282E)
Booting simulator... done
Opening simulator... done

[helloworld] Installing app...
Uninstalling any existing app version... done
Installing new app version... done

[helloworld] Starting app...
Launching app... done

[helloworld] Following simulator log output (type CTRL-C to stop log)...
================================================================================
...
```

这将启动 iOS 模拟器，安装你的应用程序并启动它。你应该能看到模拟器启动，并最终打开你的 iOS 应用程序：



如果事先知道要使用哪个 iOS 模拟器，可以通过提供 `-d`` （或 ``--device`）选项告诉 Briefcase 使用该模拟器。使用创建应用程序时选择的设备名称，运行：

```
$ briefcase run iOS -d "iPhone SE (3rd generation)"
```

如果你有多个 iOS 版本，Briefcase 会选择最高的 iOS 版本；如果你想选择某个 iOS 版本，你可以告诉它使用

该特定版本：

```
$ briefcase run iOS -d "iPhone SE (3rd generation)::iOS 15.5"
```

或者，您也可以命名一个特定的设备 UDID：

```
$ briefcase run iOS -d 2614A2DD-574F-4C1F-9F1E-478F32DE282E
```

下一步

我们现在已经在手机上安装了应用程序！还有其他地方可以部署 BeeWare 应用程序吗？请参考*Tutorial 6* 了解更多…

2.6.2 教程 5 - 移动化：安卓

现在，我们要将应用程序部署为 Android 应用程序。

将应用程序部署到 Android 的过程与部署为桌面应用程序的过程非常相似。公文包会处理 Android 依赖项的安装，包括 Android SDK、Android 模拟器和 Java 编译器。

创建并编译 Android 应用程序

首先，运行 `create` 命令。这会下载一个 Android 应用程序模板，并在其中添加您的 Python 代码。

macOS

Linux

Windows

```
(beeware-venv) $ briefcase create android

[helloworld] Generating application template...
Using app template: https://github.com/beeware/briefcase-android-gradle-template.git,␣
↪branch v0.3.14
...

[helloworld] Installing support package...
No support package required.

[helloworld] Installing application code...
Installing src/helloworld... done

[helloworld] Installing requirements...
Writing requirements file... done

[helloworld] Installing application resources...
...

[helloworld] Removing unneeded app content...
Removing unneeded app bundle content... done

[helloworld] Created build/helloworld/android/gradle
```

```
(beeware-venv) $ briefcase create android

[helloworld] Generating application template...
Using app template: https://github.com/beeware/briefcase-android-gradle-template.git,␣
↪branch v0.3.14
...

[helloworld] Installing support package...
No support package required.

[helloworld] Installing application code...
Installing src/helloworld... done

[helloworld] Installing requirements...
Writing requirements file... done

[helloworld] Installing application resources...
...

[helloworld] Removing unneeded app content...
Removing unneeded app bundle content... done

[helloworld] Created build/helloworld/android/gradle
```

```
(beeware-venv) C:\...>briefcase create android

[helloworld] Generating application template...
Using app template: https://github.com/beeware/briefcase-android-gradle-template.git,␣
↪branch v0.3.14
...

[helloworld] Installing support package...
No support package required.

[helloworld] Installing application code...
Installing src/helloworld... done

[helloworld] Installing requirements...
Writing requirements file... done

[helloworld] Installing application resources...
...

[helloworld] Removing unneeded app content...
Removing unneeded app bundle content... done

[helloworld] Created build\helloworld\android\gradle
```

首次运行 `briefcase create android` 时，Briefcase 会下载 Java JDK 和 Android SDK。文件大小和下载时间可能相当长；这可能需要一段时间（10 分钟或更长，取决于您的互联网连接速度）。下载完成后，系统将提示您接受 Google 的 Android SDK 许可证。

完成后，我们的项目中就会有一个 “build/helloworld/android/gradle“目录，其中包含一个带有 Gradle 构建配置的 Android 项目。这个项目将包含你的应用代码，以及一个包含 Python 解释器的支持包。

然后，我们可以使用 Briefcase 的 `build` 命令将其编译为 Android APK 应用程序文件。

macOS

Linux

Windows

```
(beeware-venv) $ briefcase build android

[helloworld] Updating app metadata...
Setting main module... done

[helloworld] Building Android APK...
Starting a Gradle Daemon
...
BUILD SUCCESSFUL in 1m 1s
28 actionable tasks: 17 executed, 11 up-to-date
Building... done

[helloworld] Built build/helloworld/android/gradle/app/build/outputs/apk/debug/app-
↪debug.apk
```

```
(beeware-venv) $ briefcase build android

[helloworld] Updating app metadata...
Setting main module... done

[helloworld] Building Android APK...
Starting a Gradle Daemon
...
BUILD SUCCESSFUL in 1m 1s
28 actionable tasks: 17 executed, 11 up-to-date
Building... done

[helloworld] Built build/helloworld/android/gradle/app/build/outputs/apk/debug/app-
↪debug.apk
```

```
(beeware-venv) C:\...>briefcase build android

[helloworld] Updating app metadata...
Setting main module... done

[helloworld] Building Android APK...
Starting a Gradle Daemon
...
BUILD SUCCESSFUL in 1m 1s
28 actionable tasks: 17 executed, 11 up-to-date
Building... done

[helloworld] Built build\helloworld\android\gradle\app\build\outputs\apk\debug\app-
↪debug.apk
```

Gradle 可能看起来卡住了

在 `briefcase build android` 步骤中，Gradle（Android 平台构建工具）会打印 `CONFIGURING: 100%`，似乎什么也没做。别担心，这不是卡住了，而是在下载更多的 Android SDK 组件。根据您的网络连接速度，这可能还需要 10 分钟（或更长时间）。只有在第一次运行 `build` 时才会出现这种滞后现象；这些工具已被缓存，下次构建时将使用缓存版本。

在虚拟设备上运行应用程序

现在我们可以运行应用程序了。您可以使用 Briefcase 的 `run` 命令在 Android 设备上运行应用程序。让我们从在 Android 模拟器上运行开始。

要运行应用程序，请运行 `briefcase run android`。运行时，系统会提示你可以在哪些设备上运行应用程序。最后一项总是创建新安卓模拟器的选项。

macOS

Linux

Windows

```
(beeware-venv) $ briefcase run android

Select device:

  1) Create a new Android emulator

>
```

```
(beeware-venv) $ briefcase run android

Select device:

  1) Create a new Android emulator

>
```

```
(beeware-venv) C:\...>briefcase run android

Select device:

  1) Create a new Android emulator

>
```

现在我们可以选择想要的设备。选择“创建新的安卓模拟器“选项，并接受默认的设备名称（” beePhone”）。

公文包 `run` 将自动启动虚拟设备。设备启动时，您将看到 Android 徽标：

设备启动完成后，公文包将在设备上安装您的应用程序。您将短暂看到一个启动器屏幕：

然后应用程序就会启动。程序启动时，你会看到一个闪屏：

模拟器没有启动！

安卓模拟器是一个复杂的软件，依赖于许多硬件和操作系统功能，这些功能在旧机器上可能无法使用或启用。如果在启动 Android 模拟器时遇到任何困难，请查阅 Android 开发人员文档中的“要求和建议 <https://developer.android.com/studio/run/emulator#requirements>‘__” 部分。

首次启动应用程序时，它需要将自己解压缩到设备上。这可能需要几秒钟。解压完成后，您将看到桌面应用程序的 Android 版本：

如果看不到应用程序启动，可能需要检查运行 `briefcase run` 的终端，查看是否有错误信息。

今后，如果想在该设备上运行而不使用菜单，可以向 Briefcase 提供模拟器名称，使用 `briefcase run android -d @beePhone` 直接在虚拟设备上运行。

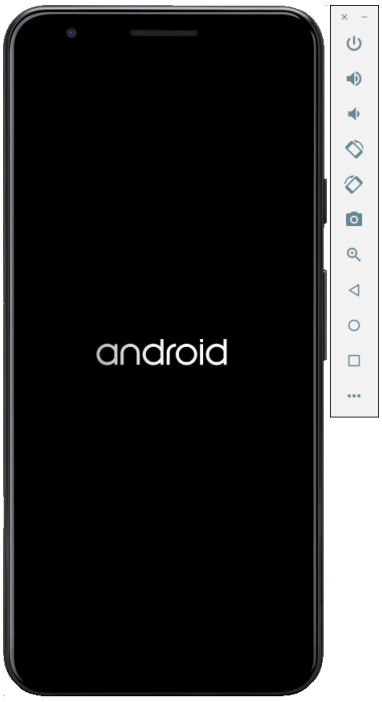


图 1: 启动安卓虚拟设备

图 2: 安卓虚拟设备已完全启动，显示在启动器屏幕上

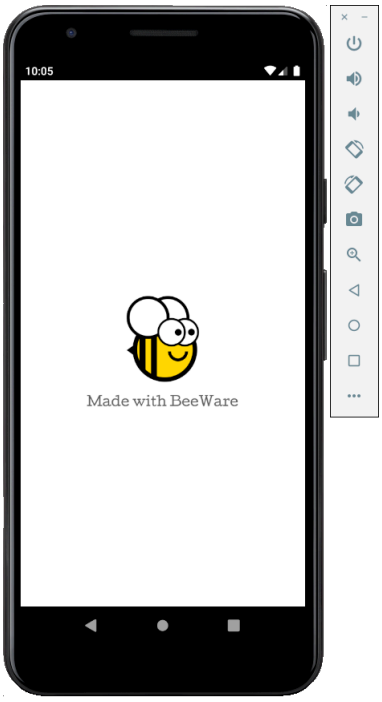


图 3: 应用程序闪屏

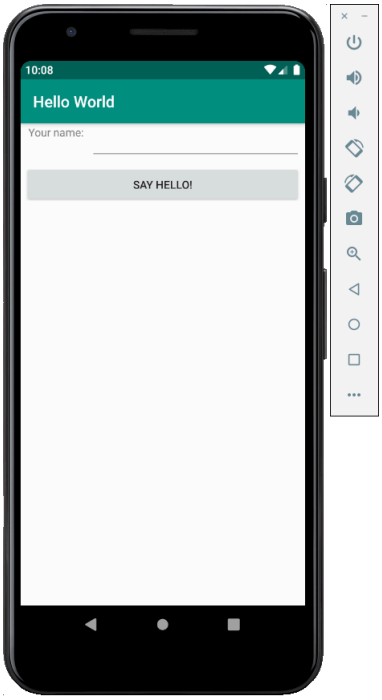


图 4: 演示应用程序全面启动

在实体设备上运行应用程序

如果您有实体安卓手机或平板电脑，可以用 USB 线缆将其连接到电脑，然后使用公文包来定位您的实体设备。

Android 要求您在设备用于开发前做好准备。您需要对设备上的选项进行两项更改：

- 启用开发人员选项
- 启用 USB 调试

有关如何进行这些更改的详细信息，请参阅 Android 开发人员文档 <https://developer.android.com/studio/debug/dev-options#enable>`__。

完成这些步骤后，在运行 `briefcase run android` 时，您的设备应出现在可用设备列表中。

macOS

Linux

Windows

```
(beeware-venv) $ briefcase run android

Select device:

  1) Pixel 3a (94ZZY0LNE8)
  2) @beePhone (emulator)
  3) Create a new Android emulator

>
```

```
(beeware-venv) $ briefcase run android

Select device:

  1) Pixel 3a (94ZZY0LNE8)
  2) @beePhone (emulator)
  3) Create a new Android emulator

>
```

```
(beeware-venv) C:\...>briefcase run android

Select device:

  1) Pixel 3a (94ZZY0LNE8)
  2) @beePhone (emulator)
  3) Create a new Android emulator

>
```

在这里，我们可以在部署列表中看到一个带有序列号的新物理设备–在本例中是 Pixel 3a。将来，如果你想不使用菜单而在该设备上运行，可以向 Briefcase 提供手机序列号（本例中为 `briefcase run android -d 94ZZY0LNE8`）。这将直接在设备上运行，无需提示。

我的设备没有出现！

如果你的设备根本没有出现在这个列表中，要么是你没有启用 USB 调试（要么是设备没有插上电源！）。

如果你的设备出现了，但却被列为“未知设备（未授权开发）”，那么开发者模式尚未正确启用。重新运行“启用开发者选项的步骤 <https://developer.android.com/studio/debug/dev-options#enable>‘__”，并重新运行“briefcase run android”。

下一步

我们现在已经在手机上安装了应用程序！还有其他地方可以部署 BeeWare 应用程序吗？请参考*Tutorial 6* 了解更多…

2.7 教程 6 - 上网！

除了支持移动平台，Toga widget 工具包还支持网络！使用与部署桌面和移动应用程序相同的应用程序接口，您可以将应用程序部署为单页网页应用程序。

概念验证

Toga Web 后端是所有 Toga 后端中最不成熟的一个。它已经足够成熟，可以展示一些功能，但很可能会有漏洞，而且会缺少许多其他平台上可用的小工具。目前，Web 部署应被视为“概念验证”–足以展示可以做什么，但还不足以作为严肃开发的依据。

如果您在教程的这一步遇到问题，可以跳到下一页。

2.7.1 作为网络应用程序部署

作为单页面 Web 应用程序部署的过程遵循相同的熟悉模式–创建应用程序，然后构建应用程序，然后运行它。不过，Briefcase 也有一点小聪明；如果您尝试运行应用程序，而 Briefcase 确定该应用程序尚未针对目标平台创建或构建，那么它会为您执行创建和构建步骤。由于这是我们第一次为网络运行应用程序，我们可以用一条命令执行所有三个步骤：

macOS

Linux

Windows

```
(beeware-venv) $ briefcase run web

[helloworld] Generating application template...
Using app template: https://github.com/beeware/briefcase-web-static-template.git,␣
↪branch v0.3.14
...

[helloworld] Installing support package...
No support package required.

[helloworld] Installing application code...
Installing src/helloworld... done

[helloworld] Installing requirements...
Writing requirements file... done
```

(续下页)

(接上页)

```
[helloworld] Installing application resources...
...

[helloworld] Removing unneeded app content...
Removing unneeded app bundle content... done

[helloworld] Created build/helloworld/web/static

[helloworld] Building web project...
...

[helloworld] Built build/helloworld/web/static/www/index.html

[helloworld] Starting web server...
Web server open on http://127.0.0.1:8080

[helloworld] Web server log output (type CTRL-C to stop log)...
========================================================================
```

```
(beeware-venv) $ briefcase run web

[helloworld] Generating application template...
Using app template: https://github.com/beeware/briefcase-web-static-template.git,␣
↪branch v0.3.14
...

[helloworld] Installing support package...
No support package required.

[helloworld] Installing application code...
Installing src/helloworld... done

[helloworld] Installing requirements...
Writing requirements file... done

[helloworld] Installing application resources...
...

[helloworld] Removing unneeded app content...
Removing unneeded app bundle content... done

[helloworld] Created build/helloworld/web/static

[helloworld] Building web project...
...

[helloworld] Built build/helloworld/web/static/www/index.html

[helloworld] Starting web server...
Web server open on http://127.0.0.1:8080

[helloworld] Web server log output (type CTRL-C to stop log)...
========================================================================
```

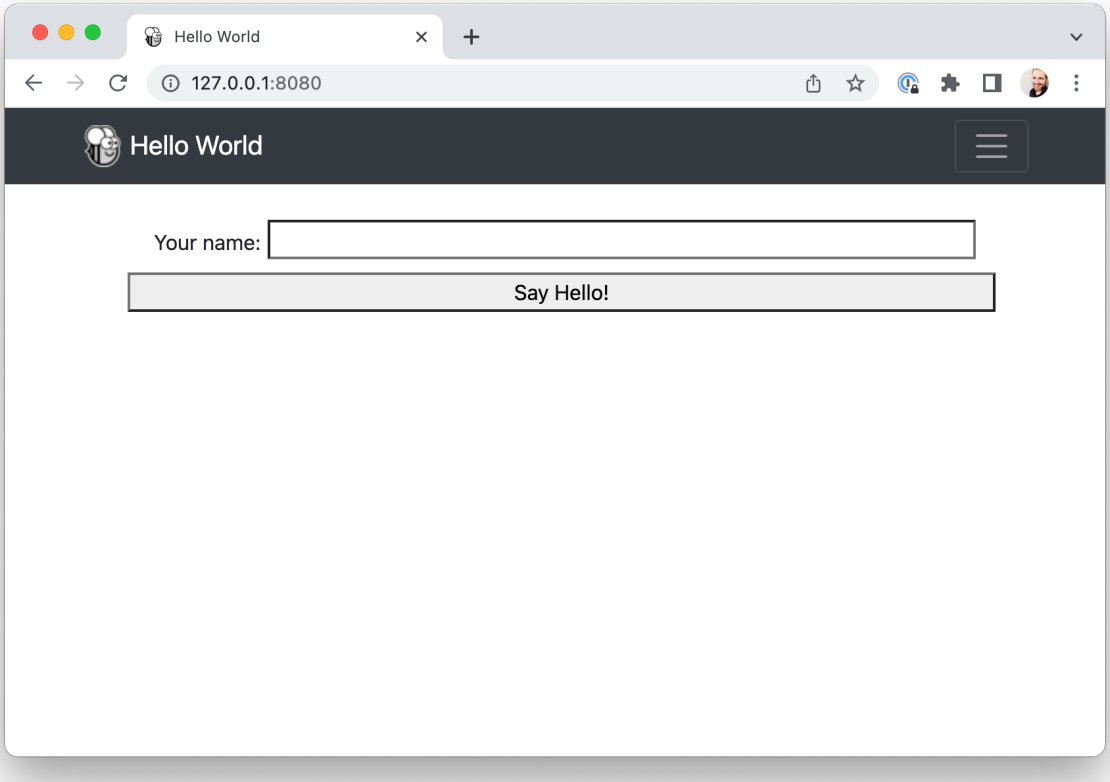
```
(beeware-venv) C:\...>briefcase run web
```

(续下页)

(接上页)

```
[helloworld] Generating application template...
Using app template: https://github.com/beeware/briefcase-web-static-template.git,␣
↪branch v0.3.14
...

[helloworld] Installing support package...
No support package required.

[helloworld] Installing application code...
Installing src/helloworld... done

[helloworld] Installing requirements...
Writing requirements file... done

[helloworld] Installing application resources...
...

[helloworld] Removing unneeded app content...
Removing unneeded app bundle content... done

[helloworld] Created build\helloworld\web\static

[helloworld] Building web project...
...

[helloworld] Built build\helloworld\web\static\www\index.html

[helloworld] Starting web server...
Web server open on http://127.0.0.1:8080

[helloworld] Web server log output (type CTRL-C to stop log)...
================================================================================
```

这将打开一个网页浏览器，指向 http://127.0.0.1:8080：



如果输入您的姓名并点击按钮，就会出现一个对话框。

2.7.2 这个怎么用？

此网络应用程序是一个静态网站–一个 HTML 源页面，包含一些 CSS 和其他资源。公文包启动了一个本地网络服务器来提供该页面，以便您的浏览器可以查看该页面。如果您想将此网页投入生产，可以将 `www` 文件夹中的内容复制到任何可以提供静态内容的网络服务器上。

但当你按下按钮时，你运行的是 Python 代码……这是怎么做到的呢？Toga 使用 PyScript 在浏览器中提供 Python 解释器。公文包将应用程序的代码打包成 PyScript 可以在浏览器中加载的轮子。加载页面时，应用代码会在浏览器中运行，并使用浏览器 DOM 构建用户界面。当您点击按钮时，该按钮会在浏览器中运行事件处理代码。

2.7.3 下一步

虽然我们现在已经在桌面、手机和网络上部署了这个应用程序，但该应用程序相当简单，不涉及任何第三方库。我们可以在应用程序中包含 Python 包索引 (PyPI) 中的库吗？请访问*Tutorial 7* 了解详情…

2.8 教程 7 - 启动这个（第三）派对

到目前为止，我们构建的应用程序只使用了我们自己的代码和 BeeWare 提供的代码。不过，在实际应用中，您很可能需要使用从 Python 软件包索引（PyPI）下载的第三方库。

让我们修改应用程序，加入第三方库。

2.8.1 访问应用程序接口

应用程序需要执行的一项常见任务是向网络应用程序接口发出请求以获取数据，并将数据显示给用户。这是一个玩具应用程序，因此我们没有 * 真正 * 的 API 可供使用，所以我们将使用 {JSON} 占位符 API 作为数据源。

{JSON} 占位符 API 有许多“伪造“的 API 端点，您可以将其用作测试数据。其中一个 API 是 `/posts/` 端点，它会返回虚假的博客文章。如果在浏览器中打开 `https://jsonplaceholder.typicode.com/posts/42`，就会得到一个描述单篇博文的 JSON 有效载荷–ID 为 42 的博文的一些 Lorum ipsum 内容。

Python 标准库包含访问 API 所需的所有工具。然而，内置的 API 是非常低级的。它们很好地实现了 HTTP 协议，但需要用户管理大量低级细节，如 URL 重定向、会话、身份验证和有效负载编码。作为一名“普通浏览器用户”，你可能已经习惯了将这些细节视为理所当然，因为浏览器会为你管理这些细节。

因此，人们开发了第三方库来封装内置的 API，并提供更简单的 API，使其更符合日常的浏览器体验。我们将使用其中一个库来访问 {JSON} 占位符 API - 一个名为 httpx 的库。

让我们在应用程序中添加一个 `httpx` API 调用。在 `app.py` 顶部添加导入，以导入 `httpx`::

```
import httpx
```

然后修改 `say_hello()` 回调，使其看起来像这样::

```
def say_hello(self, widget):
    with httpx.Client() as client:
        response = client.get("https://jsonplaceholder.typicode.com/posts/42")

    payload = response.json()

    self.main_window.info_dialog(
        greeting(self.name_input.value),
        payload["body"],
    )
```

这将更改 `say_hello()` 回调，使它在被调用时，会:

- 在 JSON 占位符 API 上发出 GET 请求，以获取职位 42；
- 将响应解码为 JSON 格式；
- 提取帖子正文；以及
- 将该帖子的正文作为对话框的文本。

让我们在公文包开发者模式下运行更新后的应用程序，检查我们的更改是否有效。

macOS

Linux

Windows

```
(beeware-venv) $ briefcase dev
Traceback (most recent call last):
File ".../venv/bin/briefcase", line 5, in <module>
    from briefcase.__main__ import main
File ".../venv/lib/python3.9/site-packages/briefcase/__main__.py", line 3, in <module>
    from .cmdline import parse_cmdline
File ".../venv/lib/python3.9/site-packages/briefcase/cmdline.py", line 6, in <module>
    from briefcase.commands import DevCommand, NewCommand, UpgradeCommand
File ".../venv/lib/python3.9/site-packages/briefcase/commands/__init__.py", line 1,␣
↪in <module>
    from .build import BuildCommand  # noqa
File ".../venv/lib/python3.9/site-packages/briefcase/commands/build.py", line 5, in
↪<module>
    from .base import BaseCommand, full_options
File ".../venv/lib/python3.9/site-packages/briefcase/commands/base.py", line 14, in
↪<module>
    import httpx
ModuleNotFoundError: No module named 'httpx'
```

```
(beeware-venv) $ briefcase dev
Traceback (most recent call last):
File ".../venv/bin/briefcase", line 5, in <module>
    from briefcase.__main__ import main
File ".../venv/lib/python3.9/site-packages/briefcase/__main__.py", line 3, in <module>
    from .cmdline import parse_cmdline
File ".../venv/lib/python3.9/site-packages/briefcase/cmdline.py", line 6, in <module>
    from briefcase.commands import DevCommand, NewCommand, UpgradeCommand
File ".../venv/lib/python3.9/site-packages/briefcase/commands/__init__.py", line 1,␣
↪in <module>
    from .build import BuildCommand  # noqa
File ".../venv/lib/python3.9/site-packages/briefcase/commands/build.py", line 5, in
↪<module>
    from .base import BaseCommand, full_options
File ".../venv/lib/python3.9/site-packages/briefcase/commands/base.py", line 14, in
↪<module>
    import httpx
ModuleNotFoundError: No module named 'httpx'
```

```
(beeware-venv) C:\...>briefcase dev
Traceback (most recent call last):
File "...\venv\bin\briefcase", line 5, in <module>
    from briefcase.__main__ import main
File "...\venv\lib\python3.9\site-packages\briefcase\__main__.py", line 3, in <module>
    from .cmdline import parse_cmdline
File "...\venv\lib\python3.9\site-packages\briefcase\cmdline.py", line 6, in <module>
    from briefcase.commands import DevCommand, NewCommand, UpgradeCommand
File "...\venv\lib\python3.9\site-packages\briefcase\commands\__init__.py", line 1,␣
↪in <module>
    from .build import BuildCommand  # noqa
File "...\venv\lib\python3.9\site-packages\briefcase\commands\build.py", line 5, in
↪<module>
    from .base import BaseCommand, full_options
File "...\venv\lib\python3.9\site-packages\briefcase\commands\base.py", line 14, in
↪<module>
    import httpx
ModuleNotFoundError: No module named 'httpx'
```

发生了什么？我们已经将 `httpx` 添加到我们的代码 * 中，但我们还没有将它添加到我们的开发虚拟环境中。我们可以用 `pip` 安装 `httpx`，然后重新运行 `briefcase dev`：

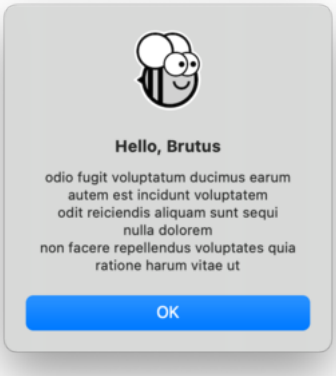
macOS

Linux

Windows

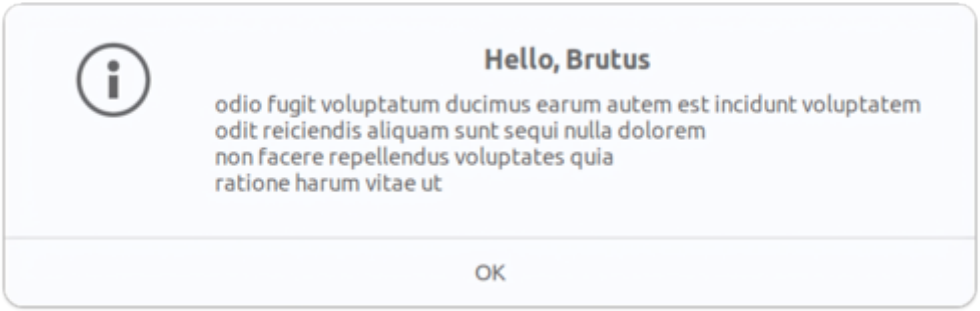
```
(beeware-venv) $ python -m pip install httpx
(beeware-venv) $ briefcase dev
```

输入名称并按下按钮后，您会看到一个类似的对话框：



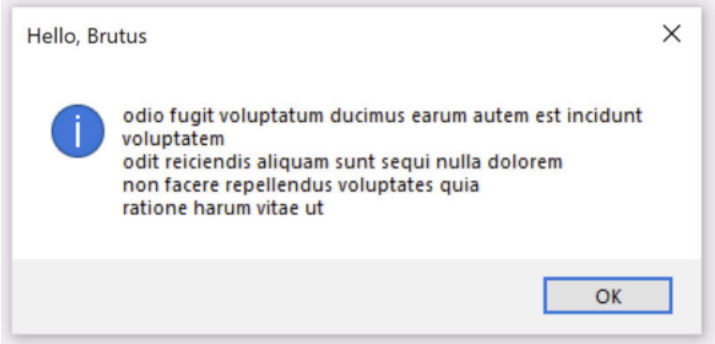

```
(beeware-venv) $ python -m pip install httpx
(beeware-venv) $ briefcase dev
```

输入名称并按下按钮后，您会看到一个类似的对话框：




```
(beeware-venv) C:\...>python -m pip install httpx
(beeware-venv) C:\...>briefcase dev
```

输入名称并按下按钮后，您会看到一个类似的对话框：



现在，我们已经有了一个可正常运行的应用程序，它使用第三方库，以开发模式运行！

2.8.2 运行更新后的应用程序

让我们将更新后的应用程序代码打包为独立应用程序。由于我们对代码进行了修改，因此需要遵循*Tutorial 4*中的相同步骤：

macOS

Linux

Windows

更新打包应用程序中的代码：

```
(beeware-venv) $ briefcase update

[helloworld] Updating application code...
...

[helloworld] Application updated.
```

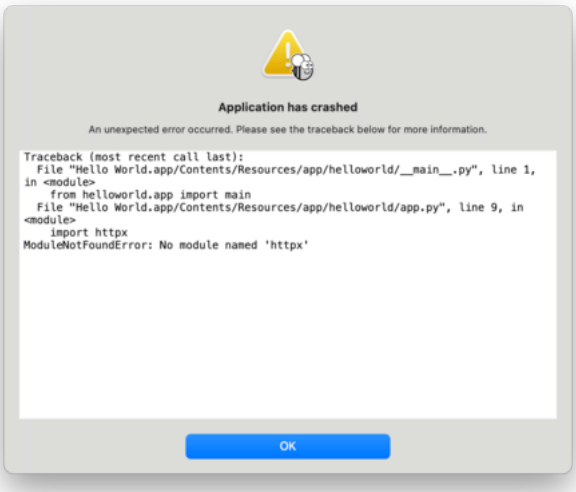
重建应用程序：

```
(beeware-venv) $ briefcase build

[helloworld] Adhoc signing app...
[helloworld] Built build/helloworld/macos/app/Hello World.app
```

最后，运行应用程序：

```
(beeware-venv) $ briefcase run

[helloworld] Starting app...
================================================================================
```

但是，当程序运行时，你会在控制台中看到一个错误，还有一个崩溃对话框：



更新打包应用程序中的代码：

```
(beeware-venv) $ briefcase update

[helloworld] Updating application code...
...

[helloworld] Application updated.
```

重建应用程序：

```
(beeware-venv) $ briefcase build

[helloworld] Finalizing application configuration...
...

[helloworld] Building application...
...

[helloworld] Built build/helloworld/linux/ubuntu/jammy/helloworld-0.0.1/usr/bin/
↪helloworld
```

最后，运行应用程序：

```
(beeware-venv) $ briefcase run

[helloworld] Starting app...
================================================================================
```

但是，当应用程序运行时，您会在控制台中看到一个错误：

```
Traceback (most recent call last):
  File "/usr/lib/python3.10/runpy.py", line 194, in _run_module_as_main
    return _run_code(code, main_globals, None,
  File "/usr/lib/python3.10/runpy.py", line 87, in _run_code
    exec(code, run_globals)
  File "/home/brutus/beeware-tutorial/helloworld/build/linux/ubuntu/jammy/helloworld-
↪0.0.1/usr/app/hello_world/__main__.py", line 1, in <module>
```

(续下页)

(接上页)

```
    from helloworld.app import main
  File "/home/brutus/beeware-tutorial/helloworld/build/linux/ubuntu/jammy/helloworld-
↪0.0.1/usr/app/hello_world/app.py", line 8, in <module>
    import httpx
ModuleNotFoundError: No module named 'httpx'

Unable to start app helloworld.
```

更新打包应用程序中的代码：

```
(beeware-venv) C:\...>briefcase update

[helloworld] Updating application code...
...

[helloworld] Application updated.
```

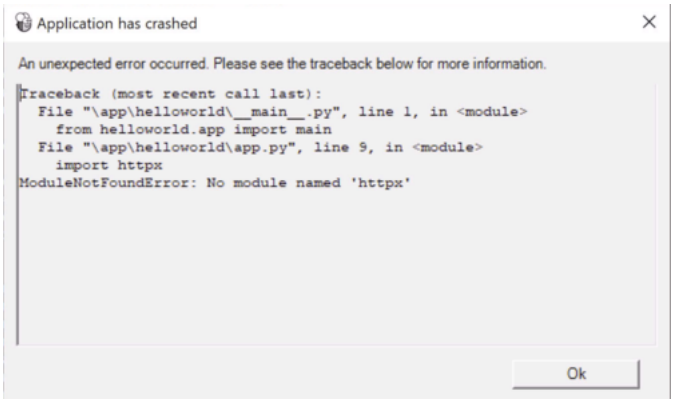
重建应用程序：

```
(beeware-venv) C:\...>briefcase build
...

[helloworld] Built build\helloworld\windows\app\src\Toga Test.exe
```

最后，运行应用程序：

```
(beeware-venv) C:\...>briefcase run

[helloworld] Starting app...
================================================================================
```

但是，当程序运行时，你会在控制台中看到一个错误，还有一个崩溃对话框：



应用程序再次启动失败，因为已经安装了 `httpx` - 但为什么呢？我们不是已经安装了 `httpx` 吗？

我们有，但仅限于开发环境。你的开发环境完全在你的机器本地，只有当你明确激活它时才会启用。虽然公文包有开发模式，但使用公文包的主要原因是打包你的代码，这样你就可以把它交给别人。

要保证别人的 Python 环境包含它所需要的一切，唯一的办法就是构建一个完全隔离的 Python 环境。这意味着有一个完全独立的 Python 安装，和一套完全独立的依赖关系。这就是当你运行 `briefcase build` 时，Briefcase 正在构建的 - 一个隔离的 Python 环境。这也解释了为什么没有安装 `httpx` - 它已经安装在你的开发环境中，但没有安装在打包的应用程序中。

因此，我们需要告诉 Briefcase，我们的应用程序有一个外部依赖关系。

2.8.3 更新依赖项

在应用程序的根目录中，有一个名为 pyproject.toml 的文件。该文件包含您最初运行 briefcase new 时提供的所有应用程序配置详细信息。

pyproject.toml “分为多个部分，其中一部分描述了应用程序的设置：：

```
[tool.briefcase.app.helloworld]
formal_name = "Hello World"
description = "A Tutorial app"
long_description = """More details about the app should go here.
"""
sources = ["src/helloworld"]
requires = []
```

requires “选项描述了应用程序的依赖关系。它是一个字符串列表，其中指定了您希望应用程序包含的库（以及可选的版本）。

修改 requires 设置为：：

```
requires = [
    "httpx",
]
```

通过添加此设置，我们告诉 Briefcase “当你构建我的应用程序时，运行 pip install httpx 到应用程序捆绑包中”。任何可以合法输入到 pip install 的内容都可以在这里使用–因此，你可以指定：

- 特定的库版本（例如，"httpx===0.19.0"）；
- 一系列库版本（例如，"httpx>=0.19"）；
- 指向 git 仓库的路径（例如，"git+https://github.com/encode/httpx"）；或
- 本地文件路径（不过需要注意的是：如果你把代码交给别人，这个路径很可能不存在于他们的机器上!）。

在 pyproject.toml 中的更下面部分，你会注意到与操作系统相关的其他部分，如 [tool.briefcase.app.helloworld.macOS] 和 [tool.briefcase.app.helloworld.windows]。这些部分 ** 也有一个 requires 设置。这些设置允许你定义额外的特定平台依赖关系，例如，如果你需要一个特定平台的库来处理应用程序的某些方面，你可以在特定平台的 requires 部分中指定该库，而该设置将仅用于该平台。你会注意到，所有的 toga 库都是在特定平台的 requires 部分中指定的，这是因为显示用户界面所需的库都是特定平台的。

在我们的例子中，我们希望 httpx 安装在所有平台上，因此使用了应用程序级的 requires 设置。应用程序级的依赖项始终会被安装；特定平台的依赖项会在应用程序级的依赖项之外 * 安装。

某些二进制软件包可能不可用

在桌面平台（macOS、Windows、Linux）上，任何可安装的 “pip “都可以添加到您的需求中。在移动和网络平台上，” 您的选择略显有限 <https://briefcase.readthedocs.io/en/latest/background/faq.html#can-i-use-third-party-python-packages-in-my-app>‘__。

简而言之，任何 * 纯 Python* 包（即 * 不包含二进制模块的包）都可以毫无困难地使用。但是，如果您的依赖包包含二进制组件，则必须对其进行编译；目前，大多数 Python 包都不支持非桌面平台的编译。

BeeWare 可以为一些流行的二进制模块（包括 “numpy”、” pandas “和 “加密”）提供二进制文件。为移动平台编译软件包通常是可行的，但设置起来并不容易–这已经超出了本入门教程的范围。

既然我们已经告诉了 Briefcase 我们的额外需求，那么我们就可以再次尝试打包应用程序了。确保已将更改保存到 `pyproject.toml`，然后再次更新应用程序–这一次，传递 `-r` 标志。这将告诉 Briefcase 更新打包应用程序中的需求：

macOS

Linux

Windows

```
(beeware-venv) $ briefcase update -r

[helloworld] Updating application code...
Installing src/hello_world...

[helloworld] Updating requirements...
Collecting httpx
  Using cached httpx-0.19.0-py3-none-any.whl (77 kB)
...
Installing collected packages: sniffio, idna, travertino, rfc3986, h11, anyio, toga-
↪core, rubicon-objc, httpcore, charset-normalizer, certifi, toga-cocoa, httpx
Successfully installed anyio-3.3.2 certifi-2021.10.8 charset-normalizer-2.0.6 h11-0.
↪12.0 httpcore-0.13.7 httpx-0.19.0 idna-3.2 rfc3986-1.5.0 rubicon-objc-0.4.1 sniffio-
↪1.2.0 toga-cocoa-0.3.0.dev28 toga-core-0.3.0.dev28 travertino-0.1.3

[helloworld] Removing unneeded app content...
...

[helloworld] Application updated.
```

```
(beeware-venv) $ briefcase update -r

[helloworld] Finalizing application configuration...
Targeting ubuntu:jammy (Vendor base debian)
Determining glibc version... done

Targeting glibc 2.35
Targeting Python3.10

[helloworld] Updating application code...
Installing src/hello_world...

[helloworld] Updating requirements...
Collecting httpx
  Using cached httpx-0.19.0-py3-none-any.whl (77 kB)
...
Installing collected packages: sniffio, idna, travertino, rfc3986, h11, anyio, toga-
↪core, rubicon-objc, httpcore, charset-normalizer, certifi, toga-cocoa, httpx
Successfully installed anyio-3.3.2 certifi-2021.10.8 charset-normalizer-2.0.6 h11-0.
↪12.0 httpcore-0.13.7 httpx-0.19.0 idna-3.2 rfc3986-1.5.0 rubicon-objc-0.4.1 sniffio-
↪1.2.0 toga-cocoa-0.3.0.dev28 toga-core-0.3.0.dev28 travertino-0.1.3

[helloworld] Removing unneeded app content...
...

[helloworld] Application updated.
```

```
(beeware-venv) C:\...>briefcase update -r
```

(续下页)

(接上页)

```
[helloworld] Updating application code...
Installing src/helloworld...

[helloworld] Updating requirements...
Collecting httpx
  Using cached httpx-0.19.0-py3-none-any.whl (77 kB)
...
Installing collected packages: sniffio, idna, travertino, rfc3986, h11, anyio, toga-
↪core, rubicon-objc, httpcore, charset-normalizer, certifi, toga-cocoa, httpx
Successfully installed anyio-3.3.2 certifi-2021.10.8 charset-normalizer-2.0.6 h11-0.
↪12.0 httpcore-0.13.7 httpx-0.19.0 idna-3.2 rfc3986-1.5.0 rubicon-objc-0.4.1 sniffio-
↪1.2.0 toga-cocoa-0.3.0.dev28 toga-core-0.3.0.dev28 travertino-0.1.3

[helloworld] Removing unneeded app content...
...

[helloworld] Application updated.
```

更新完成后，您可以运行 `briefcase build` 和 `briefcase run` - 您应该会看到打包后的应用程序，并带有新的对话框行为。

备注： 用于更新需求的 `-r` 选项也会被 `build` 和 ``run` 命令接受，因此如果你想一步完成更新、编译和运行，可以使用 ``briefcase run -u -r`。

2.8.4 下一步

我们现在有了一个使用第三方库的应用程序！不过，您可能已经注意到，当您按下按钮时，应用程序会变得有点反应迟钝。我们能解决这个问题吗？请访问*Tutorial 8* 了解详情…

2.9 教程 8 - 使其光滑

除非您的网络连接速度 * 快，否则您可能会注意到，当您按下按钮时，应用程序的图形用户界面会锁定一会儿。这是因为我们发出的网络请求是 * 同步 * 的。当我们的应用程序发出网络请求时，它会等待应用程序接口返回响应，然后再继续。在等待的过程中，应用程序 * 不允许重新绘制，结果导致应用程序锁定。

2.9.1 图形用户界面事件循环

要理解为什么会出现这种情况，我们需要深入了解图形用户界面应用程序的工作原理。具体细节因平台而异，但无论使用何种平台或图形用户界面环境，高层概念都是相同的。

从根本上说，图形用户界面应用程序就是一个单一的循环，看起来就像::

```
while not app.quit_requested():
    app.process_events()
    app.redraw()
```

这个循环被称为 *Event Loop*。(这些并不是实际的方法名称，而是“伪代码“的说明)。

当你点击一个按钮、拖动一个滚动条或输入一个键时，你就产生了一个“事件”。该“事件“被放入一个队列，应用程序将在下一次有机会时处理队列中的事件。为响应事件而触发的用户代码称为 * 事件处理程序 *。这些事件处理程序作为 “process_events()“调用的一部分被调用。

应用程序处理完所有可用事件后，就会“重绘 () “图形用户界面。这将考虑到事件对应用程序显示所造成的任何变化，以及操作系统中发生的任何其他情况，例如，其他应用程序的窗口可能会遮挡或显示我们应用程序的部分窗口，而我们应用程序的重绘将需要反映当前可见的窗口部分。

需要注意的重要细节是：当应用程序在处理事件时，不能重绘，也不能处理其他事件。

这意味着事件处理程序中包含的任何用户逻辑都需要快速完成。完成事件处理程序的任何延迟都会被用户观察到，表现为图形用户界面更新的减慢（或停止）。如果延迟时间足够长，操作系统可能会将此报告为问题–macOS 的“沙滩球“和 Windows 的“旋转器“图标就是操作系统在告诉你，你的应用程序在事件处理程序中耗时过长。

像“更新标签“或“重新计算输入总和“这样的简单操作很容易快速完成。然而，有很多操作是无法快速完成的。如果要执行复杂的数学计算，或为文件系统中的所有文件编制索引，或执行大型网络请求，就不能“快速完成”–这些操作本身就很慢。

那么，我们如何在图形用户界面应用程序中执行长期操作呢?

2.9.2 异步编程

我们需要的是一种方法，让处于长期事件处理程序中间的应用程序知道，只要我们能从上次中断的地方继续运行，就可以暂时将控制权释放回事件循环。应用程序可以自行决定何时释放控制权；但如果应用程序定期向事件循环释放控制权，我们就可以拥有一个长期运行的事件处理程序，并 ** 保持响应式用户界面。

我们可以通过使用 * 异步编程 * 来实现这一点。异步编程是一种描述程序的方法，它允许解释器同时运行多个函数，在所有并发运行的函数之间共享资源。

异步函数（称为 * 协同例程 *）需要明确声明为异步函数。它们还需要在内部声明何时有机会将上下文切换到另一个共同例程。

在 Python 中，异步编程是通过 `async` 和 `await` 关键字以及标准库中的 asyncio 模块来实现的。*async*‘关键字允许我们声明一个函数是异步协程。关键字 `await` 提供了一种方法来声明何时有机会将上下文切换到另一个协程。asyncio 模块为异步编码提供了一些其他有用的工具和原语。

2.9.3 使教程异步化

为了使我们的教程成为异步教程，请修改事件处理程序“say_hello()”，使其看起来像这样::

```
async def say_hello(self, widget):
    async with httpx.AsyncClient() as client:
        response = await client.get("https://jsonplaceholder.typicode.com/posts/42")

    payload = response.json()

    self.main_window.info_dialog(
        greeting(self.name_input.value),
        payload["body"],
    )
```

与上一版本相比，该代码只有 4 处改动:

1. 方法定义为 `asyncdef`，而不只是 `def`。这告诉 Python 该方法是一个异步协程。
2. 创建的客户端是异步的 `AsyncClient()` 而不是同步的 `Client()`。这就告诉 ``httpx 应在异步模式而非同步模式下运行。

3. 用于创建客户端的上下文管理器被标记为 `async`。这就告诉 Python，当上下文管理器进入和退出时，有机会释放控制。
4. get “调用带有“await “关键字。这指示应用程序在等待网络响应时，可以将控制权释放给事件循环。

Toga 允许你使用常规方法或异步协程作为处理程序；Toga 在幕后管理一切，确保处理程序按要求被调用或等待。

如果保存这些更改并重新运行应用程序（在开发模式下使用 `briefcase dev` 或更新并重新运行打包的应用程序），应用程序不会有任何明显的变化。不过，当您点击按钮触发对话框时，您可能会注意到一些细微的改进：

- 按钮会返回到“未点击“状态，而不是停留在“已点击“状态。
- 沙滩球”/”旋转器“图标不会出现
- 如果在等待对话框出现时移动/调整应用程序窗口的大小，窗口将重新绘制。
- 如果尝试打开应用程序菜单，菜单会立即出现。

2.9.4 下一步

现在，我们已经有了一个既流畅又反应灵敏的应用程序，即使在等待速度较慢的应用程序接口时也是如此。但是，我们如何确保在继续进一步开发的过程中，应用程序仍能正常运行？如何测试应用程序？请访问*Tutorial 9* 了解详情…

2.10 教程 9 - 测试时间

大多数软件开发并不涉及编写新代码，而是修改现有代码。确保现有代码以我们期望的方式继续工作是软件开发过程的关键部分。确保应用程序行为的方法之一就是 * 测试套件 *。

2.10.1 运行测试套件

原来我们的项目已经有了一个测试套件！我们最初生成项目时，会生成两个顶级目录：src “和“tests”。src “文件夹包含应用程序的代码；” tests “文件夹包含测试套件。在 `tests` 文件夹中，有一个名为 `test_app.py` 的文件，内容如下：：

```
def test_first():
    "An initial test for the app"
    assert 1 + 1 == 2
```

这是一个 Pytest *测试用例* - 可以执行的代码块，用于验证应用程序的某些行为。在本例中，该测试是一个占位符，并不测试我们应用程序的任何内容，但我们可以执行该测试。

我们可以使用 `briefcase dev` 的 `--test` 选项来运行这个测试套件。由于这是第一次运行测试，我们还需要传递 `-r` 选项，以确保测试需求也已安装：

macOS

Linux

Windows

```
(beeware-venv) $ briefcase dev --test -r

[helloworld] Installing requirements...
...
Installing dev requirements... done

[helloworld] Running test suite in dev environment...
=============================================================================
============================= test session starts ==============================
platform darwin -- Python 3.11.0, pytest-7.2.0, pluggy-1.0.0 -- /Users/brutus/beeware-
↪tutorial/beeware-venv/bin/python3.11
cachedir: /var/folders/b_/khqk71xd45d049kxc_59ltp80000gn/T/.pytest_cache
rootdir: /Users/brutus
plugins: anyio-3.6.2
collecting ... collected 1 item

tests/test_app.py::test_first PASSED                                     [100%]

============================== 1 passed in 0.01s ===============================
```

```
(beeware-venv) $ briefcase dev --test -r

[helloworld] Installing requirements...
...
Installing dev requirements... done

[helloworld] Running test suite in dev environment...
=============================================================================
============================= test session starts ==============================
platform linux -- Python 3.11.0
pytest==7.2.0
py==1.11.0
pluggy==1.0.0
cachedir: /tmp/.pytest_cache
rootdir: /home/brutus
plugins: anyio-3.6.2
collecting ... collected 1 item

tests/test_app.py::test_first PASSED                                     [100%]

============================== 1 passed in 0.01s ===============================
```

```
(beeware-venv) C:\...>briefcase dev --test -r

[helloworld] Installing requirements...
...
Installing dev requirements... done

[helloworld] Running test suite in dev environment...
=============================================================================
============================= test session starts ==============================
platform win32 -- Python 3.11.0
pytest==7.2.0
py==1.11.0
pluggy==1.0.0
cachedir: C:\Users\brutus\AppData\Local\Temp\.pytest_cache
rootdir: C:\Users\brutus
```

(续下页)

(接上页)

```
plugins: anyio-3.6.2
collecting ... collected 1 item

tests/test_app.py::test_first PASSED                                     [100%]

============================== 1 passed in 0.01s ===============================
```

成功了！我们刚刚执行了一个测试，验证了 Python 数学以我们预期的方式运行（真是松了一口气！）。

让我们用一个测试来替换这个占位符测试，以验证我们的 `greeting()` 方法是否按照我们预期的方式运行。用以下内容替换 `test_app.py` 中的内容：

```
from helloworld.app import greeting


def test_name():
    """If a name is provided, the greeting includes the name"""

    assert greeting("Alice") == "Hello, Alice"


def test_empty():
    """If a name is not provided, a generic greeting is provided"""

    assert greeting("") == "Hello, stranger"
```

这将定义两个新测试，验证我们期望看到的两种行为：提供名称时的输出和名称为空时的输出。

现在我们可以重新运行测试套件。这一次，我们不需要提供 `-r` 选项，因为测试需求已经安装完毕；我们只需要使用 `--test` 选项：

macOS

Linux

Windows

```
(beeware-venv) $ briefcase dev --test

[helloworld] Running test suite in dev environment...
================================================================================
============================== test session starts ==============================
...
collecting ... collected 2 items

tests/test_app.py::test_name PASSED                                      [ 50%]
tests/test_app.py::test_empty PASSED                                     [100%]

============================== 2 passed in 0.11s ===============================
```

```
(beeware-venv) $ briefcase dev --test

[helloworld] Running test suite in dev environment...
================================================================================
============================== test session starts ==============================
...
collecting ... collected 2 items

```

(续下页)

(接上页)

```
tests/test_app.py::test_name PASSED                                      [ 50%]
tests/test_app.py::test_empty PASSED                                     [100%]

============================== 2 passed in 0.11s ===============================
```

```
(beeware-venv) C:\...>briefcase dev --test

[helloworld] Running test suite in dev environment...
================================================================================
============================== test session starts ==============================
...
collecting ... collected 2 items

tests/test_app.py::test_name PASSED                                      [ 50%]
tests/test_app.py::test_empty PASSED                                     [100%]

============================== 2 passed in 0.11s ===============================
```

非常好我们的 `greeting()` 实用程序方法如期工作了。

2.10.2 测试驱动开发

现在我们有了测试套件，可以用它来推动新功能的开发。让我们修改应用程序，为某位用户添加特殊的问候语。首先，我们可以在 `test_app.py`：的底部为我们希望看到的新行为添加一个测试用例：

```
def test_brutus():
    """If the name is Brutus, a special greeting is provided"""

    assert greeting("Brutus") == "BeeWare the IDEs of Python!"
```

然后，用这个新测试运行测试套件：

macOS

Linux

Windows

```
(beeware-venv) $ briefcase dev --test

[helloworld] Running test suite in dev environment...
================================================================================
============================== test session starts ==============================
...
collecting ... collected 3 items

tests/test_app.py::test_name PASSED                                      [ 33%]
tests/test_app.py::test_empty PASSED                                     [ 66%]
tests/test_app.py::test_brutus FAILED                                    [100%]

================================== FAILURES ===================================
_____________________________________ test_brutus _____________________________________

    def test_brutus():
        """If the name is Brutus, a special greeting is provided"""
```

(续下页)

(接上页)

```
>       assert greeting("Brutus") == "BeeWare the IDEs of Python!"
E       AssertionError: assert 'Hello, Brutus' == 'BeeWare the IDEs of Python!'
E         - BeeWare the IDEs of Python!
E         + Hello, Brutus

tests/test_app.py:19: AssertionError
========================== short test summary info ===========================
FAILED tests/test_app.py::test_brutus - AssertionError: assert 'Hello, Brutus...
========================== 1 failed, 2 passed in 0.14s ==========================
```

```
(beeware-venv) $ briefcase dev --test

[helloworld] Running test suite in dev environment...
=============================================================================
============================== test session starts ==============================
...
collecting ... collected 3 items

tests/test_app.py::test_name PASSED                                      [ 33%]
tests/test_app.py::test_empty PASSED                                     [ 66%]
tests/test_app.py::test_brutus FAILED                                    [100%]

=================================== FAILURES ===================================
_____________________________________ test_brutus ______________________________

    def test_brutus():
        """If the name is Brutus, provide a special greeting"""

>       assert greeting("Brutus") == "BeeWare the IDEs of Python!"
E       AssertionError: assert 'Hello, Brutus' == 'BeeWare the IDEs of Python!'
E         - BeeWare the IDEs of Python!
E         + Hello, Brutus

tests/test_app.py:19: AssertionError
========================== short test summary info ===========================
FAILED tests/test_app.py::test_brutus - AssertionError: assert 'Hello, Brutus...
========================== 1 failed, 2 passed in 0.14s ==========================

============================== 2 passed in 0.11s ===============================
```

```
(beeware-venv) C:\...>briefcase dev --test

[helloworld] Running test suite in dev environment...
=============================================================================
============================== test session starts ==============================
...
collecting ... collected 3 items

tests/test_app.py::test_name PASSED                                      [ 33%]
tests/test_app.py::test_empty PASSED                                     [ 66%]
tests/test_app.py::test_brutus FAILED                                    [100%]

=================================== FAILURES ===================================
_____________________________________ test_brutus ______________________________

    def test_brutus():
```

(续下页)

(接上页)

```
        """If the name is Brutus, provide a special greeting"""

>       assert greeting("Brutus") == "BeeWare the IDEs of Python!"
E       AssertionError: assert 'Hello, Brutus' == 'BeeWare the IDEs of Python!'
E         - BeeWare the IDEs of Python!
E         + Hello, Brutus

tests/test_app.py:19: AssertionError
========================== short test summary info ===========================
FAILED tests/test_app.py::test_brutus - AssertionError: assert 'Hello, Brutus...
========================== 1 failed, 2 passed in 0.14s ==========================
```

这一次，我们看到了测试失败–输出结果解释了失败的原因：测试期望输出 “BeeWare the IDEs of Python!”，但我们的 `greeting()` 实现却返回 “Hello, Brutus”。让我们修改 `src/helloworld/app.py` 中 `greeting()` 的实现，使其具有新的行为：:

```
def greeting(name):
    if name:
        if name == "Brutus":
            return "BeeWare the IDEs of Python!"
        else:
            return f"Hello, {name}"
    else:
        return "Hello, stranger"
```

如果我们再次运行测试，就会发现测试通过了：

macOS

Linux

Windows

```
(beeware-venv) $ briefcase dev --test

[helloworld] Running test suite in dev environment...
================================================================================
============================== test session starts ==============================
...
collecting ... collected 3 items

tests/test_app.py::test_name PASSED                                      [ 33%]
tests/test_app.py::test_empty PASSED                                     [ 66%]
tests/test_app.py::test_brutus PASSED                                    [100%]

============================== 3 passed in 0.15s ===============================
```

```
(beeware-venv) $ briefcase dev --test

[helloworld] Running test suite in dev environment...
================================================================================
============================== test session starts ==============================
...
collecting ... collected 3 items

tests/test_app.py::test_name PASSED                                      [ 33%]
tests/test_app.py::test_empty PASSED                                     [ 66%]
```

(续下页)

(接上页)

```
tests/test_app.py::test_brutus PASSED                                    [100%]

============================== 3 passed in 0.15s ===============================
```

```
(beeware-venv) C:\...>briefcase dev --test

[helloworld] Running test suite in dev environment...
================================================================================
============================== test session starts ==============================
...
collecting ... collected 3 items

tests/test_app.py::test_name PASSED                                      [ 33%]
tests/test_app.py::test_empty PASSED                                     [ 66%]
tests/test_app.py::test_brutus PASSED                                    [100%]

============================== 3 passed in 0.15s ===============================
```

2.10.3 运行时测试

到目前为止，我们一直在开发模式下运行测试。这在开发新功能时尤其有用，因为您可以快速迭代添加测试，并添加代码使测试通过。不过，在某些情况下，您会希望验证您的代码在捆绑应用程序环境下是否也能正确运行。

-test"和-r"选项也可以传递给 'run'命令。如果使用 `briefcase run --test -r`，将运行相同的测试套件，但它将在打包的应用程序捆绑包内运行，而不是在开发环境中运行：

macOS

Linux

Windows

```
(beeware-venv) $ briefcase run --test -r

[helloworld] Updating application code...
Installing src/helloworld... done
Installing tests... done

[helloworld] Updating requirements...
...
[helloworld] Built build/helloworld/macos/app/Hello World.app (test mode)

[helloworld] Starting test suite...
================================================================================
Configuring isolated Python...
Pre-initializing Python runtime...
PythonHome: /Users/brutus/beeware-tutorial/helloworld/macOS/app/Hello World/Hello␣
↪World.app/Contents/Resources/support/python-stdlib
PYTHONPATH:
- /Users/brutus/beeware-tutorial/helloworld/macOS/app/Hello World/Hello World.app/
↪Contents/Resources/support/python311.zip
- /Users/brutus/beeware-tutorial/helloworld/macOS/app/Hello World/Hello World.app/
↪Contents/Resources/support/python-stdlib
- /Users/brutus/beeware-tutorial/helloworld/macOS/app/Hello World/Hello World.app/
```

(续下页)

(接上页)

```
↪Contents/Resources/support/python-stdlib/lib-dynload
- /Users/brutus/beeware-tutorial/helloworld/macOS/app/Hello World/Hello World.app/
↪Contents/Resources/app_packages
- /Users/brutus/beeware-tutorial/helloworld/macOS/app/Hello World/Hello World.app/
↪Contents/Resources/app
Configure argc/argv...
Initializing Python runtime...
Installing Python NSLog handler...
Running app module: tests.helloworld
--------------------------------------------------------------------------
============================= test session starts ==============================
...
collecting ... collected 3 items

tests/test_app.py::test_name PASSED [ 33%]
tests/test_app.py::test_empty PASSED [ 66%]
tests/test_app.py::test_brutus PASSED [100%]

============================== 3 passed in 0.21s ===============================

[helloworld] Test suite passed!
```

```
(beeware-venv) $ briefcase run --test -r

[helloworld] Finalizing application configuration...
Targeting ubuntu:jammy (Vendor base debian)
Determining glibc version... done

Targeting glibc 2.35
Targeting Python3.10

[helloworld] Updating application code...
Installing src/helloworld... done
Installing tests... done

[helloworld] Updating requirements...
...
[helloworld] Built build/helloworld/linux/ubuntu/jammy/helloworld-0.0.1/usr/bin/
↪helloworld (test mode)

[helloworld] Starting test suite...
==========================================================================
============================= test session starts ==============================
...
collecting ... collected 3 items

tests/test_app.py::test_name PASSED [ 33%]
tests/test_app.py::test_empty PASSED [ 66%]
tests/test_app.py::test_brutus PASSED [100%]

============================== 3 passed in 0.21s ===============================
```

```
(beeware-venv) C:\...>briefcase run --test -r

[helloworld] Updating application code...
Installing src/helloworld... done
```

(续下页)

(接上页)

```
Installing tests... done

[helloworld] Updating requirements...
...
[helloworld] Built build\helloworld\windows\app\src\Hello World.exe (test mode)

=============================================================================
Log started: 2022-12-02 10:57:34Z
PreInitializing Python runtime...
PythonHome: C:\Users\brutus\beeware-tutorial\helloworld\windows\app\Hello World\src
PYTHONPATH:
- C:\Users\brutus\beeware-tutorial\helloworld\windows\app\Hello World\src\python311.
↪zip
- C:\Users\brutus\beeware-tutorial\helloworld\windows\app\Hello World\src
- C:\Users\brutus\beeware-tutorial\helloworld\windows\app\Hello World\src\app_packages
- C:\Users\brutus\beeware-tutorial\helloworld\windows\app\Hello World\src\app
Configure argc/argv...
Initializing Python runtime...
Running app module: tests.helloworld
-----------------------------------------------------------------------------
============================= test session starts ==============================
...
collecting ... collected 3 items

tests/test_app.py::test_name PASSED [ 33%]
tests/test_app.py::test_empty PASSED [ 66%]
tests/test_app.py::test_brutus PASSED [100%]

============================== 3 passed in 0.21s ===============================
```

与 `briefcase dev --test` 一样，只有在第一次运行测试套件时才需要使用 `-r` 选项，以确保测试依赖项的存在。以后运行时，可以省略该选项。

你也可以在移动后端使用 `--test` 选项：因此 `briefcase run iOS --test` 和 `briefcase run android --test` 都可以在你选择的移动设备上运行测试套件。

2.10.4 下一步

We’ve now got a a test suite for our application. But it still looks like a tutorial app. Is there anything we can do about that? Turn to *Tutorial 10* to find out⋯

2.11 教程 10 - 制作属于自己的应用程序

到目前为止，我们的应用程序一直使用默认的“灰色蜜蜂“图标。如何更新应用程序，使用我们自己的图标?

2.11.1 添加图标

Every platform uses a different format for application icons - and some platforms need *multiple* icons in different sizes and shapes. To account for this, Briefcase provides a shorthand way to configure an icon once, and then have that definition expand in to all the different icons needed for each individual platform.

Edit your `pyproject.toml`, adding a new `icon` configuration item in the `[tool.briefcase.app.helloworld]` configuration section, just above the `sources` definition:

```
icon = "icons/helloworld"
```

This icon definition doesn’ t specify any file extension. The value will be used as a prefix; each platform will add additional items to this prefix to build the files needed for each platform. Some platforms require *multiple* icon files; this prefix will be combined with file size and variant modifiers to generate the list of icon files that are used.

We can now run `briefcase update` again - but this time, we pass in the `--update-resources` flag, telling Briefcase that we want to install new application resources (i.e., the icons):

macOS

Linux

Windows

Android

iOS

```
(beeware-venv) $ briefcase update --update-resources

[helloworld] Updating application code...
Installing src/helloworld... done

[helloworld] Updating application resources...
Unable to find icons/helloworld.icns for application icon; using default

[helloworld] Removing unneeded app content...
Removing unneeded app bundle content... done

[helloworld] Application updated.
```

```
(beeware-venv) $ briefcase update --update-resources

[helloworld] Updating application code...
Installing src/helloworld... done

[helloworld] Updating application resources...
Unable to find icons/helloworld-16.png for 16px application icon; using default
Unable to find icons/helloworld-32.png for 32px application icon; using default
Unable to find icons/helloworld-64.png for 64px application icon; using default
Unable to find icons/helloworld-128.png for 128px application icon; using default
Unable to find icons/helloworld-256.png for 256px application icon; using default
Unable to find icons/helloworld-512.png for 512px application icon; using default
```

(续下页)

(接上页)

```
[helloworld] Removing unneeded app content...
Removing unneeded app bundle content... done

[helloworld] Application updated.
```

```
(beeware-venv) C:\...>briefcase update --update-resources

[helloworld] Updating application code...
Installing src/helloworld... done

[helloworld] Updating application resources...
Unable to find icons/helloworld.ico for application icon; using default

[helloworld] Removing unneeded app content...
Removing unneeded app bundle content... done

[helloworld] Application updated.
```

```
(beeware-venv) $ briefcase update android --update-resources

[helloworld] Updating application code...
Installing src/helloworld... done

[helloworld] Updating application resources...
Unable to find icons/helloworld-round-48.png for 48px round application icon; using␣
↪default
Unable to find icons/helloworld-round-72.png for 72px round application icon; using␣
↪default
Unable to find icons/helloworld-round-96.png for 96px round application icon; using␣
↪default
Unable to find icons/helloworld-round-144.png for 144px round application icon; using␣
↪default
Unable to find icons/helloworld-round-192.png for 192px round application icon; using␣
↪default
Unable to find icons/helloworld-square-48.png for 48px square application icon; using␣
↪default
Unable to find icons/helloworld-square-72.png for 72px square application icon; using␣
↪default
Unable to find icons/helloworld-square-96.png for 96px square application icon; using␣
↪default
Unable to find icons/helloworld-square-144.png for 144px square application icon;␣
↪using default
Unable to find icons/helloworld-square-192.png for 192px square application icon;␣
↪using default
Unable to find icons/helloworld-square-320.png for 320px square application icon;␣
↪using default
Unable to find icons/helloworld-square-480.png for 480px square application icon;␣
↪using default
Unable to find icons/helloworld-square-640.png for 640px square application icon;␣
↪using default
Unable to find icons/helloworld-square-960.png for 960px square application icon;␣
↪using default
Unable to find icons/helloworld-square-1280.png for 1280px square application icon;␣
↪using default
Unable to find icons/helloworld-adaptive-108.png for 108px adaptive application icon;␣
```

(续下页)

(接上页)

```
↪using default
Unable to find icons/helloworld-adaptive-162.png for 162px adaptive application icon;␣
↪using default
Unable to find icons/helloworld-adaptive-216.png for 216px adaptive application icon;␣
↪using default
Unable to find icons/helloworld-adaptive-324.png for 324px adaptive application icon;␣
↪using default
Unable to find icons/helloworld-adaptive-432.png for 432px adaptive application icon;␣
↪using default

[helloworld] Removing unneeded app content...
Removing unneeded app bundle content... done

[helloworld] Application updated.
```

```
(beeware-venv) $ briefcase iOS --update-resources

  [helloworld] Updating application code...
  Installing src/helloworld... done

  [helloworld] Updating application resources...
  Unable to find icons/helloworld-20.png for 20px application icon; using default
  Unable to find icons/helloworld-29.png for 29px application icon; using default
  Unable to find icons/helloworld-40.png for 40px application icon; using default
  Unable to find icons/helloworld-58.png for 58px application icon; using default
  Unable to find icons/helloworld-60.png for 60px application icon; using default
  Unable to find icons/helloworld-76.png for 76px application icon; using default
  Unable to find icons/helloworld-80.png for 80px application icon; using default
  Unable to find icons/helloworld-87.png for 87px application icon; using default
  Unable to find icons/helloworld-120.png for 120px application icon; using default
  Unable to find icons/helloworld-152.png for 152px application icon; using default
  Unable to find icons/helloworld-167.png for 167px application icon; using default
  Unable to find icons/helloworld-180.png for 180px application icon; using default
  Unable to find icons/helloworld-640.png for 640px application icon; using default
  Unable to find icons/helloworld-1024.png for 1024px application icon; using default
  Unable to find icons/helloworld-1280.png for 1280px application icon; using default
  Unable to find icons/helloworld-1920.png for 1920px application icon; using default

  [helloworld] Removing unneeded app content...
  Removing unneeded app bundle content... done

  [helloworld] Application updated.
```

This reports the specific icon file (or files) that Briefcase is expecting. However, as we haven't provided the actual icon files, the install fails, and Briefcase falls back to a default value (the same "gray bee" icon).

Let's provide some actual icons. Download `this icons.zip bundle`, and unpack it into the root of your project directory. After unpacking, your project directory should look something like:

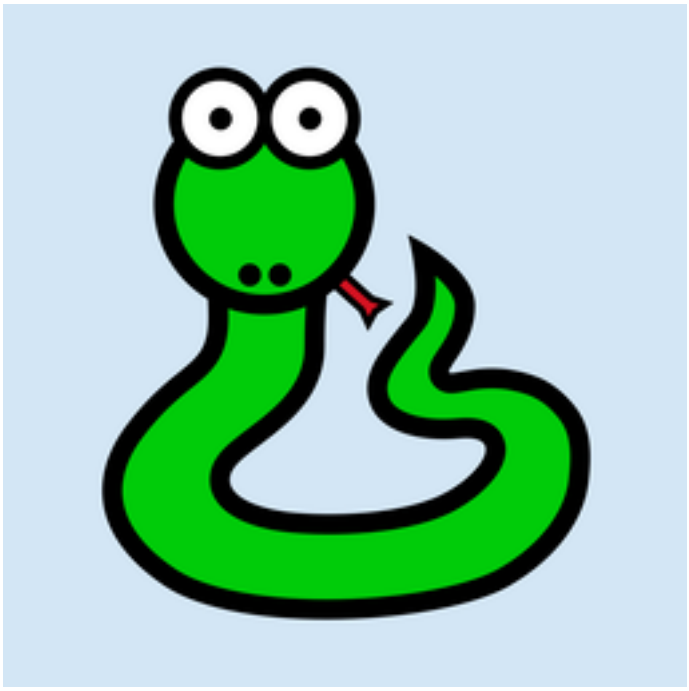
```
beeware-tutorial/
    beeware-venv/
        ...
    helloworld/
        ...
        pyproject.toml
        icons/
```

(续下页)

(接上页)

```
            helloworld.icns
            helloworld.ico
            helloworld.png
            helloworld-16.png
            ...
        src/
        ...
```

There's a *lot* of icons in this folder, but most of them should look the same: a green snake on a light blue background:



The only exception will be the icons with `-adaptive-` in their name; these will have a transparent background. This represents all the different icon sizes and shapes you need to support an app on every platform that Briefcase supports.

Now that we have icons, we can update the application again. However, `briefcase update` will only copy the updated resources into the build directory; we also want to rebuild the app to make sure the new icon is included, then start the app. We can shortcut this process by passing `--update-resources` to our call to `run` - this will update the app, update the app's resources, and then start the app:

macOS

Linux

Windows

Android

iOS

```
(beeware-venv) $ briefcase run --update-resources

[helloworld] Updating application code...
Installing src/helloworld... done

[helloworld] Updating application resources...
Installing icons/helloworld.icns as application icon... done
```

(续下页)

(接上页)

```
[helloworld] Removing unneeded app content...
Removing unneeded app bundle content... done

[helloworld] Application updated.

[helloworld] Ad-hoc signing app...
     ———————————————————————————————————————————— 100.0% • 00:01

[helloworld] Built build/helloworld/macos/app/Hello World.app

[helloworld] Starting app...
```

```
(beeware-venv) $ briefcase run --update-resources

[helloworld] Updating application code...
Installing src/helloworld... done

[helloworld] Updating application resources...
Installing icons/helloworld-16.png as 16px application icon... done
Installing icons/helloworld-32.png as 32px application icon... done
Installing icons/helloworld-64.png as 64px application icon... done
Installing icons/helloworld-128.png as 128px application icon... done
Installing icons/helloworld-256.png as 256px application icon... done
Installing icons/helloworld-512.png as 512px application icon... done

[helloworld] Removing unneeded app content...
Removing unneeded app bundle content... done

[helloworld] Application updated.

[helloworld] Building application...
Build bootstrap binary...
...

[helloworld] Built build/helloworld/linux/ubuntu/jammy/helloworld-0.0.1/usr/bin/
↪helloworld

[helloworld] Starting app...
```

```
(beeware-venv) C:\...>briefcase build --update-resources

[helloworld] Updating application code...
Installing src/helloworld... done

[helloworld] Updating application resources...
Installing icons/helloworld.ico as application icon... done

[helloworld] Removing unneeded app content...
Removing unneeded app bundle content... done

[helloworld] Application updated.

[helloworld] Building App...
Removing any digital signatures from stub app... done
Setting stub app details... done
```

(续下页)

(接上页)

```
[helloworld] Built build\helloworld\windows\app\src\Hello World.exe

[helloworld] Starting app...
```

```
(beeware-venv) $ briefcase build android --update-resources

[helloworld] Updating application code...
Installing src/helloworld... done

[helloworld] Updating application resources...
Installing icons/helloworld-round-48.png as 48px round application icon... done
Installing icons/helloworld-round-72.png as 72px round application icon... done
Installing icons/helloworld-round-96.png as 96px round application icon... done
Installing icons/helloworld-round-144.png as 144px round application icon... done
Installing icons/helloworld-round-192.png as 192px round application icon... done
Installing icons/helloworld-square-48.png as 48px square application icon... done
Installing icons/helloworld-square-72.png as 72px square application icon... done
Installing icons/helloworld-square-96.png as 96px square application icon... done
Installing icons/helloworld-square-144.png as 144px square application icon... done
Installing icons/helloworld-square-192.png as 192px square application icon... done
Installing icons/helloworld-square-320.png as 320px square application icon... done
Installing icons/helloworld-square-480.png as 480px square application icon... done
Installing icons/helloworld-square-640.png as 640px square application icon... done
Installing icons/helloworld-square-960.png as 960px square application icon... done
Installing icons/helloworld-square-1280.png as 1280px square application icon... done
Installing icons/helloworld-adaptive-108.png as 108px adaptive application icon...␣
↪done
Installing icons/helloworld-adaptive-162.png as 162px adaptive application icon...␣
↪done
Installing icons/helloworld-adaptive-216.png as 216px adaptive application icon...␣
↪done
Installing icons/helloworld-adaptive-324.png as 324px adaptive application icon...␣
↪done
Installing icons/helloworld-adaptive-432.png as 432px adaptive application icon...␣
↪done

[helloworld] Removing unneeded app content...
Removing unneeded app bundle content... done

[helloworld] Application updated.

[helloworld] Starting app...
```

备注： If you're using a recent version of Android, you may notice that the app icon has been changed to a green snake, but the background of the icon is *white*, rather than light blue. We'll fix this in the next step.

```
(beeware-venv) $ briefcase build iOS --update-resources

[helloworld] Updating application code...
Installing src/helloworld... done

[helloworld] Updating application resources...
Installing icons/helloworld-20.png as 20px application icon... done
```

(续下页)

(接上页)

```
Installing icons/helloworld-29.png as 29px application icon... done
Installing icons/helloworld-40.png as 40px application icon... done
Installing icons/helloworld-58.png as 58px application icon... done
Installing icons/helloworld-60.png as 60px application icon... done
Installing icons/helloworld-76.png as 76px application icon... done
Installing icons/helloworld-80.png as 80px application icon... done
Installing icons/helloworld-87.png as 87px application icon... done
Installing icons/helloworld-120.png as 120px application icon... done
Installing icons/helloworld-152.png as 152px application icon... done
Installing icons/helloworld-167.png as 167px application icon... done
Installing icons/helloworld-180.png as 180px application icon... done
Installing icons/helloworld-640.png as 640px application icon... done
Installing icons/helloworld-1024.png as 1024px application icon... done
Installing icons/helloworld-1280.png as 1280px application icon... done
Installing icons/helloworld-1920.png as 1920px application icon... done

[helloworld] Removing unneeded app content...
Removing unneeded app bundle content... done

[helloworld] Application updated.

[helloworld] Starting app...
```

When you run the app on iOS or Android, in addition to the icon change, you should also notice that the splash screen incorporates the new icon. However, the light blue background of the icon looks a little out of place against the white background of the splash screen. We can fix this by customizing the background color of the splash screen. Add the following definition to your `pyproject.toml`, just after the `icon` definition:

```
splash_background_color = "#D3E6F5"
```

Unfortunately, Briefcase isn't able to apply this change to an already generated project, as it requires making modifications to one of the files that was generated during the original call to `briefcase create`. To apply this change, we have to re-create the app by re-running `briefcase create`. When we do this, we'll be prompted to confirm that we want to overwrite the existing project:

macOS

Linux

Windows

Android

iOS

```
(beeware-venv) $ briefcase create

Application 'helloworld' already exists; overwrite [y/N]? y

[helloworld] Removing old application bundle...

[helloworld] Generating application template...
...

[helloworld] Created build/helloworld/macos/app
```

```
(beeware-venv) $ briefcase create

Application 'helloworld' already exists; overwrite [y/N]? y

[helloworld] Removing old application bundle...

[helloworld] Generating application template...
...

[helloworld] Created build/helloworld/linux/ubuntu/jammy
```

```
(beeware-venv) C:\...>briefcase create

Application 'helloworld' already exists; overwrite [y/N]? y

[helloworld] Removing old application bundle...

[helloworld] Generating application template...
...

[helloworld] Created build\helloworld\windows\app
```

```
(beeware-venv) $ briefcase create android

Application 'helloworld' already exists; overwrite [y/N]? y

[helloworld] Removing old application bundle...

[helloworld] Generating application template...
...
[helloworld] Created build/helloworld/android/gradle
```

```
(beeware-venv) $ briefcase create iOS

Application 'helloworld' already exists; overwrite [y/N]? y

[helloworld] Removing old application bundle...

[helloworld] Generating application template...
...
[helloworld] Created build/helloworld/ios/xcode
```

You can then re-build and re-run the app using `briefcase run`. You won't notice any changes to the desktop app; but the Android or iOS apps should now have a light blue splash screen background.

You'll need to re-create the app like this whenever you make a change to your `pyproject.toml` that doesn't relate to source code or dependencies. Any change to descriptions, version numbers, colors, or permissions will require a re-create step. Because of this, while you are developing your project, you shouldn't make any manual changes to the contents of the `build` folder, and you shouldn't add the `build` folder to your version control system. The `build` folder should be considered entirely ephemeral - an output of the build system that can be recreated as needed to reflect the current configuration of your project.

2.11.2 下一步

This has been a taste for what you can do with the tools provided by the BeeWare project. What you do from here is up to you!

Some places to go from here:

- Tutorials demonstrating features of the Toga widget toolkit.
- Details on the options available when configuring your Briefcase project.