PyWebIO

发行版本 *1.8.3*

Weimin Wang

2024 年 04 月 21 日

使用手册

PyWebIO 提供了一系列命令式的交互函数来在浏览器上获取用户输入和进行输出，将浏览器变成了一个“富文本终端”，可以用于构建简单的 Web 应用或基于浏览器的 GUI 应用。使用 PyWebIO，开发者能像编写终端脚本一样 (基于 input 和 print 进行交互) 来编写应用，无需具备 HTML 和 JS 的相关知识；PyWebIO 还可以方便地整合进现有的 Web 服务。非常适合快速构建对 UI 要求不高的应用。

CHAPTER 1

特性

- 使用同步而不是基于回调的方式获取输入，代码编写逻辑更自然
- 非声明式布局，布局方式简单高效
- 代码侵入性小，旧脚本代码仅需修改输入输出逻辑便可改造为 Web 服务
- 支持整合到现有的 Web 服务，目前支持与 Flask、Django、Tornado、aiohttp、FastAPI(Starlette) 框架集成
- 同时支持基于线程的执行模型和基于协程的执行模型
- 支持结合第三方库实现数据可视化

CHAPTER 2

Installation

稳定版:

```
pip3 install -U pywebio
```

开发版:

```
pip3 install -U https://github.com/pywebio/PyWebIO/archive/dev-release.zip
```

系统要求: PyWebIO 要求 Python 版本在 3.5.2 及以上

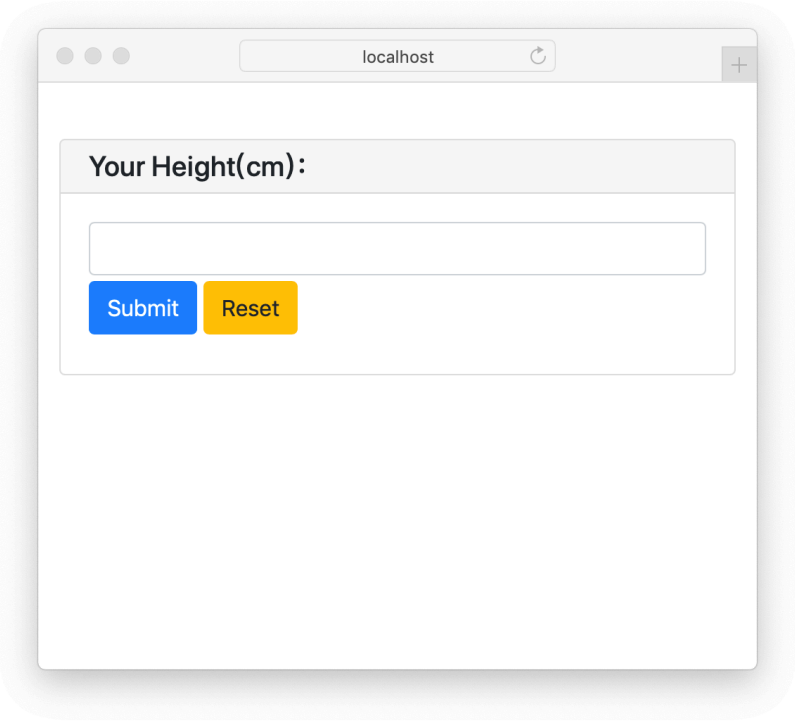
CHAPTER 3

Hello, world

这是一个使用 PyWebIO 计算 BMI 指数 的脚本:

```
from pywebio.input import input, FLOAT
from pywebio.output import put_text

def bmi():
    height = input("请输入你的身高(cm)：", type=FLOAT)
    weight = input("请输入你的体重(kg)：", type=FLOAT)

    BMI = weight / (height / 100) ** 2

    top_status = [(14.9, '极瘦'), (18.4, '偏瘦'),
                  (22.9, '正常'), (27.5, '过重'),
                  (40.0, '肥胖'), (float('inf'), '非常肥胖')]

    for top, status in top_status:
        if BMI <= top:
            put_text('你的 BMI 值：%.1f，身体状态：%s' % (BMI, status))
            break

if __name__ == '__main__':
    bmi()
```

如果没有使用 PyWebIO，这只是一个非常简单的脚本，而通过使用 PyWebIO 提供的输入输出函数，你可以在浏览器中与代码进行交互:



将上面代码最后一行对 `bmi()` 的直接调用改为使用*`pywebio.start_server(bmi, port=80)`* 便可以在 80 端口提供 `bmi()` 服务 (在线 Demo)。

将 `bmi()` 服务整合到现有的 Web 框架请参考*与 Web 框架集成* 。

CHAPTER 4

Documentation

这个文档同时也提供 PDF 和 Epub 格式.

4.1 User’s guide

如果你接触过 Web 开发，你可能对接下来描述的 PyWebIO 的用法感到不太习惯，不同于传统 Web 开发的后端实现接口、前端进行展示交互的模式，在 PyWebIO 中，所有的逻辑都通过编写 Python 代码实现。

你可以按照编写控制台程序的逻辑编写 PyWebIO 应用，只不过这里的终端变成了浏览器。通过 PyWebIO 提供的命令式 API，你可以简单地调用 `put_text()` 、`put_image()` 、`put_table()` 等函数输出文本、图片、表格等内容到浏览器，也可以调用 `input()` 、`select()` 、`file_upload()` 等函数在浏览器上显示不同表单来接收用户的输入。此外 PyWebIO 中还提供了点击事件、布局等支持，让你可以使用最少的代码完成与用户的交互，并尽可能提供良好的用户体验。

This user guide introduces you the most of the features of PyWebIO. There is a demo link at the top right of the example codes in this document, where you can run the example code online and see what happens. Also, the PyWebIO Playground is a good place to write, run and share your PyWebIO code online.

4.1.1 输入

输入函数都定义在*pywebio.input* 模块中，可以使用 `from pywebio.input import *` 引入。

调用输入函数会在浏览器上弹出一个输入表单来获取输入。PyWebIO 的输入函数是阻塞式的（和 Python 内置的 `input` 一样），在表单被成功提交之前，输入函数不会返回。

基本输入

首先是一些基本类型的输入。

文本输入:

```
age = input("How old are you?", type=NUMBER)
```

这样一行代码的效果为：浏览器会弹出一个文本输入框来获取输入，在用户完成输入将表单提交后，函数返回用户输入的值。

下面是一些其他类型的输入函数:

```
# Password input
password = input("Input password", type=PASSWORD)

# Drop-down selection
gift = select('Which gift you want?', ['keyboard', 'ipad'])

# Checkbox
agree = checkbox("User Term", options=['I agree to terms and conditions'])

# Single choice
answer = radio("Choose one", options=['A', 'B', 'C', 'D'])

# Multi-line text input
text = textarea('Text Area', rows=3, placeholder='Some text')

# File Upload
img = file_upload("Select a image:", accept="image/*")
```

输入选项

输入函数可指定的参数非常丰富（全部参数及含义请见函数文档）：

```
input('This is label', type=TEXT, placeholder='This is placeholder',
        help_text='This is help text', required=True)
```

以上代码将在浏览器上显示如下:

我们可以为输入指定校验函数，校验函数应在校验通过时返回 None，否则返回错误消息:

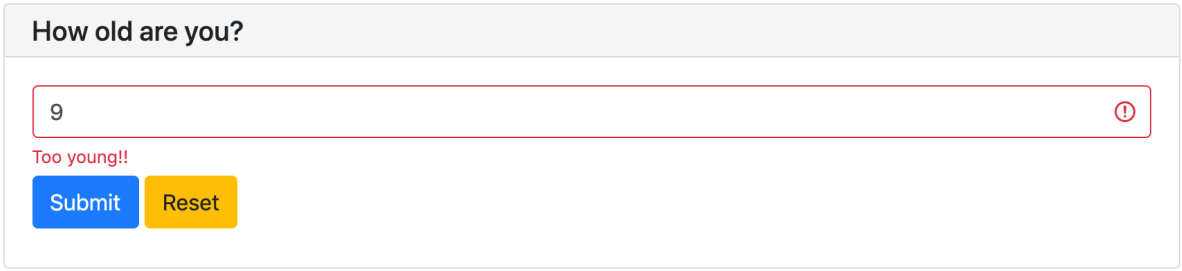
```
def check_age(p):  # return None when the check passes, otherwise return the error␣
↪message
    if p < 10:
        return 'Too young!!'
```

(续下页)

(接上页)

```
    if p > 60:
        return 'Too old!!'

age = input("How old are you?", type=NUMBER, validate=check_age)
```

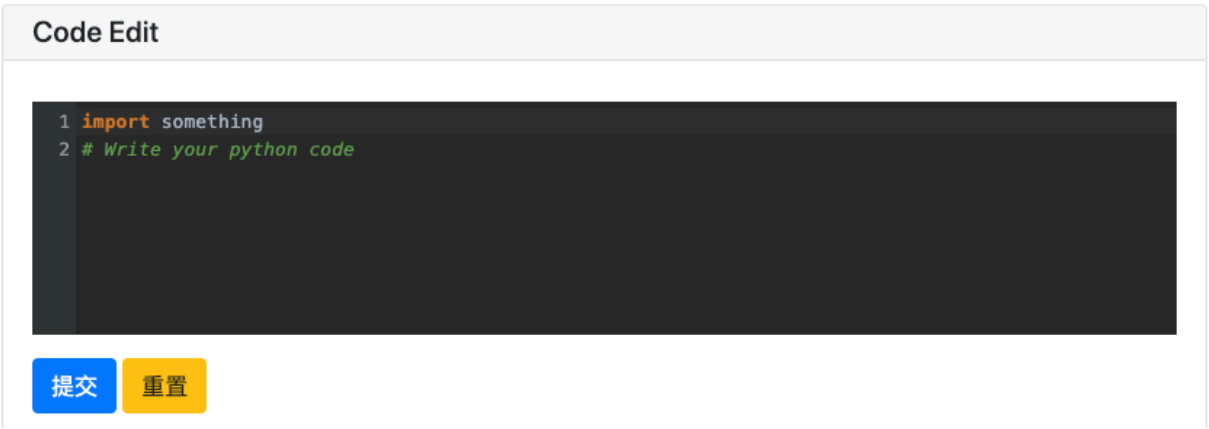
当用户输入了不合法的值时，页面上的显示如下:



pywebio.input.textarea() 中可以使用 code 参数来开启代码风格的编辑区。

```
code = textarea('Code Edit', code={
    'mode': "python",
    'theme': 'darcula',
}, value='import something\n# Write your python code')
```

以上代码将在浏览器上显示如下:



输入组

PyWebIO 支持输入组, 返回结果为一个字典。*pywebio.input.input_group()* 接受单项输入组成的列表作为参数, 返回以单项输入中的 name 作为键、以输入数据为值的字典:

```
data = input_group("Basic info",[
  input('Input your name', name='name'),
  input('Input your age', name='age', type=NUMBER, validate=check_age)
])
put_text(data['name'], data['age'])
```

输入组中同样支持使用 validate 参数设置校验函数，其接受整个表单数据作为参数:

```python
def check_form(data):  # return (input name, error msg) when validation fail
    if len(data['name']) > 6:
        return ('name', 'Name too long!')
    if data['age'] <= 0:
        return ('age', 'Age can not be negative!')
```

注意： PyWebIO 根据是否在输入函数中传入 `name` 参数来判断输入函数是在 `input_group()` 中还是被单独调用。所以当单独调用一个输入函数时，**不要**设置 `name` 参数；而在 `input_group()` 中调用输入函数时，需 **务必提供** `name` 参数。

4.1.2 输出

输出函数都定义在*pywebio.output* 模块中，可以使用 `from pywebio.output import *` 引入。

调用输出函数后，内容会实时输出到浏览器，在应用的生命周期内，可以在任意时刻调用输出函数。

基本输出

PyWebIO 提供了一系列函数来输出文本、表格、图像等格式:

```python
# Text Output
put_text("Hello world!")

# Table Output
put_table([
    ['Commodity', 'Price'],
    ['Apple', '5.5'],
    ['Banana', '7'],
])

# Image Output
put_image(open('/path/to/some/image.png', 'rb').read())  # local image
put_image('http://example.com/some-image.png')  # internet image

# Markdown Output
put_markdown('~~Strikethrough~~')

# File Output
put_file('hello_word.txt', b'hello word!')

# Show a PopUp
popup('popup title', 'popup text content')

# Show a notification message
toast('New message 🔔')
```

PyWebIO 提供的全部输出函数见*pywebio.output* 模块。另外，PyWebIO 还支持一些第三方库来进行数据可视化，参见第三方库生态 。

备注： 如果你在 Python shell, IPython 或 jupyter notebook 这种交互式执行环境中使用 PyWebIO，你需要显式调用 `show()` 方法来显示输出:

```
>>> put_text("Hello world!").show()
>>> put_table([
...     ['A', 'B'],
...     [put_markdown(...), put_text('C')]
... ]).show()
```

组合输出

函数名以 put_ 开始的输出函数，可以与一些输出函数组合使用，作为最终输出的一部分：

put_table() 支持以 put_xxx() 调用作为单元格内容:

```
put_table([
    ['Type', 'Content'],
    ['html', put_html('X<sup>2</sup>')],
    ['text', '<hr/>'],  # equal to ['text', put_text('<hr/>')]
    ['buttons', put_buttons(['A', 'B'], onclick=...)],
    ['markdown', put_markdown('`Awesome PyWebIO!`')],
    ['file', put_file('hello.text', b'hello world')],
    ['table', put_table([['A', 'B'], ['C', 'D']])]
])
```

以上代码将在浏览器上显示如下：

Type	Content
html	X^2
text	
buttons	A B
markdown	`Awesome PyWebIO!`
file	hello.text
table	A B / C D

类似地，*popup()* 也可以将 put_xxx() 调用作为弹窗内容:

```
popup('Popup title', [
    put_html('<h3>Popup Content</h3>'),
    'plain html: <br/>',  # Equivalent to: put_text('plain html: <br/>')
    put_table([['A', 'B'], ['C', 'D']]),
    put_button('close_popup()', onclick=close_popup)
])
```

另外，你可以使用*put_widget()* 来创建可以接受 put_xxx() 的自定义输出控件。

接受 put_xxx() 调用作为参数的完整输出函数清单请见输出函数列表

上下文管理器

一些接受 put_xxx() 调用作为参数的输出函数支持作为上下文管理器来使用：

```
with put_collapse('This is title'):
    for i in range(4):
        put_text(i)

    put_table([
        ['Commodity', 'Price'],
        ['Apple', '5.5'],
        ['Banana', '7'],
    ])
```

支持上下文管理器的完整函数清单请见输出函数列表

事件回调

从上面可以看出，PyWebIO 把交互分成了输入和输出两部分：输入函数为阻塞式调用，会在用户浏览器上显示一个表单，在用户提交表单之前输入函数将不会返回；输出函数将内容实时输出至浏览器。这种交互方式和控制台程序是一致的，因此 PyWebIO 应用非常适合使用控制台程序的编写逻辑来进行开发。

此外，PyWebIO 还支持事件回调：PyWebIO 允许你输出一些控件并绑定回调函数，当控件被点击时相应的回调函数便会被执行。

下面是一个例子:

```
from functools import partial

def edit_row(choice, row):
    put_text("You click %s button ar row %s" % (choice, row))

put_table([
    ['Idx', 'Actions'],
    [1, put_buttons(['edit', 'delete'], onclick=partial(edit_row, row=1))],
    [2, put_buttons(['edit', 'delete'], onclick=partial(edit_row, row=2))],
    [3, put_buttons(['edit', 'delete'], onclick=partial(edit_row, row=3))],
])
```

put_table() 的调用不会阻塞。当用户点击了某行中的按钮时，PyWebIO 会自动调用相应的回调函数:

Idx	Actions
1	edit delete
2	edit delete
3	edit delete

当然，PyWebIO 还支持单独的按钮控件:

```
def btn_click(btn_val):
    put_text("You click %s button" % btn_val)
```

(续下页)

(接上页)

```
put_buttons(['A', 'B', 'C'], onclick=btn_click)  # a group of buttons

put_button("Click me", onclick=lambda: toast("Clicked"))  # single button
```

事实上，不仅是按钮，所有的输出都可以绑定点击事件。你可以在输出函数之后调用 `onclick()` 方法来绑定点击事件:

```
put_image('some-image.png').onclick(lambda: toast('You click an image'))

# set onclick in combined output
put_table([
    ['Commodity', 'Price'],
    ['Apple', put_text('5.5').onclick(lambda: toast('You click the text'))],
])
```

`onclick()` 方法的返回值为对象本身，所以可以继续用于组合输出中。

输出域 Scope

PyWebIO 使用 scope 模型来控制内容输出的位置。scope 为输出内容的容器，你可以创建一个 scope 并将内容输出到其中。

每个输出函数（函数名形如 `put_xxx()`）都会将内容输出到一个 Scope，默认为”当前 Scope”,”当前 Scope”由*use_scope()* 设置。

use_scope()

可以使用*use_scope()* 开启并进入一个新的输出域，或进入一个已经存在的输出域:

```
with use_scope('scope1'):  # 创建并进入scope 'scope1'
    put_text('text1 in scope1')  # 输出内容到 scope1

put_text('text in parent scope of scope1')  # 输出内容到 ROOT scope

with use_scope('scope1'):  # 进入之前创建的scope 'scope1'
    put_text('text2 in scope1')  # 输出内容到 scope1
```

以上代码将会输出:

```
text1 in scope1
text2 in scope1
text in parent scope of scope1
```

use_scope() 还可以使用 `clear` 参数将 scope 中原有的内容清空:

```
with use_scope('scope2'):
    put_text('create scope2')

put_text('text in parent scope of scope2')

with use_scope('scope2', clear=True):  # enter the existing scope and clear the␣
↪previous content
    put_text('text in scope2')
```

以上代码将会输出:

```
text in scope2
text in parent scope of scope2
```

use_scope() 还可以作为装饰器来使用:

```
from datetime import datetime

@use_scope('time', clear=True)
def show_time():
    put_text(datetime.now())
```

第一次调用 show_time 时，将会创建 time 输出域并在其中输出当前时间，之后每次调用 show_time()，输出域都会被新的内容覆盖。

Scope 支持嵌套。会话开始时，PyWebIO 应用只有一个 ROOT scope。你可以在一个 scope 中创建新的 scope。比如，以下代码将会创建 3 个 scope:

```
with use_scope('A'):
    put_text('Text in scope A')

    with use_scope('B'):
        put_text('Text in scope B')

with use_scope('C'):
    put_text('Text in scope C')
```

以上代码将会产生如下 Scope 布局:

```
┌─ROOT─────────────────────┐
│                          │
│ ┌─A──────────────────┐   │
│ │ Text in scope A    │   │
│ │ ┌─B──────────────┐ │   │
│ │ │ Text in scope B │ │   │
│ │ └────────────────┘ │   │
│ └────────────────────┘   │
│                          │
│ ┌─C──────────────────┐   │
│ │ Text in scope C    │   │
│ └────────────────────┘   │
└──────────────────────────┘
```

put_scope()

我们已经知道 scope 实际上是输出内容的容器，那么我们能否将 scope 作为输出的子元素呢（比如将 scope 作为表格的一个 cell），答案是肯定的。你可以使用*put_scope()* 来显式创建一个 scope，而从它以 put_ 开头的函数名可以看出，它也可以被传递到任何可以接受 put_xxx() 调用的地方。

```
put_table([
    ['Name', 'Hobbies'],
    ['Tom', put_scope('hobby', content=put_text('Coding'))]  # hobby is initialized␣
↪to coding
])

with use_scope('hobby', clear=True):
    put_text('Movie')  # hobby is reset to Movie

# append Music, Drama to hobby
```

(续下页)

(接上页)

```
with use_scope('hobby'):
    put_text('Music')
    put_text('Drama')

# insert the Coding into the top of the hobby
put_markdown('**Coding**', scope='hobby', position=0)
```

小心： It is not allowed to have two scopes with the same name in the application.

输出域控制函数

除了*use_scope()* 和*put_scope()* , PyWebIO 还提供了以下 scope 控制函数：

- *clear(scope)*：清除 scope 的内容
- *remove(scope)*：移除 scope
- *scroll_to(scope)*：将页面滚动到 scope 处

另外，所有的输出函数还支持使用 `scope` 参数来指定输出的目的 scope，也可使用 `position` 参数来指定在目标 scope 中输出的位置。更多信息参见*output 模块* 。

布局

通常，使用上述输出函数足以完成大部分输出，但是这些输出之间全都是竖直排列的。如果想创建更复杂的布局，需要使用布局函数。

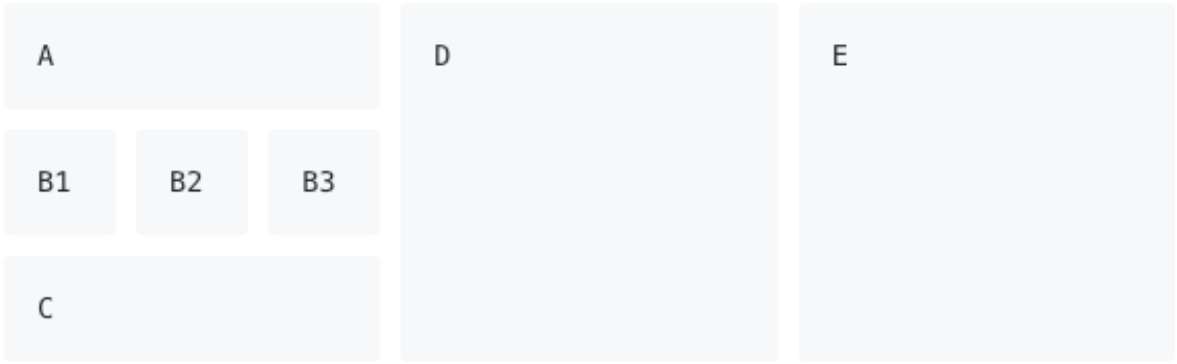
`pywebio.output` 模块提供了 3 个布局函数，通过对他们进行组合可以完成各种复杂的布局:

- *put_row()*：使用行布局输出内容. 内容在水平方向上排列
- *put_column()*：使用列布局输出内容. 内容在竖直方向上排列
- *put_grid()*：使用网格布局输出内容

通过组合 `put_row()` 和 `put_column()` 可以实现灵活布局:

```
put_row([
    put_column([
        put_code('A'),
        put_row([
            put_code('B1'), None,  # None represents the space between the output
            put_code('B2'), None,
            put_code('B3'),
        ]),
        put_code('C'),
    ]), None,
    put_code('D'), None,
    put_code('E')
])
```

以上代码将在浏览器上显示如下：



布局函数还支持自定义各部分的尺寸:

```
put_row([put_image(…), put_image(…)], size='40% 60%')  # 左右两图宽度比2:3
```

更多布局函数的用法及代码示例请查阅*布局函数文档*.

样式

如果你熟悉 CSS 样式，你还可以在输出函数后调用 `style()` 方法给输出设定自定义样式。

可以给单个的 `put_xxx()` 输出设定 CSS 样式，也可以配合组合输出使用:

```
put_text('hello').style('color: red; font-size: 20px')

# in combined output
put_row([
    put_text('hello').style('color: red'),
    put_markdown('markdown')
]).style('margin-top: 20px')
```

`style()` 方法的返回值为对象本身，所以可以继续用于组合输出中。

4.1.3 Run application

在 PyWebIO 中，有两种方式用来运行 PyWebIO 应用：作为脚本运行和使用*pywebio.start_server()* 或*pywebio.platform.path_deploy()* 来作为 Web 服务运行。

Overview

Server 模式

在 Server 模式下，PyWebIO 会启动一个 Web 服务来持续性地提供服务。当用户访问服务地址时，PyWebIO 会开启一个新会话并运行 PyWebIO 应用。

将 PyWebIO 应用部署为 web 服务的最常用方式是使用*start_server()*

```
from pywebio import *

def main():  # PyWebIO application function
    name = input.input("what's your name")
```

(续下页)

(接上页)

```
    output.put_text("hello", name)

start_server(main, port=8080, debug=True)
```

现在，在 http://127.0.0.1:8080/ 页面就会看到欢迎页面了。

使用 `debug=True` 来开启 debug 模式，这时 server 会在检测到代码发生更改后进行重启。

`start_server()` 提供了对远程访问的支持，当开启远程访问后（通过在 `start_server()` 中传入 `remote_access=True` 开启），你将会得到一个用于访问当前应用的临时的公网访问地址，其他任何人都可以使用此地址访问你的应用。远程接入可以很方便地将应用临时分享给其他人。

将 PyWebIO 应用部署为 web 服务的另一种方式是使用*`path_deploy()`* 。*`path_deploy()`* 可以从一个目录中部署 PyWebIO 应用，只需要在该目录下的 python 文件中定义 PyWebIO 应用，就可以通过 URL 中的路径来访问这些应用了。

注意： 注意，在 Server 模式下，`pywebio.input` 、`pywebio.output` 和 `pywebio.session` 模块内的函数仅能在任务函数上下文中进行调用。比如如下调用是 **不被允许的**

```
import pywebio
from pywebio.input import input

port = input('Input port number:')    # ❌ error
pywebio.start_server(my_task_func, port=int(port))
```

Script 模式

如果你在代码中没有调用 `start_server()` 或 `path_deploy()` 函数，那么你就是以脚本模式在运行 PyWebIO 应用。

在脚本模式中，当首次运行到对 PyWebIO 交互函数的调用时，会自动打开浏览器的一个页面，后续的 PyWebIO 交互都会在这个页面上进行。当脚本运行结束，这个页面也将不再有效。

如果用户在脚本结束运行之前关闭了浏览器，那么之后会话内对于 PyWebIO 交互函数的调用将会引发一个 `SessionException` 异常。

并发

PyWebIO 支持在多线程环境中使用。

Script 模式

在 Script 模式下，你可以自由地启动线程，并在其中调用 PyWebIO 的交互函数。当所有非 Daemon 线程 运行结束后，脚本退出。

Server 模式

Server 模式下，如果需要在新创建的线程中使用 PyWebIO 的交互函数，需要手动调用*`register_thread(thread)`* 对新进程进行注册（这样 PyWebIO 才能知道新创建的线程属于哪个会话）。如果新创建的线程中没有使用到 PyWebIO 的交互函数，则无需注册。没有使用*`register_thread(thread)`* 注册的线程不受会话管理，其调用 PyWebIO 的交互函数将会产生 `SessionNotFoundException` 异常。

Server 模式下多线程的使用示例:

```
def show_time():
    while True:
        with use_scope(name='time', clear=True):
            put_text(datetime.datetime.now())
            time.sleep(1)

def app():
    t = threading.Thread(target=show_time)
    register_thread(t)
    put_markdown('## Clock')
    t.start()  # run `show_time()` in background

    # ⍰ this thread will cause `SessionNotFoundException`
    threading.Thread(target=show_time).start()

    put_text('Background task started.')


start_server(app, port=8080, debug=True)
```

会话的结束

当用户关闭浏览器页面时，与之相应的会话也将被关闭。会话关闭后，应用中未返回的 PyWebIO 输入函数的调用将会抛出 `SessionClosedException` 异常，后续对 PyWebIO 交互函数的调用将会引发 `SessionNotFoundException` 或 `SessionClosedException` 异常。

大部分情况下，你不需要捕获这些异常，让这些异常来终止代码的执行通常是比较合适的。

可以使用*pywebio.session.defer_call(func)* 来设置会话结束时需要调用的函数。无论是因为用户主动关闭页面还是任务结束使得会话关闭，设置的函数都会被执行。*defer_call(func)* 可以用于资源清理等工作。在会话中可以多次调用*defer_call()*，会话结束后将会顺序执行设置的函数。

4.1.4 More about PyWebIO

目前为止，你已经了解了 PyWebIO 中最重要的特性，并且可以开始编写 PyWebIO 应用了。然而，有些功能前面我们并没有覆盖到，这里提供了对剩余特性的一些简短介绍，如果你在应用编写过程中需要用到这里的某个特性，你可以查阅对应的详细文档。

另外，你可以在*cookbook* 页面找到一些对于编写 PyWebIO 应用很有帮助的代码片段。

`session` 模块

pywebio.session 模块提供了对会话的更多控制。

- 使用*set_env()* 来为当前会话设置标题、页面外观、输入栏等内容。
- *info* 对象提供了关于当前绘画的很多信息，比如用户 IP 地址、用户语言、用户浏览器信息等。
- *local* 是一个 session-local 的存储对象，用于存储会话独立的数据。
- *run_js()* 让你在用户浏览器中执行 JavaScript 代码，*eval_js()* 让你执行并获取 JavaScript 表达式的值。

pin 模块

你已经知道，PyWebIO 的输入函数是阻塞式的，并且输入表单会在成功提交后消失。在某些时候，你可能想要输入表单一直显示并可以持续性接收用户输入，这时你可以使用*pywebio.pin* 模块。

platform 模块

pywebio.platform 模块提供了将 PyWebIO 应用以多种方式部署的支持。

PyWebIO 的服务端与浏览器可以通过两种协议 (WebSocket 和 HTTP 协议) 进行通信，默认使用 WebSocket 协议，如果你想使用 HTTP 协议，你可以选择本模块中的其他 `start_server()` 函数。

如果要为 PyWebIO 应用设置一些网页相关的配置，可以尝试使用*`pywebio.config()`* 。

高级特性

可以将 PyWebIO 应用整合到现存的 Python Web 项目中，PyWebIO 应用和 web 项目使用一个 web 框架。详细信息参见*Advanced Topic: Integration with Web Framework* 。

PyWebIO 还支持基于协程的会话。具体参见*Advanced Topic: Coroutine-based session* 。

如果你想要将 PyWebIO 应用打包到一个单独的可执行文件里面，从而使用户可以在没有安装 python 解释器的情况下运行应用，你可以参考*Build stand-alone App*

如果你想在 PyWebIO 应用中进行一些数据可视化，可以参考*Data visualization*

4.1.5 Last but not least

以上基本就是 PyWebIO 的全部功能了，你可以继续阅读接下来的文档，或者立即开始 PyWebIO 应用的编写了。

最后再提供一条建议，当你在使用 PyWebIO 遇到设计上的问题时，可以问一下自己：如果在是在终端程序中我会怎么做？如果你已经有答案了，那么在 PyWebIO 中一样可以使用这样的方式完成。如果问题依然存在或者觉得解决方案不够好，你可以考虑使用回调机制 或*pin* 模块。

OK, Have fun with PyWebIO!

4.2 pywebio.input —输入模块

本模块提供了一系列函数来从浏览器接收用户不同的形式的输入

输入函数大致分为两类，一类是单项输入:

```
name = input("What's your name")
print("Your name is %s" % name)
```

另一类是使用*input_group* 的输入组:

```
info = input_group("User info",[
  input('Input your name', name='name'),
  input('Input your age', name='age', type=NUMBER)
])
print(info['name'], info['age'])
```

输入组中需要在每一项输入函数中提供 name 参数来用于在结果中标识不同输入项.

备注： PyWebIO 根据是否在输入函数中传入 name 参数来判断输入函数是在*input_group* 中还是被单独调用。所以当你想要单独调用一个输入函数时，请不要设置 name 参数；而在*input_group* 中调用输入函数时，**务必提供** name 参数。

输入默认可以为空，如果需要用户必须提供值，则需要在输入函数中传入 required=True (部分输入函数不支持 required 参数)

本模块中的输入函数都是阻塞式的，输入表单会在成功提交后销毁。如果你想让表单可以一直显示在页面上并可以持续性接收输入，你可以考虑使用*pin* 模块。

4.2.1 函数清单

函数	简介
input	文本输入
textarea	多行文本输入
select	下拉选择框
checkbox	勾选选项
radio	单选选项
slider	滑块输入
actions	按钮选项
file_upload	文件上传
input_group	输入组
input_update	更新输入项

4.2.2 函数文档

pywebio.input.**input**(*label: str = ''*, *type: str = 'text'*, *, *validate: Callable[[Any], str | None] | None = None*, *name: str | None = None*, *value: str | int | None = None*, *action: Tuple[str, Callable[[Callable], None]] | None = None*, *onchange: Callable[[Any], None] | None = None*, *placeholder: str | None = None*, *required: bool | None = None*, *readonly: bool | None = None*, *datalist: List[str] | None = None*, *help_text: str | None = None*, ***other_html_attrs*)

文本输入

参数

- **label** (*str*) –输入框标签
- **type** (*str*) –Input type. Currently, supported types are: TEXT , NUMBER , FLOAT , PASSWORD , URL , DATE , TIME, DATETIME, COLOR

  The value of DATE , TIME, DATETIME type is a string in the format of YYYY-MM-DD , HH:MM:SS , YYYY-MM-DDTHH:MM respectively (%Y-%m-%d, %H:%M:%S, %Y-%m-%dT%H:%M in python strptime() format).
- **validate** (*callable*) –输入值校验函数。如果提供，当用户输入完毕或提交表单后校验函数将被调用。

  validate receives the input value as a parameter. When the input value is valid, it returns None. When the input value is invalid, it returns an error message string.

For example:

```
def check_age(age):
    if age>30:
        return 'Too old'
    elif age<10:
        return 'Too young'
input('Input your age', type=NUMBER, validate=check_age)
```

- **name** (*str*) –输入框的名字。与*input_group*配合使用，用于在输入组的结果中标识不同输入项。**在单个输入中，不可以设置该参数！**
- **value** (*str*) –输入框的初始值
- **action** (*tuple(label:str, callback:callable)*) –在输入框右侧显示一个按钮，用户可通过点击按钮为输入框设置值。

  label 为按钮的显示文本，callback 为按钮点击的回调函数。

  回调函数需要接收一个 set_value 位置参数，set_value 是一个可调用对象，接受单参数调用和双参数调用。

  单参数调用时，签名为 set_value(value:str) ，调用 set_value 即可将表单项的值设置为传入的 value 参数。

  双参数调用时，签名为 set_value(value:any, label:str) ，其中：

  – value 参数为最终输入项的返回值，可以为任意 Python 对象，并不会传递给用户浏览器

  – label 参数用于显示在用户表单项上

  使用双参数调用 set_value 后，用户表单项会变为只读状态。

  双参数调用的使用场景为：表单项的值通过回调动态生成，同时希望用户表单显示的和实际提交的数据不同 (例如表单项上可以显示更人性化的内容，而表单项的值则可以保存更方便被处理的对象)

  使用示例

```
import time
def set_now_ts(set_value):
    set_value(int(time.time()))

ts = input('Timestamp', type=NUMBER, action=('Now', set_now_ts))
from datetime import date,timedelta
def select_date(set_value):
    with popup('Select Date'):
        put_buttons(['Today'], onclick=[lambda: set_value(date.
↪today(), 'Today')])
        put_buttons(['Yesterday'], onclick=[lambda: set_value(date.
↪today() - timedelta(days=1), 'Yesterday')])

d = input('Date', action=('Select', select_date), readonly=True)
put_text(type(d), d)
```

  Note: 当使用基于协程的会话实现 时，回调函数 callback 可以为协程函数.

- **onchange** (*callable*) –A callback function which will be called when user change the value of this input field.

  onchange 回调函数接收一个参数——输入项改变后的值。onchange 的典型用途是配合*input_update()* 来在一个表单中实现相互依赖的输入。

- **placeholder** (*str*) –输入框的提示内容。提示内容会在输入框未输入值时以浅色字体显示在输入框中
- **required** (*bool*) –当前输入是否为必填项，默认为 `False`
- **readonly** (*bool*) –输入框是否为只读
- **datalist** (*list*) –输入建议内容列表，在页面上的显示效果为下拉候选列表，用户可以忽略建议内容列表而输入其他内容。仅当输入类型 `type` 为 `TEXT` 时可用
- **help_text** (*str*) –输入框的帮助文本。帮助文本会以小号字体显示在输入框下方
- **other_html_attrs** –在输入框上附加的额外 html 属性。参考：https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/input#%E5%B1%9E%E6%80%A7

返回

用户输入的值

pywebio.input.**textarea**(*label: str = ''*, ***, *rows: int = 6*, *code: bool | Dict | None = None*, *maxlength: int | None = None*, *minlength: int | None = None*, *validate: Callable[[Any], str | None] | None = None*, *name: str | None = None*, *value: str | None = None*, *onchange: Callable[[Any], None] | None = None*, *placeholder: str | None = None*, *required: bool | None = None*, *readonly: bool | None = None*, *help_text: str | None = None*, ***other_html_attrs*)

文本输入域（多行文本输入）

参数

- **rows** (*int*) –输入框的最多可显示的文本的行数，内容超出时会显示滚动条
- **maxlength** (*int*) –最大允许用户输入的字符长度 (Unicode) 。未指定表示无限长度
- **minlength** (*int*) –最少需要用户输入的字符长度 (Unicode)
- **code** (*dict/bool*) –通过提供 Codemirror 参数让文本输入域具有代码编辑器样式:

```
res = textarea('Text area', code={
    'mode': "python",
    'theme': 'darcula'
})
```

可以直接使用 `code={}` 或 `code=True` 开启代码编辑样式。代码编辑区支持使用 `Esc` 或 `F11` 切换全屏。

这里列举了一些常用的 Codemirror 选项

- **other_html_attrs** (*- label, validate, name, value, onchange, placeholder, required, readonly, help_text,*) –与*input* 输入函数的同名参数含义一致

返回

用户输入的文本

pywebio.input.**select**(*label: str = ''*, *options: List[Dict[str, Any] | Tuple | List | str] | None = None*, ***, *multiple: bool | None = None*, *validate: Callable[[Any], str | None] | None = None*, *name: str | None = None*, *value: List | str | None = None*, *onchange: Callable[[Any], None] | None = None*, *native: bool = True*, *required: bool | None = None*, *help_text: str | None = None*, ***other_html_attrs*)

下拉选择框

默认单选，可以通过设置 `multiple` 参数来允许多选

参数

- **options** (*list*) –可选项列表。列表项的可用形式有：
  - dict:

```
{
    "label":(str) 选项标签,
    "value":(object) 选项值,
    "selected":(bool, optional) 是否默认选中,
    "disabled":(bool, optional) 是否禁止选中
}
```

  - tuple or list: `(label, value, [selected,] [disabled])`
  - 单值: 此时 label 和 value 使用相同的值

  注意:

  1. `options` 中的 `value` 可以为任意可 JSON 序列化对象
  2. 若 `multiple` 选项不为 `True` 则可选项列表最多仅能有一项的 `selected` 为 `True`。
- **multiple** (*bool*) –是否可以多选. 默认单选
- **value** (*list or str*) –下拉选择框初始选中项的值。当 `multiple=True` 时，`value` 需为 list，否则为单个选项的值。你也可以通过设置 `options` 列表项中的 `selected` 字段来设置默认选中选项。最终选中项为 `value` 参数和 `options` 中设置的并集。
- **required** (*bool*) –是否至少选择一项，仅在 `multiple=True` 时可用
- **native** (*bool*) –Using browser’s native select component rather than bootstrap-select. This is the default behavior.
- **other_html_attrs** (– *label, validate, name, onchange, help_text,*) –与*input* 输入函数的同名参数含义一致

返回

如果 `multiple=True` 时，返回用户选中的 `options` 中的值的列表；否则，返回用户选中的 `options` 中的值

pywebio.input.**checkbox**(*label: str = ''*, *options: List[Dict[str, Any] | Tuple | List | str] | None = None*, ***, *inline: bool | None = None*, *validate: Callable[[Any], str | None] | None = None*, *name: str | None = None*, *value: List | None = None*, *onchange: Callable[[Any], None] | None = None*, *help_text: str | None = None*, ***other_html_attrs*)

勾选选项。可以多选，也可以不选。

参数

- **options** (*list*) –可选项列表。格式与同*select()* 函数的 `options` 参数
- **inline** (*bool*) –是否将选项显示在一行上。默认每个选项单独占一行
- **value** (*list*) –勾选选项初始选中项。为选项值的列表。你也可以通过设置 `options` 列表项中的 `selected` 字段来设置默认选中选项。
- **other_html_attrs** (– *label, validate, name, onchange, help_text,*) –与*input* 输入函数的同名参数含义一致

返回

用户选中的 options 中的值的列表。当用户没有勾选任何选项时，返回空列表

pywebio.input.**radio**(*label: str = ''*, *options: List[Dict[str, Any] | Tuple | List | str] | None = None*, ***, *inline: bool | None = None*, *validate: Callable[[Any], str | None] | None = None*, *name: str | None = None*, *value: str | None = None*, *onchange: Callable[[Any], None] | None = None*, *required: bool | None = None*, *help_text: str | None = None*, ***other_html_attrs*)

单选选项

参数

- **options** (*list*) –可选项列表。格式与同*select()* 函数的 options 参数
- **inline** (*bool*) –是否将选项显示在一行上。默认每个选项单独占一行
- **value** (*str*) –可选项列表。格式与同*select()* 函数的 options 参数
- **required** (*bool*) –是否一定要选择一项（默认条件下用户可以不选择任何选项）
- **other_html_attrs** (*– label, validate, name, onchange, help_text,*) –与*input* 输入函数的同名参数含义一致

返回

用户选中的选项的值, 如果用户没有选任何值，返回 None

pywebio.input.**actions**(*label: str = ''*, *buttons: List[Dict[str, Any] | Tuple | List | str] | None = None*, *name: str | None = None*, *help_text: str | None = None*)

按钮选项

在表单上显示为一组按钮，用户点击按钮后依据按钮类型的不同有不同的表现。

参数

- **buttons** (*list*) –按钮列表。列表项的可用形式有：
  - dict:

```
{
    "label":(str) 按钮标签,
    "value":(object) 按钮值,
    "type":(str, optional) 按钮类型,
    "disabled":(bool, optional) 是否禁止选择,
    "color":(str, optional) 按钮颜色

}
```

    若 type='reset'/'cancel' 或 disabled=True 可省略 value
  - tuple or list: (label, value, [type], [disabled])
  - 单值: 此时 label 和 value 使用相同的值

  其中，value 可以为任意可 JSON 序列化的对象。

  type 可选值为:
  - 'submit'：点击按钮后，立即将整个表单提交，最终表单中本项的值为被点击按钮的 value 值。'submit' 为 type 的默认值
  - 'cancel'：取消输入。点击按钮后，立即将整个表单提交，表单值返回 None
  - 'reset'：点击按钮后，将整个表单重置，输入项将变为初始状态。注意：点击 type=reset 的按钮后，并不会提交表单，actions() 调用也不会返回

  按钮的 color 值可以为：primary, secondary, success, danger, warning, info, light, dark .
- **help_text** (*– label, name,*) –与*input* 输入函数的同名参数含义一致

返回

若用户点击点击 type=submit 按钮进行表单提交，返回用户点击的按钮的值；若用户点击 type=cancel 按钮或通过其它方式提交表单，则返回 None

当 actions() 作为*input_group()* 中的最后一个输入项、并且含有 type='submit' 的按钮时，*input_group()* 表单默认的提交按钮会被当前 actions() 替换

actions() 的使用场景

- 实现简单的选择操作:

```
confirm = actions('Confirm to delete file?', ['confirm', 'cancel'],
                  help_text='Unrecoverable after file deletion')
if confirm=='confirm':
    ...
```

相比于其他输入项，使用*actions()* 用户只需要点击一次就可完成提交。

- 替换默认的提交按钮:

```
info = input_group('Add user', [
    input('username', type=TEXT, name='username', required=True),
    input('password', type=PASSWORD, name='password', required=True),
    actions('actions', [
        {'label': 'Save', 'value': 'save'},
        {'label': 'Save and add next', 'value': 'save_and_continue'},
        {'label': 'Reset', 'type': 'reset', 'color': 'warning'},
        {'label': 'Cancel', 'type': 'cancel', 'color': 'danger'},
    ], name='action', help_text='actions'),
])
put_code('info = ' + json.dumps(info, indent=4))
if info is not None:
    save_user(info['username'], info['password'])
    if info['action'] == 'save_and_continue':
        add_next()
```

pywebio.input.**file_upload**(*label: str = ''*, *accept: List | str | None = None*, *name: str | None = None*, *placeholder: str = 'Choose file'*, *multiple: bool = False*, *max_size: int | str = 0*, *max_total_size: int | str = 0*, *required: bool | None = None*, *help_text: str | None = None*, ***other_html_attrs*)

文件上传

参数

- **accept** (*str or list*) –单值或列表, 表示可接受的文件类型。文件类型的可用形式有：
  - 以 . 字符开始的文件扩展名（例如：.jpg, .png, .doc）。注意：截至本文档编写之时，微信内置浏览器还不支持这种语法
  - 一个有效的 MIME 类型。例如：application/pdf 、audio/* 表示音频文件、video/* 表示视频文件、image/* 表示图片文件。参考 https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types
- **placeholder** (*str*) –未上传文件时，文件上传框内显示的文本
- **multiple** (*bool*) –是否允许多文件上传，默认关闭
- **max_size** (*int/str*) –

单个文件的最大大小，超过限制将会禁止上传。
默认为 0，表示不限制上传文件的大小。

max_size 值可以为数字表示的字节数，或以 K / M / G 结尾表示的字符串 (分别表示千字节、兆字节、吉字节，大小写不敏感)。例如: max_size=500，max_size='40K'，max_size='3M'

- **max_total_size** (*int/str*) –所有文件的最大大小，超过限制将会禁止上传。仅在 multiple=True 时可用，默认不限制上传文件的大小。格式同 max_size 参数
- **required** (*bool*) –是否必须要上传文件。默认为 False
- **other_html_attrs** (*- label, name, help_text,*) –与*input* 输入函数的同名参数含义一致

返回

multiple=False 时 (默认)，返回 dict:

```
{
    'filename': 文件名,
    'content': 文件二进制数据(bytes object),
    'mime_type': 文件的MIME类型,
    'last_modified': 文件上次修改时间(时间戳)
}
```

若用户没有上传文件，返回 None 。

multiple=True 时，返回列表，列表项格式同上文 multiple=False 时的返回值；若用户没有上传文件，返回空列表。

备注: 若上传大文件请留意 Web 框架的文件上传大小限制设置。在使用*start_server()* 或*path_deploy()* 启动 PyWebIO 应用时，可通过 max_payload_size 参数设置 Web 框架允许上传的最大文件大小

```
# Upload a file and save to server
f = input.file_upload("Upload a file")
open('asset/'+f['filename'], 'wb').write(f['content'])

imgs = file_upload("Select some pictures:", accept="image/*", multiple=True)
for img in imgs:
    put_image(img['content'])
```

pywebio.input.**slider**(*label: str = ''*, ***, *name: str | None = None*, *value: int | float = 0*, *min_value: int | float = 0*, *max_value: int | float = 100*, *step: int = 1*, *validate: Callable[[Any], str | None] | None = None*, *onchange: Callable[[Any], None] | None = None*, *required: bool | None = None*, *help_text: str | None = None*, ***other_html_attrs*)

滑块输入

参数

- **value** (*int/float*) –滑块的初始值
- **min_value** (*int/float*) –滑块最小允许的值
- **max_value** (*int/float*) –滑块最大允许的值
- **step** (*int*) –滑动的步长。仅当 value、min_value 和 max_value 全为 int 时有效

- **other_html_attrs** (– *label, name, validate, onchange, required, help_text,*) –与*input* 输入函数的同名参数含义一致

Return int/float

若 value, min_value 和 max_value 中含有 float 类型，则返回值为 float，否则返回值为 int 类型

pywebio.input.**input_group**(*label: str = ''*, *inputs: List | None = None*, *validate: Callable[[Dict], Tuple[str, str] | None] | None = None*, *cancelable: bool = False*)

输入组。向页面上展示一组输入

参数

- **label** (*str*) –输入组标签
- **inputs** (*list*) –输入项列表。列表的内容为对单项输入函数的调用，并在单项输入函数中传入 name 参数。
- **validate** (*callable*) –输入组校验函数。

  函数签名：`callback(data) -> (name, error_msg)` validate 接收整个表单的值为参数，当校验表单值有效时，返回 None ，当某项输入值无效时，返回出错输入项的 name 值和错误提示. 比如:

```
def check_form(data):
    if len(data['name']) > 6:
        return ('name', 'Name to long!')
    if data['age'] <= 0:
        return ('age', 'Age cannot be negative!')

data = input_group("Basic info",[
    input('Input your name', name='name'),
    input('Repeat your age', name='age', type=NUMBER)
], validate=check_form)

put_text(data['name'], data['age'])
```

参数

cancelable (*bool*) –表单是否可以取消。若 cancelable=True 则会在表单底部显示一个“取消”按钮，默认为 False 。

注意：若 inputs 中最后一项输入为*actions()* ，则忽略 cancelable

返回

若用户取消表单，返回 None , 否则返回一个 dict , 其键为输入项的 name 值，字典值为输入项的值

pywebio.input.**input_update**(*name: str | None = None*, ***spec*)

更新输入项的属性。本函数仅能在输入函数的 onchange 回调中使用。

参数

- **name** (*str*) –目标输入项的 name 。可选，默认为当前触发 onchange 回调的输入项
- **spec** –需要更新的输入项参数。注意一下参数无法被更新：type, name, validate, action, code, onchange, multiple

一个具有依赖关系的输入项的示例：

```
country2city = {
    'China': ['Beijing', 'Shanghai', 'Hong Kong'],
    'USA': ['New York', 'Los Angeles', 'San Francisco'],
}
countries = list(country2city.keys())
location = input_group("Select a location", [
    select('Country', options=countries, name='country',
           onchange=lambda c: input_update('city', options=country2city[c])),
    select('City', options=country2city[countries[0]], name='city'),
])
```

4.3 pywebio.output —输出模块

本模块提供了一系列函数来输出不同形式的内容到用户浏览器，并支持灵活的输出控制。

4.3.1 函数清单

下表为 PyWebIO 提供的输出相关的函数。
其中标记有 * 的函数表示其支持接收 put_xxx 调用作为参数。
标记有 † 的函数表示其支持作为上下文管理器使用。

	函数	简介
输出域 Scope	*put_scope*	创建一个新的 scope.
	use_scope†	进入输出域
	get_scope	获取当前正在使用的输出域
	clear	清空 scope 内容
	remove	移除 Scope
	scroll_to	将页面滚动到 scope Scope 处
内容输出	*put_text*	输出文本
	put_markdown	输出 Markdown
	*put_info**† *put_success**† *put_warning**† *put_error**†	输出通知消息
	put_html	输出 Html
	put_link	输出链接
	put_progressbar	Output a progress bar
	put_loading†	输出加载提示
	put_code	输出代码块
	*put_table**	输出表格

续下页

表 1 – 接上页

	put_datatable *datatable_update* *datatable_insert* *datatable_remove*	Output and update data table
	put_button *put_buttons*	输出按钮，并绑定点击事件
	put_image	输出图片
	put_file	显示一个文件下载链接
	put_tabs[*]	输出横向标签栏 Tabs
	put_collapse[*†]	输出可折叠的内容
	put_scrollable[*†]	固定高度内容输出区域 内容超出则显示滚动条 .
	put_widget[*]	输出自定义的控件
其他交互	*toast*	显示一条通知消息
	popup[*†]	显示弹窗
	close_popup	关闭正在显示的弹窗
布局与样式	*put_row*[*†]	使用行布局输出内容
	put_column[*†]	使用列布局输出内容
	put_grid[*]	使用网格布局输出内容
	span	在*put_table()*和*put_grid()*中设置内容跨单元格
	style[*]	自定义输出内容的 css 样式

4.3.2 输出域 Scope

参见:

- *Use Guide: Output Scope*

pywebio.output.**put_scope**(*name: str*, *content: Output | List[Output] = []*, *scope: str = None*, *position: int = -1*) → Output

输出一个输出域

参数

- **name** (*str*)
- **content** (*list/put_xxx()*) –输出域里的初始内容，可以为 put_xxx() 调用或其列表。
- **position** (*int scope,*) –与*put_text* 函数的同名参数含义一致

pywebio.output.**use_scope**(*name=None*, *clear=False*)

scope 的上下文管理器和装饰器。用于创建一个新的输出域并进入，或进入一个已经存在的输出域。

参见*用户手册-use_scope()*

参数

- **name** (*str*) –scope 名. 若为 None 则生成一个全局唯一的 scope 名.（以上下文管理器形式的调用时，上下文管理器会返回 scope 名）
- **clear** (*bool*) –在进入 scope 前是否要清除 scope 里的内容

Usage

```
with use_scope(...) as scope_name:
    put_xxx()

@use_scope(...)
def app():
    put_xxx()
```

pywebio.output.**get_scope**(*stack_idx: int = -1*)

获取当前运行时 scope 栈中的 scope 名

参数

stack_idx (*int*) –当前运行时的 scope 栈索引。-1 表示当前 scope，-2 表示进入当前 scope 前的 scope，依次类推；0 表示 ROOT scope

返回

返回 Scope 栈中对应索引的 scope 名，索引错误时返回 None

pywebio.output.**clear**(*scope: str | None = None*)

清空 scope 内容

参数

scope (*str*) –目标 scope 名。默认为当前 scope

pywebio.output.**remove**(*scope: str | None = None*)

移除 Scope

参数

scope (*str*) –目标 scope 名。默认为当前 scope

pywebio.output.**scroll_to**(*scope: str | None = None, position: str = 'top'*)

将页面滚动到 scope Scope 处

参数

- **scope** (*str*) –Target scope. Default is the current scope.
- **position** (*str*) –将 Scope 置于屏幕可视区域的位置。可用值:
  - 'top'：滚动页面，让 Scope 位于屏幕可视区域顶部
  - 'middle'：滚动页面，让 Scope 位于屏幕可视区域中间
  - 'bottom'：滚动页面，让 Scope 位于屏幕可视区域底部

4.3.3 内容输出

Scope related parameters of output function

输出函数默认将内容输出到“当前 scope”，“当前 scope”可由*use_scope()* 设置。

另外，所有输入函数都支持使用 `scope` 参数来指定输出的目的 scope:

```
with use_scope('scope3'):
    put_text('text1 in scope3')   # output to current scope: scope3
    put_text('text in ROOT scope', scope='ROOT')   # output to ROOT Scope

put_text('text2 in scope3', scope='scope3')   # output to scope3
```

以上代码运行结果如下:

```
text1 in scope3
text2 in scope3
text in ROOT scope
```

一个 scope 可以包含多个输出项，输出函数的默认行为是将内容追加到目标 scope 中。可以使用输出函数的 `position` 参数来指定输出内容在目标 scope 中的插入位置。

一个 Scope 中各次输出的元素具有像数组一样的索引，最前面的编号为 0，以此往后递增加一；同样可以使用负数对 Scope 中的元素进行索引，-1 表示最后面的元素，-2 表示次后面的元素……

`position` 参数类型为整形，`position>=0` 时表示输出内容到目标 Scope 的第 position 号元素的前面；`position<0` 时表示输出内容到目标 Scope 第 position 号元素之后:

```
with use_scope('scope1'):
    put_text('A')
    put_text('B', position=0)   # insert B before A -> B A
    put_text('C', position=-2)  # insert C after B -> B C A
    put_text('D', position=1)   # insert D before C B -> B D C A
```

Output functions

pywebio.output.**put_text**(**texts: Any*, *sep: str = ' '*, *inline: bool = False*, *scope: str | None = None*, *position: int = -1*) → Output

输出文本

参数

- **texts** –要输出的内容。类型可以为任意对象，对非字符串对象会应用 `str()` 函数作为输出值。
- **sep** (*str*) –输出分隔符
- **inline** (*bool*) –将文本作为行内元素 (连续输出的文本显示在相同的段落中)。默认每次输出的文本都作为一个独立的段落
- **scope** (*str*) –内容输出的目标 scope，若 scope 不存在，则不进行任何输出操作。

  可以直接指定目标 Scope 名，或者使用 int 通过索引 Scope 栈来确定 Scope
- **position** (*int*) –在 scope 中输出的位置。

参数 `scope` 和 `position` 的更多使用说明参见用户手册

pywebio.output.**put_markdown**(*mdcontent: str*, *lstrip: bool = True*, *options: Dict[str, str | bool] | None = None*, *sanitize: bool = True*, *scope: str | None = None*, *position: int = -1*, ***kwargs*) → Output

输出 Markdown

参数

- **mdcontent** (*str*) –Markdown 文本
- **lstrip** (*bool*) –是否自动移除 mdcontent 每一行的前导空白锁进
- **options** (*dict*) –解析 Markdown 时的配置参数。PyWebIO 使用 marked 解析 Markdown, 可配置项参见: https://marked.js.org/using_advanced#options (仅支持配置 string 和 boolean 类型的项)
- **sanitize** (*bool*) –是否使用 DOMPurify 对内容进行过滤来防止 XSS 攻击。
- **position** (*int scope,*) –与*put_text* 函数的同名参数含义一致

当使用 python 三引号语法在函数中输出多行 Markdown 内容时，你可以缩进 Markdown 内容来使代码保持美观。PyWebIO 会智能地移除 Markdown 中的缩进:

```
# good code format
def hello():
    put_markdown(r""" # H1
    This is content.
    """)
```

在 1.5 版本发生变更: Enable lstrip by default. Deprecate strip_indent.

pywebio.output.**put_info**(**contents*, *closable=False*, *scope=None*, *position=-1*) → Output:

pywebio.output.**put_success**(**contents*, *closable=False*, *scope=None*, *position=-1*) → Output:

pywebio.output.**put_warning**(**contents*, *closable=False*, *scope=None*, *position=-1*) → Output:

pywebio.output.**put_error**(**contents*, *closable=False*, *scope=None*, *position=-1*) → Output:

输出通知消息

参数

- **contents** –消息内容. 元素为 put_xxx() 调用，其他类型会被转换成 put_text(content)
- **closable** (*bool*) –是否在消息框右侧展示一个关闭按钮。
- **position** (*int scope,*) –与*put_text* 函数的同名参数含义一致

Added in version 1.2.

pywebio.output.**put_html**(*html: Any*, *sanitize: bool = False*, *scope: str | None = None*, *position: int = -1*) → Output

输出 Html 内容

参数

- **html** –html 字符串
- **sanitize** (*bool*) –是否使用 DOMPurify 对内容进行过滤来防止 XSS 攻击。
- **position** (*int scope,*) –与*put_text* 函数的同名参数含义一致

pywebio.output.**put_link**(*name: str*, *url: str | None = None*, *app: str | None = None*, *new_window: bool = False*, *scope: str | None = None*, *position: int = -1*) → Output

输出链接到其他网页或 PyWebIO App 的超链接

参数

- **name** (*str*) –链接名称
- **url** (*str*) –链接到的页面地址
- **app** (*str*) –链接到的 PyWebIO 应用名。参见*Server 模式*
- **new_window** (*bool*) –是否在新窗口打开链接
- **position** (*int scope,*) –与*put_text* 函数的同名参数含义一致

url 和 app 参数必须指定一个但不可以同时指定

pywebio.output.**put_progressbar**(*name: str*, *init: float = 0*, *label: str | None = None*, *auto_close: bool = False*, *scope: str | None = None*, *position: int = -1*) → Output

Output a progress bar

参数

- **name** (*str*) –进度条名称，为进度条的唯一标识
- **init** (*float*) –进度条初始值. 进度条的值在 0 ~ 1 之间
- **label** (*str*) –The label of progress bar. The default is the percentage value of the current progress.
- **auto_close** (*bool*) –是否在进度完成后关闭进度条
- **position** (*int scope,*) –与*put_text* 函数的同名参数含义一致

Example:

```
import time

put_progressbar('bar');
for i in range(1, 11):
    set_progressbar('bar', i / 10)
    time.sleep(0.1)
```

参见:

use *set_progressbar()* to set the progress of progress bar

pywebio.output.**set_progressbar**(*name: str*, *value: float*, *label: str | None = None*)

设置进度条进度

参数

- **name** (*str*) –设置进度条进度
- **value** (*float*) –进度条的值. 范围在 0 ~ 1 之间
- **label** (*str*) –The label of progress bar. The default is the percentage value of the current progress.

See also: *put_progressbar()*

pywebio.output.**put_loading**(*shape: str = 'border'*, *color: str = 'dark'*, *scope: str | None = None*, *position: int = -1*) → Output

输出加载提示

参数

- **shape** (*str*) –加载提示的形状, 可选值: `'border'` (默认，旋转的圆环)、`'grow'` (大小渐变的圆点)

- **color**(*str*)–加载提示的颜色, 可选值: `'primary'`、`'secondary'`、`'success'`、`'danger'`、`'warning'`、`'info'`、`'light'`、`'dark'` (默认)
- **position**(*int scope,*)–与*put_text*函数的同名参数含义一致

put_loading() 支持直接调用和上下文管理器:

```
for shape in ('border', 'grow'):
    for color in ('primary', 'secondary', 'success', 'danger', 'warning', 'info',
↪'light', 'dark'):
        put_text(shape, color)
        put_loading(shape=shape, color=color)

# The loading prompt and the output inside the context will disappear
# automatically when the context block exits.
with put_loading():
    put_text("Start waiting...")
    time.sleep(3)  # Some time-consuming operations
put_text("The answer of the universe is 42")

# using style() to set the size of the loading prompt
put_loading().style('width:4rem; height:4rem')
```

在 1.8 版本发生变更: when use *put_loading()* as context manager, the output inside the context will also been removed after the context block exits.

pywebio.output.**put_code**(*content: str*, *language: str = ''*, *rows: int | None = None*, *scope: str | None = None*, *position: int = -1*) → Output

输出代码块

参数

- **content**(*str*)–代码内容
- **language**(*str*)–代码语言
- **rows**(*int*)–代码块最多可显示的文本行数，默认不限制。内容超出时会使用滚动条。
- **position**(*int scope,*)–与*put_text*函数的同名参数含义一致

pywebio.output.**put_table**(*tdata: List[List | Dict]*, *header: List[str | Tuple[Any, str]] = None*, *scope: str = None*, *position: int = -1*) → Output

输出表格

参数

- **tdata**(*list*)–表格数据。列表项可以为 `list` 或者 `dict`，单元格的内容可以为字符串或 `put_xxx` 类型的输出函数。数组项可以使用*span()* 函数来设定单元格跨度。
- **header**(*list*)–表头。当 `tdata` 的列表项为 `list` 类型时，若省略 `header` 参数，则使用 `tdata` 的第一项作为表头。表头项可以使用*span()* 函数来设定单元格跨度。

  When `tdata` is list of dict, `header` can be used to specify the order of table headers. In this case, the `header` can be a list of dict key or a list of `(<label>, <dict key>)`.
- **position**(*int scope,*)–与*put_text*函数的同名参数含义一致

Example:

```
# 'Name' cell across 2 rows, 'Address' cell across 2 columns
put_table([
    [span('Name',row=2), span('Address', col=2)],
    ['City', 'Country'],
    ['Wang', 'Beijing', 'China'],
    ['Liu', 'New York', 'America'],
])

# Use `put_xxx()` in `put_table()`
put_table([
    ['Type', 'Content'],
    ['html', put_html('X<sup>2</sup>')],
    ['text', '<hr/>'],
    ['buttons', put_buttons(['A', 'B'], onclick=...)],
    ['markdown', put_markdown('`Awesome PyWebIO!`')],
    ['file', put_file('hello.text', b'hello world')],
    ['table', put_table([['A', 'B'], ['C', 'D']])]
])

# Set table header
put_table([
    ['Wang', 'M', 'China'],
    ['Liu', 'W', 'America'],
], header=['Name', 'Gender', 'Address'])

# When ``tdata`` is list of dict
put_table([
    {"Course":"OS", "Score": "80"},
    {"Course":"DB", "Score": "93"},
], header=["Course", "Score"])  # or header=[(put_markdown("*Course*"), "Course"),
↪ (put_markdown("*Score*") ,"Score")]
```

Added in version 0.3: 单元格的内容支持 `put_xxx` 类型的输出函数

pywebio.output.**span**(*content: str | Output*, *row: int = 1*, *col: int = 1*)

用于在*put_table()* 和*put_grid()* 中设置内容跨单元格

参数

- **content** –单元格内容。可以为字符串或 `put_xxx()` 调用。
- **row** (*int*) –竖直方向跨度, 即：跨行的数目
- **col** (*int*) –水平方向跨度, 即：跨列的数目

Example

```
put_table([
    ['C'],
    [span('E', col=2)],  # 'E' across 2 columns
], header=[span('A', row=2), 'B'])  # 'A' across 2 rows

put_grid([
    [put_text('A'), put_text('B')],
    [span(put_text('A'), col=2)],  # 'A' across 2 columns
])
```

pywebio.output.**put_buttons**(*buttons: List[Dict[str, Any] | Tuple[str, Any] | List | str]*, *onclick: Callable[[Any], None] | Sequence[Callable[[], None]]*, *small: bool | None = None*, *link_style: bool = False*, *outline: bool = False*, *group: bool = False*, *scope: str | None = None*, *position: int = -1*, ***callback_options*) → Output

输出一组按钮，并绑定点击事件

参数

- **buttons** (*list*) –按钮列表。列表项的可用形式有：
  - dict:

```
{
    "label":(str) 按钮标签,,
    "value":(str) 按钮值,
    "color":(str, optional) 按钮颜色,
    "disabled":(bool, optional) 按钮时否被禁用
}
```

  - tuple or list: `(label, value)`
  - 单值: 此时 label 和 value 使用相同的值

  其中，`value` 可以为任意对象。使用 dict 类型的列表项时，支持使用 `color` key 设置按钮颜色，可选值为 `primary` 、`secondary` 、`success` 、`danger` 、`warning` 、`info` 、`light` 、`dark` .

  Example:

```
put_buttons([dict(label='success', value='s', color='success')],
↪onclick=...)
```

- **onclick** (*callable / list*) –按钮点击回调函数. `onclick` 可以是函数或者函数组成的列表.

  `onclick` 为函数时，签名为 `onclick(btn_value)`. `btn_value` 为被点击的按钮的 `value` 值

  `onclick` 为列表时，列表内函数的签名为 `func()`. 此时，回调函数与 `buttons` 一一对应

  Tip: 可以使用 `functools.partial` 来在 `onclick` 中保存更多上下文信息.

  Note: 当使用基于协程的会话实现 时，回调函数可以为协程函数.

- **small** (*bool*) –是否使用小号按钮，默认为 False
- **link_style** (*bool*) –是否将按钮显示为链接样式，默认为 False
- **outline** (*bool*) –是否将按钮显示为镂空样式，默认为 False
- **group** (*bool*) –是否将按钮合并在一起，默认为 False
- **position** (*int scope,*) –与*put_text* 函数的同名参数含义一致
- **callback_options** –回调函数的其他参数。根据选用的 session 实现有不同参数

  CoroutineBasedSession 实现

  - mutex_mode: 互斥模式。默认为 `False` 。若为 `True` ，则在运行回调函数过程中，无法响应当前按钮组的新点击事件，仅当 `onclick` 为协程函数时有效

  ThreadBasedSession 实现

– serial_mode: 串行模式模式。默认为 `False`，此时每次触发回调，回调函数会在新线程中立即执行。

开启 serial_mode 后，该按钮的回调会在会话内的一个固定线程内串行执行，且其他所有新的点击事件的回调 (包括 `serial_mode=False` 的回调) 都将排队等待当前点击事件运行完成。如果回调函数运行时间很短，可以开启 `serial_mode` 来提高性能。

Example:

```
from functools import partial

def row_action(choice, id):
    put_text("You click %s button with id: %s" % (choice, id))

put_buttons(['edit', 'delete'], onclick=partial(row_action, id=1))

def edit():
    put_text("You click edit button")
def delete():
    put_text("You click delete button")

put_buttons(['edit', 'delete'], onclick=[edit, delete])
```

在 1.5 版本发生变更: Add `disabled` button support. The `value` of button can be any object.

`pywebio.output.`**put_button**(*label: str*, *onclick: Callable[[], None]*, *color: str | None = None*, *small: bool | None = None*, *link_style: bool = False*, *outline: bool = False*, *disabled: bool = False*, *scope: str | None = None*, *position: int = -1*) → Output

输出一个按钮，并绑定点击事件

参数

- **label** (*str*) –Button label
- **onclick** (*callable*) –按钮点击回调函数
- **color** (*str*) –按钮颜色。可选值为 `primary` 、`secondary` 、`success` 、`danger` 、`warning` 、`info` 、`light` 、`dark` .
- **disabled** (*bool*) –Whether the button is disabled
- **position** (*– small, link_style, outline, scope,*) – 与*put_buttons()* 函数的同名参数含义一致

Example:

```
put_button("click me", onclick=lambda: toast("Clicked"), color='success',␣
↪outline=True)
```

Added in version 1.4.

在 1.5 版本发生变更: add `disabled` parameter

`pywebio.output.`**put_image**(*src: str | bytes | Image*, *format: str | None = None*, *title: str = ''*, *width: str | None = None*, *height: str | None = None*, *scope: str | None = None*, *position: int = -1*) → Output

输出图片

参数

- **src** –图片内容. 可以为: 字符串类型的 URL, bytes-like object 表示的图片二进制内容, `PIL.Image.Image` 实例
- **title** (*str*) –图片描述
- **width** (*str*) –图像的宽度，可以是 CSS 像素 (数字 px) 或者百分比 (数字%)。
- **height** (*str*) –图像的高度，可以是 CSS 像素 (数字 px) 或者百分比 (数字%)。可以只指定 width 和 height 中的一个值，浏览器会根据原始图像进行缩放。
- **format** (*str*) –图片格式，非必须。如 `png` , `jpeg` , `gif` 等, 仅在 `src` 为非 URL 时有效
- **position** (*int scope,*) –与*put_text* 函数的同名参数含义一致

Example:

```
img = open('/path/to/some/image.png', 'rb').read()
put_image(img, width='50px')

put_image('https://www.python.org/static/img/python-logo.png')
```

pywebio.output.**put_file**(*name: str*, *content: bytes*, *label: str | None = None*, *scope: str | None = None*, *position: int = -1*) → Output

显示一个文件下载链接

在浏览器上的显示为一个以文件名为名的链接，点击链接后浏览器自动下载文件。

参数

- **name** (*str*) –下载保存为的文件名
- **content** –文件内容. 类型为 bytes-like object
- **label** (*str*) –下载链接的显示文本，默认和文件名相同
- **position** (*int scope,*) –与*put_text* 函数的同名参数含义一致

Example:

```
content = open('./some-file', 'rb').read()
put_file('hello-world.txt', content, 'download me')
```

pywebio.output.**put_tabs**(*tabs: List[Dict[str, Any]]*, *scope: str = None*, *position: int = -1*) → Output

输出横向标签栏 Tabs

参数

- **tabs** (*list*) –标签列表, 列表项为一个 dict: `{"title": "Title", "content": ...}` ，其中 `content` 表示标签内容，可以为字符串、`put_xxx()` 调用或由它们组成的列表。
- **position** (*int scope,*) –与*put_text* 函数的同名参数含义一致

```
put_tabs([
    {'title': 'Text', 'content': 'Hello world'},
    {'title': 'Markdown', 'content': put_markdown('~~Strikethrough~~')},
    {'title': 'More content', 'content': [
        put_table([
            ['Commodity', 'Price'],
            ['Apple', '5.5'],
            ['Banana', '7'],
```

(续下页)

(接上页)

```
        ]),
        put_link('pywebio', 'https://github.com/wang0618/PyWebIO')
    ]},
])
```

Added in version 1.3.

pywebio.output.**put_collapse**(*title: str*, *content: str | Output | List[str | Output] = []*, *open: bool = False*, *scope: str = None*, *position: int = -1*) → Output

输出可折叠的内容

参数

- **title** (*str*) – 内容标题
- **content** (*list/str/put_xxx()*) – 内容可以为字符串或 put_xxx 类输出函数的返回值，或者由它们组成的列表。
- **open** (*bool*) – 是否默认展开折叠内容。默认不展开内容
- **position** (*int scope,*) – 与*put_text* 函数的同名参数含义一致

Example:

```
put_collapse('Collapse title', [
    'text',
    put_markdown('~~Strikethrough~~'),
    put_table([
        ['Commodity', 'Price'],
        ['Apple', '5.5'],
    ])
], open=True)

put_collapse('Large text', 'Awesome PyWebIO! '*30)
```

pywebio.output.**put_scrollable**(*content: str | Output | List[str | Output] = []*, *height: int | Tuple[int, int] = 400*, *keep_bottom: bool = False*, *border: bool = True*, *scope: str = None*, *position: int = -1*, ***kwargs*) → Output

固定高度内容输出区域，内容超出则显示滚动条

参数

- **content** (*list/str/put_xxx()*) – 内容可以为字符串或 put_xxx 类输出函数的返回值，或者由它们组成的列表。
- **height** (*int/tuple*) – 区域的高度（像素），内容超出此高度则使用滚动条。可以传入 (min_height, max_height) 来表示高度的范围，比如 (100, 200) 表示区域高度最小 100 像素、最高 200 像素。若不想限制高度，则传入 None
- **keep_bottom** (*bool*) – Whether to keep the content area scrolled to the bottom when updated.
- **border** (*bool*) – 是否显示边框
- **position** (*int scope,*) – 与*put_text* 函数的同名参数含义一致

Example:

```
import time

put_scrollable(put_scope('scrollable'), height=200, keep_bottom=True)
put_text("You can click the area to prevent auto scroll.", scope='scrollable')

while 1:
    put_text(time.time(), scope='scrollable')
    time.sleep(0.5)
```

在 1.1 版本发生变更: 添加 height 参数，移除 max_height 参数；添加 keep_bottom 参数

在 1.5 版本发生变更: remove horizon_scroll parameter

pywebio.output.**put_datatable**(*records: Sequence[Mapping]*, *actions: Sequence[Tuple[str, Callable[[str | int | List[str | int]], None]]] | None = None*, *onselect: Callable[[str | int | List[str | int]], None] | None = None*, *multiple_select=False*, *id_field: str | None = None*, *height: str | int = 600*, *theme: Literal['alpine', 'alpine-dark', 'balham', 'balham-dark', 'material'] = 'balham'*, *cell_content_bar=True*, *instance_id=''*, *column_order: Sequence[str] | Mapping | None = None*, *column_args: Mapping[str | Tuple, Mapping] | None = None*, *grid_args: Mapping[str, Any] | None = None*, *enterprise_key=''*, *scope: str | None = None*, *position: int = -1*) → Output

Output a datatable.

Compared with *put_table()*, *put_datatable()* is more suitable for displaying large amounts of data (both data fields and data entries), while *put_table()* is more suitable for displaying diverse data types (pictures, buttons, etc.) in cells.

This widget is powered by the awesome ag-grid library.

参数

- **records** (*list[dict]*) –data of rows, each row is a python dict, which can be nested.
- **actions** (*list*) –actions for selected row(s), they will be shown as buttons when row is selected. The format of the action item: (button_label:str, on_click:callable). Specifically, None item is allowed, which will be rendered as a separator. The on_click callback receives the selected row ID as parameter.
- **onselect** (*callable*) –callback when row is selected, receives the selected row ID as parameter.
- **multiple_select** (*bool*) –whether multiple rows can be selected. When enabled, the on_click callback in actions and the onselect callback will receive ID list of selected rows as parameter.
- **id_field** (*str/tuple*) –row ID field, that is, the key of the row dict to uniquely identifies a row. When not provide, the datatable will use the index in records to assign row ID.

  To specify the ID field of a nested dict, use a tuple to specify the path of the ID field. For example, if the row record is in {'a': {'b': ...}} format, you can use id_field=('a', 'b') to set 'b' column as the ID field.
- **height** (*int/str*) –widget height. When pass int type, the unit is pixel, when pass str type, you can specify any valid CSS height value. In particular, you can use 'auto' to make the datatable auto-size it's height to fit the content.
- **theme** (*str*) –datatable theme. Available themes are: 'balham' (default), 'alpine', 'alpine-dark', 'balham-dark', 'material'. You can preview the themes in ag-grid official example.

- **cell_content_bar** (*bool*) –whether to add a text bar to datatable to show the content of current focused cell. Default is `True`.
- **instance_id** (*str*) –Assign a unique ID to the datatable, so that you can refer this datatable in *datatable_update()*, *datatable_insert()* and *datatable_remove()* functions.
- **column_order** (*list*) –column order, the order of the column names in the list will be used as the column order. If not provided, the column order will be the same as the order of the keys in the first row of `records`. When provided, the column not in the list will not be shown. Note that `column_order` must be specified when `records` is empty.

  Since the `dict` in python is ordered after py3.7, you can use dict to specify the column order when the row record is nested dict. For example:

```
column_order = {'a': {'b': {'c': None, 'd': None}, 'e': None}, 'f
↪': None}
```

- **column_args** –column properties. Dict type, the key is str to specify the column field, the value is ag-grid column properties in dict.

  Given the row record is in this format:

```
{
    "a": {"b": ..., "c": ...},
    "b": ...,
    "c": ...
}
```

  When you set `column_args={"b": settings}`, the column settings will be applied to the column `a.b` and `b`. Use tuple as key to specify the nested key path, for example, `column_args={("a", "b"): settings}` will only apply the settings to column `a.b`.

- **grid_args** –ag-grid grid options. Refer ag-grid doc - grid options for more information.
- **enterprise_key** (*str*) –ag-grid enterprise license key. When not provided, will use the ag-grid community version.

The ag-grid library is so powerful, and you can use the `column_args` and `grid_args` parameters to achieve high customization.

Example of `put_datatable()`:

```
import urllib.request
import json

with urllib.request.urlopen('https://fakerapi.it/api/v1/persons?_quantity=30') as␣
↪f:
    data = json.load(f)['data']

put_datatable(
    data,
    actions=[
        ("Edit Email", lambda row_id: datatable_update('user', input("Email"),␣
↪row_id, "email")),
        ("Insert a Row", lambda row_id: datatable_insert('user', data[0], row_
↪id)),
        None,  # separator
        ("Delete", lambda row_id: datatable_remove('user', row_id)),
```

(续下页)

(接上页)

```
    ],
    onselect=lambda row_id: toast(f'Selected row: {row_id}'),
    instance_id='user'
)
```

The ag-grid instance can be accessed with JS global variable `ag_grid_${instance_id}_promise`:

```
ag_grid_xxx_promise.then(function(gridOptions) {
    // gridOptions is the ag-grid gridOptions object
    gridOptions.columnApi.autoSizeAllColumns();
});
```

To pass JS functions as value of `column_args` or `grid_args`, you can use `JSFunction` object:

`pywebio.output.`**`JSFunction`**(*[param1,][param2,]..., [param n,]body*)

Example:

```
put_datatable(..., grid_args=dict(sortChanged=JSFunction("event",
↪"console.log(event.source)")))
```

Since the ag-grid don't native support nested dict as row record, PyWebIO will internally flatten the nested dict before passing to ag-grid. So when you access or modify data in ag-grid directly, you need to use the following functions to help you convert the data:

- `gridOptions.flatten_row(nested_dict_record)`: flatten the nested dict record to a flat dict record
- `gridOptions.path2field(field_path_array)`: convert the field path array to field name used in ag-grid
- `gridOptions.field2path(ag_grid_column_field_name)`: convert the field name back to field path array

The implement of *`datatable_update()`*, *`datatable_insert`* and *`datatable_remove`* functions are good examples to show how to interact with ag-grid in Javascript.

`pywebio.output.`**`datatable_update`**(*instance_id: str*, *data: Any*, *row_id: str | int | None = None*, *field: str | List[str] | Tuple[str] | None = None*)

Update the whole data / a row / a cell of the datatable.

To use *`datatable_update()`*, you need to specify the `instance_id` parameter when calling *`put_datatable()`*.

When `row_id` and `field` is not specified (`datatable_update(instance_id, data)`), the whole data of datatable will be updated, in this case, the `data` parameter should be a list of dict (same as `records` in *`put_datatable()`*).

To update a row, specify the `row_id` parameter and pass the row data in dict to `data` parameter (`datatable_update(instance_id, data, row_id)`). See `id_field` of *`put_datatable()`* for more info of `row_id`.

To update a cell, specify the `row_id` and `field` parameters, in this case, the `data` parameter should be the cell value To update a row, specify the `row_id` parameter and pass the row data in dict to `data` parameter (`datatable_update(instance_id, data, row_id, field)`). The `field` can be a tuple to indicate nested key path.

`pywebio.output.`**`datatable_insert`**(*instance_id: str*, *records: List*, *row_id=None*)

Insert rows to datatable.

参数

- **instance_id** (*str*) –Datatable instance id (i.e., the `instance_id` parameter when calling *put_datatable()*)
- **records** (*dict/list[dict]*) –row record or row record list to insert
- **row_id** (*str/int*) –row id to insert before, if not specified, insert to the end

Note:

When use `id_field=None` (default) in *put_datatable()*, the row id of new inserted rows will auto increase from the last max row id.

`pywebio.output.`**datatable_remove**(*instance_id: str*, *row_ids: List*)

Remove rows from datatable.

参数

- **instance_id** (*str*) –Datatable instance id (i.e., the `instance_id` parameter when calling *put_datatable()*)
- **row_ids** (*int/str/list*) –row id or row id list to remove

`pywebio.output.`**put_widget**(*template: str*, *data: Dict[str, Any]*, *scope: str = None*, *position: int = -1*) → Output

输出自定义的控件

参数

- **template** –html 模版，使用 mustache.js 语法
- **data** (*dict*) –渲染模版使用的数据.

  数据可以包含输出函数（`put_xxx()`）的返回值，可以使用 `pywebio_output_parse` 函数来解析 `put_xxx()` 内容；对于字符串输入，`pywebio_output_parse` 会将解析成文本.

  ⚠：使用 `pywebio_output_parse` 函数时，需要关闭 mustache 的 html 转义：`{{& pywebio_output_parse}}`，参见下文示例.
- **position** (*int scope,*) –与*put_text* 函数的同名参数含义一致

Example

```
tpl = '''
<details {{#open}}open{{/open}}>
    <summary>{{title}}</summary>
    {{#contents}}
        {{& pywebio_output_parse}}
    {{/contents}}
</details>
'''

put_widget(tpl, {
    "open": True,
    "title": 'More content',
    "contents": [
        'text',
        put_markdown('~~Strikethrough~~'),
        put_table([
            ['Commodity', 'Price'],
```

(续下页)

(接上页)

```
            ['Apple', '5.5'],
            ['Banana', '7'],
        ])
    ]
})
```

4.3.4 其他交互

pywebio.output.**toast**(*content: str*, *duration: float = 2*, *position: str = 'center'*, *color: str = 'info'*, *onclick: Callable[[], None] | None = None*)

显示一条通知消息

参数

- **content** (*str*) –通知内容
- **duration** (*float*) –通知显示持续的时间，单位为秒。0 表示不自动关闭 (此时消息旁会显示一个关闭图标，用户可以手动关闭消息)
- **position** (*str*) –通知消息显示的位置，可以为 `'left'` / `'center'` / `'right'`
- **color** (*str*) –通知消息的背景颜色，可以为 `'info'` / `'error'` / `'warn'` / `'success'` 或以 `'#'` 开始的十六进制颜色值
- **onclick** (*callable*) –点击通知消息时的回调函数，回调函数不接受任何参数。

  Note: 当使用基于协程的会话实现 时，回调函数可以为协程函数.

Example:

```
def show_msg():
    put_text("You clicked the notification.")

toast('New messages', position='right', color='#2188ff', duration=0, onclick=show_
↪msg)
```

pywebio.output.**popup**(*title: str*, *content: str | Output | List[str | Output] = None*, *size: str = 'normal'*, *implicit_close: bool = True*, *closable: bool = True*)

显示弹窗

⚠：PyWebIO 不允许同时显示多个弹窗，在显示新弹窗前，会自动关闭页面上存在的弹窗。可以使用*close_popup()* 主动关闭弹窗

参数

- **title** (*str*) –弹窗标题
- **content** (*list/str/put_xxx()*) –The content of the popup. Can be a string, the put_xxx() calls, or a list of them.
- **size** (*str*) –弹窗窗口大小，可选值: `'large'`, `'normal'` and `'small'`.
- **implicit_close** (*bool*) –是否可以通过点击弹窗外的内容或按下 `Esc` 键来关闭弹窗，默认为 `False`
- **closable** (*bool*) –是否可由用户关闭弹窗. 默认情况下，用户可以通过点击弹窗右上角的关闭按钮来关闭弹窗。设置为 `False` 时弹窗仅能通过 `popup_close()` 关闭，此时 `implicit_close` 参数将被忽略.

`popup()` can be used in 2 ways: direct call and context manager.

- 直接传入内容:

```
popup('popup title', 'popup text content', size=PopupSize.SMALL)

popup('Popup title', [
    put_html('<h3>Popup Content</h3>'),
    'html: <br/>',
    put_table([['A', 'B'], ['C', 'D']]),
    put_buttons(['close_popup()'], onclick=lambda _: close_popup())
])
```

- 作为上下文管理器使用:

```
with popup('Popup title') as s:
    put_html('<h3>Popup Content</h3>')
    put_text('html: <br/>')
    put_buttons([('clear()', s)], onclick=clear)

put_text('Also work!', scope=s)
```

上下文管理器会开启一个新的输出 scope 并返回 scope 名。在上下文管理器中的输出内容默认会输出到 popup 窗口中。在上下文管理器退出后，popup 窗口并不会随之关闭，你可以继续使用输出函数的 `scope` 参数来输出内容到 popup 窗口内。

pywebio.output.**close_popup**()

关闭正在显示的弹窗

See also: *popup()*

4.3.5 布局与样式

pywebio.output.**put_row**(*content: List[Output | None] = []*, *size: str = None*, *scope: str = None*, *position: int = -1*) → Output

使用行布局输出内容. 内容在水平方向从左往右排列成一行

参数

- **content** (*list*) – 子元素列表, 列表项为 `put_xxx()` 调用或者 `None` , `None` 表示空白行间距
- **size** (*str*) –

  用于指示子元素的宽度, 为空格分割的宽度值列表.
  宽度值需要和 `content` 中子元素一一对应 (`None` 子元素也要对应宽度值).
  size 默认给 `None` 元素分配 10 像素宽度，并为剩余元素平均分配宽度.

  宽度值可用格式:

  - 像素值: 例如: `100px`
  - 百分比: 表示占可用宽度的百分比. 例如: `33.33%`
  - `fr` 关键字: 表示比例关系, 2fr 表示的宽度为 1fr 的两倍
  - `auto` 关键字: 表示由浏览器自己决定长度
  - `minmax(min, max)` : 产生一个长度范围，表示长度就在这个范围之中。它接受两个参数，分别为最小值和最大值。例如: `minmax(100px, 1fr)` 表示长度不小于 100px，不大于 1fr

- **position** (*int scope,*) –与*put_text* 函数的同名参数含义一致

Example

```
# Two code blocks of equal width, separated by 10 pixels
put_row([put_code('A'), None, put_code('B')])

# The width ratio of the left and right code blocks is 2:3, which is equivalent␣
↪to size='2fr 10px 3fr'
put_row([put_code('A'), None, put_code('B')], size='40% 10px 60%')
```

pywebio.output.**put_column**(*content: List[Output | None] = []*, *size: str = None*, *scope: str = None*, *position: int = -1*) → Output

使用列布局输出内容. 内容在竖直方向从上往下排列成一列

参数

- **content** (*list*) –子元素列表, 列表项为 `put_xxx()` 调用或者 `None` , `None` 表示空白行间距
- **size** (*str*) –用于指示子元素的高度, 为空格分割的高度值列表. 可用格式参考*put_row()* 函数的 `size` 参数注释.
- **position** (*int scope,*) –与*put_text* 函数的同名参数含义一致

pywebio.output.**put_grid**(*content: List[List[Output | None]]*, *cell_width: str = 'auto'*, *cell_height: str = 'auto'*, *cell_widths: str = None*, *cell_heights: str = None*, *direction: str = 'row'*, *scope: str = None*, *position: int = -1*) → Output

使用网格布局输出内容

参数

- **content** –输出内容. `put_xxx()` / None 组成的二维数组, None 表示空白. 数组项可以使用*span()* 函数设置元素在网格的跨度.
- **cell_width** (*str*) –网格元素的宽度.
- **cell_height** (*str*) –网格元素的高度.
- **cell_widths** (*str*) –网格每一列的宽度. 宽度值用空格分隔. 不可以和 `cell_width` 参数同时使用.
- **cell_heights** (*str*) –网格每一行的高度. 高度值用空格分隔. 不可以和 `cell_height` 参数同时使用.
- **direction** (*str*) –排列方向. 为 `'row'` 或 `'column'` .

  `'row'` 时表示，content 中的每一个子数组代表网格的一行;
  `'column'` 时表示，content 中的每一个子数组代表网格的一列.
- **position** (*int scope,*) –与*put_text* 函数的同名参数含义一致

`cell_width`,“cell_height“,“cell_widths“,“cell_heights“ 参数中的宽度/高度值格式参考*put_row()* 函数的 `size` 参数注释.

Example:

```
put_grid([
    [put_text('A'), put_text('B'), put_text('C')],
    [None, span(put_text('D'), col=2, row=1)],
    [put_text('E'), put_text('F'), put_text('G')],
], cell_width='100px', cell_height='100px')
```

pywebio.output.**style**(*outputs: Output | List[Output]*, *css_style: str*) → Output | OutputList

自定义输出内容的 css 样式

自 1.3 版本弃用: 为输出设置样式的新方式参见*User Guide*.

参数

- **outputs** (*list/put_xxx()*) –输出内容，可以为 put_xxx() 调用或其列表。outputs 为列表时将为每个列表项都添加自定义的 css 样式。
- **css_style** (*str*) –css 样式字符串

返回

添加了 css 样式的输出内容

Note: 若 outputs 为 list，则 outputs 中每一项都会被添加 css 样式, 其返回值可用于任何接受 put_xxx() 列表的地方。

Example

```
style(put_text('Red'), 'color:red')

style([
    put_text('Red'),
    put_markdown('~~del~~')
], 'color:red')

put_table([
    ['A', 'B'],
    ['C', style(put_text('Red'), 'color:red')],
])

put_collapse('title', style([
    put_text('text'),
    put_markdown('~~del~~'),
], 'margin-left:20px'))
```

4.4 pywebio.session —会话相关

pywebio.session.**download**(*name*, *content*)

向用户推送文件，用户浏览器会将文件下载到本地

参数

- **name** (*str*) –下载保存为的文件名
- **content** –文件内容. 类型为 bytes-like object

Example:

```
put_button('Click to download', lambda: download('hello-world.txt', b'hello world!
↪'))
```

pywebio.session.**run_js**(*code_*, ***args*)

在用户浏览器中运行 JavaScript 代码.

代码运行在浏览器的 JS 全局作用域中

参数

- **code** (*str*) –js 代码
- **args** –传递给 js 代码的局部变量。变量值需要可以被 json 序列化

Example:

```
run_js('console.log(a + b)', a=1, b=2)
```

pywebio.session.**eval_js**(*expression_*, ***args*)

在用户浏览器中执行 JavaScript 表达式，并获取表达式的值

参数

- **expression** (*str*) –JavaScript 表达式。表达式的值需要可以被 JSON 序列化。如果表达式的值是一个 promise ，`eval_js()` 会一直等待 promise 被 resolve，然后返回它的值，若 promise 被 reject，则返回 `None` 。
- **args** –传递给 js 代码的局部变量。变量值需要可以被 json 序列化

返回

js 表达式的值

注意 ⚠：在基于协程 的会话上下文中，需要使用 `await eval_js(expression)` 语法来进行调用。

Example:

```
current_url = eval_js("window.location.href")

function_res = eval_js('''(function(){
    var a = 1;
    a += b;
    return a;
})()''', b=100)

promise_res = eval_js('''new Promise(resolve => {
    setTimeout(() => {
        resolve('Returned inside callback.');
    }, 2000);
});''')
```

在 1.3 版本发生变更: The JS expression support return promise.

pywebio.session.**register_thread**(*thread: Threading*)

注册线程，以便在线程内调用 PyWebIO 交互函数。

仅能在默认的基于线程的会话上下文中调用。

参见*Server 模式下并发与会话的结束*

参数

thread (*threading.Thread*) –线程对象

pywebio.session.**defer_call**(*func*)

设置会话结束时调用的函数。

无论是用户主动关闭会话还是任务结束会话关闭，设置的函数都会被运行。

在会话中可以多次调用*defer_call()*，会话结束后将会顺序执行设置的函数。

defer_call 同样支持以装饰器的方式使用:

```
@defer_call
def cleanup():
   pass
```

注意： 通过*defer_call()* 设置的函数被调用时会话已经关闭，所以在函数体内不可以调用 PyWebIO 的交互函数

pywebio.session.**local**

当前会话的数据对象 (session-local object)。

`local` 是一个可以通过属性取值的字典，它的目标是用来存储应用中会话独立的状态。`local` 中存储的内容不会在会话之间共享，每个会话只能访问到自己存储在其中的数据。

使用场景

当需要在多个函数中保存一些会话独立的数据时，使用 session-local 对象保存状态会比通过函数参数传递更方便。

以下是一个会话独立的计数器的实现示例:

```
from pywebio.session import local
def add():
    local.cnt = (local.cnt or 0) + 1

def show():
    put_text(local.cnt or 0)

def main():
    put_buttons(['Add counter', 'Show counter'], [add, show])
```

而通过函数参数传递状态的实现方式为:

```
from functools import partial
def add(cnt):
    cnt[0] += 1

def show(cnt):
    put_text(cnt[0])

def main():
    cnt = [0]  # Trick: to pass by reference
    put_buttons(['Add counter', 'Show counter'], [partial(add, cnt), partial(show,
↪ cnt)])
```

当然，还可以通过函数闭包来实现相同的功能:

```
def main():
    cnt = 0

    def add():
        nonlocal cnt
        cnt += 1

    def show():
        put_text(cnt)
```

(续下页)

(接上页)

```
    put_buttons(['Add counter', 'Show counter'], [add, show])
```

local 对象支持的操作

`local` 是一个可以通过属性访问的字典，访问不存在的属性时会返回 `None` 而不是抛出异常。`local` 不支持字典的方法，支持使用 `in` 操作符来判断键是否存在，可以使用 `local._dict` 获取底层的字典表示。

```
local.name = "Wang"
local.age = 22
assert local.foo is None
local[10] = "10"

for key in local:
    print(key)

assert 'bar' not in local
assert 'name' in local

print(local._dict)
```

Added in version 1.1.

pywebio.session.**set_env**(***env_info*)

当前会话的环境设置

可配置项有:

- `title` (str): 当前页面的标题
- `output_animation` (bool): 是否启用输出动画（在输出内容时，使用过渡动画），默认启用
- `auto_scroll_bottom` (bool): 是否在内容输出时将页面自动滚动到底部，默认关闭。注意，开启后，只有输出到 ROOT Scope 才可以触发自动滚动。
- `http_pull_interval` (int): HTTP 轮询后端消息的周期（单位为毫秒，默认 1000ms），仅在基于 HTTP 连接的会话（使用 Flask 或 Django 后端）中可用
- `input_panel_fixed` (bool): 是否将输入栏固定在页面底部, 默认启用。
- `input_panel_min_height` (int): 输入栏的最小高度 (像素, 默认为 300px，最小允许 75px)。仅当 `input_panel_fixed=True` 时可用。
- `input_panel_init_height` (int): 输入栏的初始高度 (像素, 默认为 300px，最小允许 175px)。仅当 `input_panel_fixed=True` 时可用。
- `input_auto_focus` (bool): 输入栏是否自动获取输入焦点，默认为 `True` 。
- `output_max_width` (str): 页面内容区域的最大宽度（像素或百分比，例如: `'1080px'`,`'80%'`. 默认为 `'880px'`）

Example:

```
set_env(title='Awesome PyWebIO!!', output_animation=False)
```

在 1.4 版本发生变更: Added the `output_max_width` parameter

pywebio.session.**go_app**(*name*, *new_window=True*)

在同一 PyWebIO 应用的不同服务之间跳转。仅在 PyWebIO Server 模式下可用

参数

- **name** (*str*) –目标 PyWebIO 任务名
- **new_window** (*bool*) –是否在新窗口打开，默认为 True

参见：*Server 模式*

pywebio.session.**info**

表示会话信息的对象，属性有：

- user_agent：表示用户浏览器信息的对象，属性有
  - is_mobile (bool): 用户使用的设备是否为手机 (比如 iPhone, Android phones, Blackberry, Windows Phone 等设备)
  - is_tablet (bool): 用户使用的设备是否为平板 (比如 iPad, Kindle Fire, Nexus 7 等设备)
  - is_pc (bool): 用户使用的设备是否为桌面电脑 (比如运行 Windows, OS X, Linux 的设备)
  - is_touch_capable (bool): 用户使用的设备是否支持触控
  - browser.family (str): 浏览器家族. 比如‘Mobile Safari’
  - browser.version (tuple): 浏览器版本元组. 比如 (5, 1)
  - browser.version_string (str): 浏览器版本字符串. 比如‘5.1’
  - os.family (str): 操作系统家族. 比如‘iOS’
  - os.version (tuple): 操作系统版本元组. 比如 (5, 1)
  - os.version_string (str): 操作系统版本字符串. 比如‘5.1’
  - device.family (str): 设备家族. 比如‘iPhone’
  - device.brand (str): 设备品牌. 比如‘Apple’
  - device.model (str): 设备型号. 比如‘iPhone’
- user_language (str): 用户操作系统使用的语言. 比如 'zh-CN'
- server_host (str): 当前会话的服务器 host，包含域名和端口，端口为 80 时可以被省略
- origin (str): 当前用户的页面地址. 包含协议、主机、端口部分. 比如 'http://localhost:8080'. 可能为空，但保证当用户的页面地址不在当前服务器下 (即主机、端口部分和 server_host 不一致) 时有值.
- user_ip (str): 用户的 ip 地址.
- backend (str): 当前 PyWebIO 使用的后端 Server 实现. 可能出现的值有 'tornado', 'flask', 'django', 'aiohttp', 'starlette'.
- protocol (str): PyWebIO 服务器与浏览器之间的通信协议。可能的值为 'websocket' 或 'http'
- request (object): 创建当前会话时的 Web 请求对象. 根据 PyWebIO 使用的后端 Server 不同，request 的类型也不同:
  - 使用 Tornado 后端时, request 为 tornado.httputil.HTTPServerRequest 实例
  - 使用 Flask 后端时, request 为 flask.Request 实例
  - 使用 Django 后端时, request 为 django.http.HttpRequest 实例

- 使用 aiohttp 后端时, `request` 为 aiohttp.web.BaseRequest 实例
- 当使用 FastAPI 或 Starlette 时，`request` 属性为 starlette.websockets.WebSocket 实例

会话信息对象的 `user_agent` 属性是通过 user-agents 库进行解析生成的。参见 https://github.com/selwin/python-user-agents#usage

在 1.2 版本发生变更: Added the `protocol` attribute.

Example:

```
import json
from pywebio.session import info as session_info

put_code(json.dumps({
    k: str(getattr(session_info, k))
    for k in ['user_agent', 'user_language', 'server_host',
              'origin', 'user_ip', 'backend', 'protocol', 'request']
}, indent=4), 'json')
```

class pywebio.session.coroutinebased.**TaskHandler**(*close*, *closed*)

协程任务句柄

See also: *run_async()*

close()

关闭协程任务

closed() → bool

任务是否关闭

pywebio.session.**hold**()

保持会话，直到用户关闭浏览器。

> **注意:** Since PyWebIO v1.4, in *server mode*, it's no need to call this function manually, PyWebIO will automatically hold the session for you when needed. The only case to use it is to prevent the application from exiting in script mode.

如果你还在使用旧版本的 PyWebIO(强烈建议你升级版本)，这是 `hold()` 函数旧版本的文档:

在 PyWebIO 会话结束后，页面和服务端的连接便会断开，页面上需要和服务端通信才可实现的功能 (比如：下载通过*put_file()* 输出的文件，*put_buttons()* 按钮回调) 便无法使用。可以在任务函数末尾处调用 `hold()` 函数来将会话保持，这样在用户关闭浏览器页面前，会话将一直保持连接。

pywebio.session.**run_async**(*coro_obj*)

异步运行协程对象。协程中依然可以调用 PyWebIO 交互函数。

`run_async()` 仅能在基于协程 的会话上下文中调用

参数

coro_obj –协程对象

返回

`TaskHandle` 实例。通过 TaskHandle 可以查询协程运行状态和关闭协程。

参见：协程会话的并发

pywebio.session.**run_asyncio_coroutine**(*coro_obj*)

若会话线程和运行 asyncio 事件循环的线程不是同一个线程，需要用*run_asyncio_coroutine()* 来运行 asyncio 中的协程。

仅能在基于协程 的会话上下文中调用。

参数

coro_obj – asyncio 库中的协程对象

Example:

```
async def app():
    put_text('hello')
    await run_asyncio_coroutine(asyncio.sleep(1))
    put_text('world')

pywebio.platform.flask.start_server(app)
```

4.5 pywebio.platform —应用部署

platform 模块提供了以不同方式部署 PyWebIO 应用的支持。

参见:

- *Server* 模式与 *Script* 模式
- 与 *Web* 框架集成

4.5.1 Directory Deploy

可以使用 `path_deploy()` 或 `path_deploy_http()` 来从一个目录中部署 PyWebIO 应用。该目录下的 python 文件需要包含一个名为 `main` 的函数作为 PyWebIO 应用。服务端会根据用户访问的 URL 来确定需要加载的文件并从中读取 PyWebIO 应用来运行。

用户无法访问该目录下以下划线开始的文件和目录。

例如，给定如下文件结构:

```
.
├── A
│   └── a.py
├── B
│   └── b.py
└── c.py
```

三个 python 文件都含有 `main` PyWebIO 应用函数。

如果使用以上路径调用*path_deploy()*，你可以通过 URL `http://<host>:<port>/A/b`` 来访问 `b.py` 文件中的 PyWebIO 应用。若文件在运行*path_deploy()*之后被修改，可以使用 `reload` URL 参数来重载文件: `http://<host>:<port>/A/b?reload`

你还可以使用 `pywebio-path-deploy` 命令来启动一个和*path_deploy()*效果一样的 server。关于命令的更多信息请查阅命令帮助：`pywebio-path-deploy --help`

pywebio.platform.**path_deploy**(*base*, *port=0*, *host=''*, *index=True*, *static_dir=None*, *reconnect_timeout=0*, *cdn=True*, *debug=False*, *allowed_origins=None*, *check_origin=None*, *max_payload_size='200M'*, ***tornado_app_settings*)

从一个路径中部署 PyWebIO 应用

服务端使用 WebSocket 协议与浏览器进行通讯。

参数

- **base** (*str*) –用于加载 PyWebIO 应用的根目录
- **port** (*int*) –服务器监听的端口
- **host** (*str*) –服务绑定的地址
- **index** (*bool/callable*) –当请求一个文件夹时是否显示默认的索引页面，默认为 `True` 。`index` 也可以为一个函数来自定义索引页面，其接收请求的文件夹路径作为参数，返回页面 HTML 字符串。你可以在文件夹中创建一个 `index.py` PyWebIO 应用文件来重写文件夹的索引页。
- **static_dir** (*str*) –应用静态文件目录。目录下的文件可以通过 `http://<host>:<port>/static/files` 访问。例如 `static_dir` 路径下存在文件 `A/B.jpg`，则其 URL 为 `http://<host>:<port>/static/A/B.jpg`。
- **reconnect_timeout** (*int*) –客户端重连超时时间 (秒)。客户端若意外与服务端断开连接，在 `reconnect_timeout` 秒内可以重新连接并自动恢复会话。

剩余参数的详细说明见*pywebio.platform.tornado.start_server()*的同名参数。

pywebio.platform.**path_deploy_http**(*base*, *port=0*, *host=''*, *index=True*, *static_dir=None*, *cdn=True*, *debug=False*, *allowed_origins=None*, *check_origin=None*, *session_expire_seconds=None*, *session_cleanup_interval=None*, *max_payload_size='200M'*, ***tornado_app_settings*)

从一个路径中部署 PyWebIO 应用

服务端使用 HTTP 协议与浏览器进行通讯。

关于 path_deploy_http() 的 base, port, host, index, static_dir 参数的详细说明见*pywebio.platform.path_deploy()* 的同名参数。

剩余参数的详细说明见*pywebio.platform.tornado_http.start_server()* 的同名参数。

4.5.2 Application Deploy

start_server() 函数可以启动一个 Web 服务器来将 PyWebIO 应用作为 Web 服务运行。

webio_handler() 和 webio_view() 函数用于将 PyWebIO 应用整合到现有的 Python Web 项目中。

wsgi_app() 和 asgi_app() 用于获取运行 PyWebIO 应用的 WSGI 或 ASGI app。很适合当你不想使用 Web 框架内置的 server 来启动服务的情况。比如，你想使用其他 WSGI server 来启动应用，或者你正在将应用部署到一些云环境中。目前仅在 Flask、Django 和 FastApi 后端中支持 wsgi_app() / asgi_app()

在 1.1 版本发生变更: Added the cdn parameter in start_server(), webio_handler() and webio_view().

在 1.2 版本发生变更: Added the static_dir parameter in start_server().

在 1.3 版本发生变更: Added the wsgi_app() and asgi_app().

Tornado 相关

服务端可以通过两种协议（WebSocket 和 HTTP）来与用户浏览器通信。

WebSocket

pywebio.platform.tornado.**start_server**(*applications: Callable[[], None] | List[Callable[[], None]] | Dict[str, Callable[[], None]]*, *port: int = 0*, *host: str = ''*, *debug: bool = False*, *cdn: bool | str = True*, *static_dir: str | None = None*, *remote_access: bool = False*, *reconnect_timeout: int = 0*, *allowed_origins: List[str] | None = None*, *check_origin: Callable[[str], bool] | None = None*, *auto_open_webbrowser: bool = False*, *max_payload_size: int | str = '200M'*, ***tornado_app_settings*)

启动一个 Tornado server 将 PyWebIO 应用作为 Web 服务提供。

Tornado server 使用 WebSocket 协议与浏览器进行通讯。

Tornado 为 PyWebIO 应用的默认后端 Server，可以直接使用 from pywebio import start_server 导入。

参数

- **applications** (*list/dict/callable*) –PyWebIO 应用. 可以是任务函数或者任务函数的字典或列表。详细用法参见*使用 start_server() 部署多应用* 。

  任务函数为协程函数时，使用*基于协程的会话实现*；任务函数为普通函数时，使用基于线程的会话实现。

- **port** (*int*) –服务监听的端口。设置为 0 时，表示自动选择可用端口。
- **host** (*str*) –服务绑定的地址。host 可以是 IP 地址或者为 hostname。如果为 hostname，服务会监听所有与该 hostname 关联的 IP 地址。通过设置 host 为空字符串或 None 来将服务绑定到所有可用的地址上。

- **debug** (*bool*) –是否开启 Tornado Server 的 debug 模式，开启后，代码发生修改后服务器会自动重启。详情请参阅 tornado 文档
- **cdn** (*bool/str*) –是否从 CDN 加载前端静态资源，默认为 `True` 。支持传入自定义的 URL 来指定静态资源的部署地址
- **static_dir** (*str*) –应用静态文件目录。目录下的文件可以通过 `http://<host>:<port>/static/files` 访问。例如 `static_dir` 路径下存在文件 `A/B.jpg` ，则其 URL 为 `http://<host>:<port>/static/A/B.jpg`。
- **remote_access** (*bool*) –是否开启远程接入的功能。开启后，你将得到一个你当前应用的临时公网访问地址，其他人可以通过此地址访问你的应用。
- **auto_open_webbrowser** (*bool*) –当服务启动后，是否自动打开浏览器来访问服务。(该操作需要操作系统支持)
- **reconnect_timeout** (*int*) –客户端重连超时时间 (秒)。客户端若意外与服务端断开连接，在 `reconnect_timeout` 秒内可以重新连接并自动恢复会话。
- **allowed_origins** (*list*) –除当前域名外，服务器还允许的请求的来源列表。来源包含协议、域名和端口部分，允许使用 Unix shell 风格的匹配模式:
  - `*` 为通配符
  - `?` 匹配单个字符
  - `[seq]` 匹配 seq 中的任何字符
  - `[!seq]` 匹配任何不在 seq 中的字符

  比如 `https://*.example.com` 、`*://*.example.com`

  全部规则参见 Python 文档
- **check_origin** (*callable*) –请求来源检查函数。接收请求来源 (包含协议、域名和端口部分) 字符串作为参数，返回 `True/False` 指示服务器接受/拒绝该请求。若设置了 `check_origin` ，`allowed_origins` 参数将被忽略
- **auto_open_webbrowser** –当服务启动后，是否自动打开浏览器来访问服务。(该操作需要操作系统支持)
- **max_payload_size** (*int/str*) –Tornado Server 可以接受的最大 websocket 消息的大小。超过 `max_payload_size` (默认 200MB) 的消息会被拒绝接受。`max_payload_size` 可以是整形表示的字节数或以 `K` / `M` / `G` 结尾的字符串，比如: `500`, `'40K'`, `'3M'`
- **tornado_app_settings** –传递给 `tornado.web.Application` 构造函数的额外的关键字参数可设置项参考: https://www.tornadoweb.org/en/stable/web.html#tornado.web.Application.settings

pywebio.platform.tornado.**webio_handler**(*applications*, *cdn=True*, *reconnect_timeout=0*, *allowed_origins=None*, *check_origin=None*)

获取在 Tornado 中运行 PyWebIO 应用的 RequestHandler 类。RequestHandler 类基于 WebSocket 协议与浏览器进行通讯。

关于各参数的详细说明见*pywebio.platform.tornado.start_server()* 的同名参数。

HTTP

pywebio.platform.tornado_http.**start_server**(*applications*, *port=8080*, *host=''*, *debug=False*, *cdn=True*, *static_dir=None*, *allowed_origins=None*, *check_origin=None*, *auto_open_webbrowser=False*, *session_expire_seconds=None*, *session_cleanup_interval=None*, *max_payload_size='200M'*, ***tornado_app_settings*)

启动一个 Tornado server 将 PyWebIO 应用作为 Web 服务提供。

服务端使用 HTTP 协议与浏览器进行通讯。

参数

- **session_expire_seconds** (*int*) –会话过期时间，单位为秒（默认 60 秒）。若 session_expire_seconds 秒内没有收到客户端的请求，则认为会话过期。
- **session_cleanup_interval** (*int*) –会话清理间隔，单位为秒 (默认 12 秒)。服务端会周期性清理过期的会话，释放会话占用的资源。
- **max_payload_size** (*int/str*) –Tornado Server 可以接受的最大单个 HTTP 请求的大小

剩余参数的详细说明见*pywebio.platform.tornado.start_server()* 的同名参数。

Added in version 1.2.

pywebio.platform.tornado_http.**webio_handler**(*applications*, *cdn=True*, *session_expire_seconds=None*, *session_cleanup_interval=None*, *allowed_origins=None*, *check_origin=None*)

获取在 Tornado 中运行 PyWebIO 应用的 RequestHandler 类。RequestHandler 类基于 HTTP 协议与浏览器进行通讯。

关于各参数的详细说明见*pywebio.platform.tornado_http.start_server()* 的同名参数。

Added in version 1.2.

Flask support

使用 Flask 后端作为 PyWebIO 应用 Server 时，需要您自行安装 Flask，并确保版本大于等于 `0.10` 。可通过以下命令安装:

```
pip3 install -U flask>=0.10
```

pywebio.platform.flask.**webio_view**(*applications*, *cdn=True*, *session_expire_seconds=None*, *session_cleanup_interval=None*, *allowed_origins=None*, *check_origin=None*)

获取在 Flask 中运行 PyWebIO 任务的视图函数。基于 http 请求与前端页面进行通讯

关于各参数的详细说明见*pywebio.platform.flask.start_server()* 的同名参数。

pywebio.platform.flask.**wsgi_app**(*applications*, *cdn=True*, *static_dir=None*, *allowed_origins=None*, *check_origin=None*, *session_expire_seconds=None*, *session_cleanup_interval=None*, *max_payload_size='200M'*)

将 PyWebIO 应用转换为 Flask WSGI 应用

关于各参数的详细说明见*pywebio.platform.flask.start_server()* 的同名参数。

pywebio.platform.flask.**start_server**(*applications*, *port=8080*, *host=''*, *cdn=True*, *static_dir=None*, *remote_access=False*, *allowed_origins=None*, *check_origin=None*, *session_expire_seconds=None*, *session_cleanup_interval=None*, *debug=False*, *max_payload_size='200M'*, ***flask_options*)

启动一个 Flask server 将 PyWebIO 应用作为 Web 服务提供。

参数

- **session_expire_seconds** (*int*) –会话过期时间，单位为秒（默认 600 秒）。若 session_expire_seconds 秒内没有收到客户端的请求，则认为会话过期。
- **session_cleanup_interval** (*int*) –会话清理间隔，单位为秒 (默认 300 秒)。服务端会周期性清理过期的会话，释放会话占用的资源。
- **debug** (*bool*) –是否开启 Flask Server 的 debug 模式，开启后，代码发生修改后服务器会自动重启。
- **max_payload_size** (*int/str*) –Flask Server 可以接受的最大单个 HTTP 请求的大小
- **flask_options** –传递给 `flask.Flask.run` 函数的额外的关键字参数可设置项参考: https://flask.palletsprojects.com/en/1.1.x/api/#flask.Flask.run

关于各参数的详细说明见*pywebio.platform.tornado.start_server()* 的同名参数

Django support

使用 Django 后端作为 PyWebIO 应用 Server 时，需要您自行安装 Django，并确保版本大于等于 `2.2` 。可通过以下命令安装:

```
pip3 install -U django>=2.2
```

pywebio.platform.django.**webio_view**(*applications*, *cdn=True*, *session_expire_seconds=None*, *session_cleanup_interval=None*, *allowed_origins=None*, *check_origin=None*)

获取在 django 中运行 PyWebIO 任务的视图函数。基于 http 请求与前端进行通讯

关于各参数的详细说明见*pywebio.platform.flask.webio_view()* 的同名参数。

pywebio.platform.django.**wsgi_app**(*applications*, *cdn=True*, *static_dir=None*, *allowed_origins=None*, *check_origin=None*, *session_expire_seconds=None*, *session_cleanup_interval=None*, *debug=False*, *max_payload_size='200M'*, ***django_options*)

将 PyWebIO 应用转换为 Django WSGI 应用

关于各参数的详细说明见*pywebio.platform.django.start_server()* 的同名参数。

pywebio.platform.django.**start_server**(*applications*, *port=8080*, *host=''*, *cdn=True*, *static_dir=None*, *remote_access=False*, *allowed_origins=None*, *check_origin=None*, *session_expire_seconds=None*, *session_cleanup_interval=None*, *debug=False*, *max_payload_size='200M'*, ***django_options*)

启动一个 Django server 将 PyWebIO 应用作为 Web 服务提供。

参数

- **debug** (*bool*) –开启 Django debug mode。参见 Django doc .

- **django_options** –django 应用的其他设置，见 https://docs.djangoproject.com/en/3.0/ref/settings/ . 其中 DEBUG 、ALLOWED_HOSTS 、ROOT_URLCONF 、SECRET_KEY 被 PyWebIO 设置，无法在 django_options 中指定

剩余参数的详细说明见*pywebio.platform.flask.start_server()* 的同名参数。

aiohttp support

使用 aiohttp 后端作为 PyWebIO 应用 Server 时，需要您自行安装 aiohttp，并确保版本大于等于 3.1 。可通过以下命令安装:

```
pip3 install -U aiohttp>=3.1
```

pywebio.platform.aiohttp.**webio_handler**(*applications*, *cdn=True*, *reconnect_timeout=0*, *allowed_origins=None*, *check_origin=None*, *max_payload_size='200M'*, *websocket_settings=None*)

Get the Request Handler coroutine for running PyWebIO applications in aiohttp. The handler communicates with the browser by WebSocket protocol.

The arguments of webio_handler() have the same meaning as for *pywebio.platform.aiohttp.start_server()*

返回

aiohttp Request Handler

pywebio.platform.aiohttp.**start_server**(*applications*, *port=0*, *host=''*, *debug=False*, *cdn=True*, *static_dir=None*, *remote_access=False*, *reconnect_timeout=0*, *allowed_origins=None*, *check_origin=None*, *auto_open_webbrowser=False*, *max_payload_size='200M'*, *websocket_settings=None*, ***aiohttp_settings*)

Start a aiohttp server to provide the PyWebIO application as a web service.

参数

- **websocket_settings** (*dict*) –The parameters passed to the constructor of aiohttp.web.WebSocketResponse. For details, please refer: https://docs.aiohttp.org/en/stable/web_reference.html#websocketresponse
- **aiohttp_settings** –Additional keyword arguments passed to the constructor of aiohttp.web.Application. For details, please refer: https://docs.aiohttp.org/en/stable/web_reference.html#application

剩余参数的详细说明见*pywebio.platform.tornado.start_server()* 的同名参数。

FastAPI/Starlette support

当使用 FastAPI/Starlette 作为 PyWebIO 的后端 server 时，你需要手动安装 fastapi 或 starlette ，另外还需要安装一些其他的依赖库，可以使用以下命令安装:

```
pip3 install -U fastapi starlette uvicorn aiofiles websockets
```

pywebio.platform.fastapi.**webio_routes**(*applications*, *cdn=True*, *reconnect_timeout=0*, *allowed_origins=None*, *check_origin=None*)

获取在 FastAPI/Starlette 中运行 PyWebIO 的路由组件。

服务端使用 WebSocket 协议与浏览器进行通讯。

关于各参数的详细说明见*pywebio.platform.fastapi.start_server()* 的同名参数。

Added in version 1.3.

返回
FastAPI/Starlette routes

pywebio.platform.fastapi.**asgi_app**(*applications*, *cdn=True*, *reconnect_timeout=0*, *static_dir=None*, *debug=False*, *allowed_origins=None*, *check_origin=None*)

将 PyWebIO 应用转换为 starlette/Fastapi ASGI 应用

如果你需要自己托管静态文件，请使用*pywebio.platform.fastapi.webio_routes()*

关于各参数的详细说明见*pywebio.platform.fastapi.start_server()* 的同名参数。

Example

与 `FastAPI.mount()` 一起使用以将 pywebio 作为子应用包含到现有的 Starlette/FastAPI 应用程序中:

```
from fastapi import FastAPI
from pywebio.platform.fastapi import asgi_app
from pywebio.output import put_text
app = FastAPI()
subapp = asgi_app(lambda: put_text("hello from pywebio"))
app.mount("/pywebio", subapp)
```

Returns
Starlette/Fastapi ASGI app

Added in version 1.3.

pywebio.platform.fastapi.**start_server**(*applications*, *port=0*, *host=''*, *cdn=True*, *reconnect_timeout=0*, *static_dir=None*, *remote_access=False*, *debug=False*, *allowed_origins=None*, *check_origin=None*, *auto_open_webbrowser=False*, *max_payload_size='200M'*, ***uvicorn_settings*)

启动一个 FastAPI/Starlette server 将 PyWebIO 应用作为 Web 服务提供。

参数

- **debug** (*bool*) –发生异常时是否打印调用栈。
- **uvicorn_settings** –传递给 `uvicorn.run()` 的额外关键字参数可设置项参考: https://www.uvicorn.org/settings/

剩余参数的详细说明见*pywebio.platform.tornado.start_server()* 的同名参数。

Added in version 1.3.

4.5.3 其他

pywebio.**config**(*, *title=None*, *description=None*, *theme=None*, *js_code=None*, *js_file=[]*, *css_style=None*, *css_file=[]*, *manifest=True*)

PyWebIO 应用配置

参数

- **title** (*str*) –应用标题
- **description** (*str*) –应用简介

- **theme** (*str*) –主题设置。可用主题有：`dark`, `sketchy`, `minty`, `yeti` 。同样可以使用环境变量 `PYWEBIO_THEME` 来设置主题 (有更高优先级)。

  主题预览

  dark 主题更改自 bootstrap-dark ， sketchy, minty 和 yeti 主题来自 bootswatch 。
- **js_code** (*str*) –需要注入到页面上的 Javascript 代码
- **js_file** (*str/list*) –需要添加到页面上的 Javascript 脚本文件，可以是文件的 URL 或 URL 列表。
- **css_style** (*str*) –需要注入到页面上的 CSS 样式规则。
- **css_file** (*str/list*) –需要添加到页面上的 CSS 样式文件，可以是文件的 URL 或 URL 列表。
- **manifest** (*bool/dict*) –Web application manifest configuration. This feature allows you to add a shortcut to the home screen of your mobile device, and launch the app like a native app. If set to `True`, the default manifest will be used. You can also specify the manifest content in dict. If `False`, the manifest will be disabled.

  Currently, the icons field of the manifest is not supported. Instead, you can use the `icon` field to specify the icon url.

`config()` 可以通过两种方式使用：直接调用或作为装饰器使用。如果直接调用 `config()` ，将会作用于全局；如果使用装饰器，配置进将作用于被装饰的 PyWebIO 应用函数:

```
config(title="My application")  # global configuration

@config(css_style="* { color:red }")  # only works on this application
def app():
    put_text("hello PyWebIO")
```

备注： The configuration will affect all sessions

`title` 和 `description` 被用来设置 SEO(在被搜索引擎索引时提供的网页信息)。如果没有提供 `title` 或 `description` ，PyWebIO 默认会将任务函数的 函数注释 作为 SEO 信息:

```
def app():
    """应用标题

    应用简介...
    (应用简介和标题之间使用一个空行分隔)
    """
    pass
```

以上代码等价于:

```
@config(title="Application title", description="Application description...")
def app():
    pass
```

Added in version 1.4.

在 1.5 版本发生变更: add `theme` parameter

`pywebio.platform.`**run_event_loop**(*debug=False*)

运行 asyncio 事件循环

See also: *Integration coroutine-based session with Web framework*

参数

debug –Set the debug mode of the event loop. See also: https://docs.python.org/3/library/asyncio-dev.html#asyncio-debug-mode

4.6 pywebio.pin —持续性输入

pin == Persistent input == Pinning input widget to the page

4.6.1 Overview

我们已经知道，PyWebIO 中的输入函数是阻塞式的，输入表单会在成功提交后被销毁。在大多数场景下，使用这种方式接收用户输入已经够用了。但在一些场景下，你或许希望输入表单在提交后不消失，并且可以继续接收输入。

所以，PyWebIO 提供了 `pin` 模块来实现持续性输入。

`pin` 模块主要有 3 部分内容:

首先，`pin` 模块提供了一些 pin 组件 (widgets)。Pin 组件和 `pywebio.output` 模块中的输出组件并没有什么不同，只不过它还可以接收输入。

以下代码输出了一个最基本的文本框 pin 组件:

```
put_input('input', label='This is a input widget')
```

实际上, pin 组件函数的调用方式和输出函数是一致的，你可以将其作为组合输出的一部分, 也可以将其输出到某个 scope 中:

```
put_row([
    put_input('input'),
    put_select('select', options=['A', 'B', 'C'])
])

with use_scope('search-area'):
    put_input('search', placeholder='Search')
```

然后，你可以使用*pin* 对象来获取 pin 组件的值:

```
put_input('pin_name')
put_buttons(['Get Pin Value'], lambda _: put_text(pin.pin_name))
```

Pin 组件函数的第一个参数为 pin 组件的 `name` 。你可以使用*pin* 对象的同名属性来获取对应 pin 组件的当前值。

另外，*pin* 对象同样支持以索引的方式获取 pin 组件的值，即:

```
pin['pin_name'] == pin.pin_name
```

Pin 模块中还有两个有用的函数: *pin_wait_change()* 和*pin_update()*。

由于 pin 组件输出函数是非阻塞的，所以使用*pin_wait_change()* 来等待一组 pin 组件的值发生变化，它是一个阻塞式函数。

pin_update() 可以用来更新 pin 组件的输出属性。

4.6.2 Pin widgets

Each pin widget function corresponds to an input function of *input* module.

Pin 组件函数支持大多数对应的输入函数的参数。这里列举了两者之间的一些不同：

- Pin 组件函数的第一个参数始终是 Pin 组件的 `name`，且当输出了同名的 pin 组件时，旧的 pin 组件会不可用。
- Pin functions don't support the `on_change` and `validate` callbacks, and the `required` parameter. (There is a *pin_on_change()* function as an alternative to `on_change`)
- Pin 组件函数多了用于输出控制的 `scope` 和 `position` 参数。

pywebio.pin.**put_input**(*name: str*, *type: str = 'text'*, ***, *label: str = ''*, *value: str | None = None*, *placeholder: str | None = None*, *readonly: bool | None = None*, *datalist: List[str] | None = None*, *help_text: str | None = None*, *scope: str | None = None*, *position: int = -1*) → Output

输出一个文本输入组件。参见*pywebio.input.input()*

pywebio.pin.**put_textarea**(*name: str*, ***, *label: str = ''*, *rows: int = 6*, *code: bool | Dict | None = None*, *maxlength: int | None = None*, *minlength: int | None = None*, *value: str | None = None*, *placeholder: str | None = None*, *readonly: bool | None = None*, *help_text: str | None = None*, *scope: str | None = None*, *position: int = -1*) → Output

输出一个文本域输入组件。参见*pywebio.input.textarea()*

pywebio.pin.**put_select**(*name: str*, *options: List[Dict[str, Any] | Tuple | List | str] | None = None*, ***, *label: str = ''*, *multiple: bool | None = None*, *value: List | str | None = None*, *native: bool | None = None*, *help_text: str | None = None*, *scope: str | None = None*, *position: int = -1*) → Output

输出一个下拉选择输入组件。参见*pywebio.input.select()*

备注： Unlike *pywebio.input.select()*, when `multiple=True` and the user is using PC/macOS, *put_select()* will use bootstrap-select by default. Setting `native=True` will force PyWebIO to use native select component on all platforms and vice versa.

---

pywebio.pin.**put_checkbox**(*name: str*, *options: List[Dict[str, Any] | Tuple | List | str] | None = None*, ***, *label: str = ''*, *inline: bool | None = None*, *value: List | None = None*, *help_text: str | None = None*, *scope: str | None = None*, *position: int = -1*) → Output

输出一个多选框组件。参见*pywebio.input.checkbox()*

pywebio.pin.**put_radio**(*name: str*, *options: List[Dict[str, Any] | Tuple | List | str] | None = None*, ***, *label: str = ''*, *inline: bool | None = None*, *value: str | None = None*, *help_text: str | None = None*, *scope: str | None = None*, *position: int = -1*) → Output

输出一个单选按钮组件。参见*pywebio.input.radio()*

pywebio.pin.**put_slider**(*name: str*, ***, *label: str = ''*, *value: int | float = 0*, *min_value: int | float = 0*, *max_value: int | float = 100*, *step: int = 1*, *required: bool | None = None*, *help_text: str | None = None*, *scope: str | None = None*, *position: int = -1*) → Output

输出一个滑块输入组件。参见*pywebio.input.slider()*

pywebio.pin.**put_actions**(*name: str*, ***, *label: str = ''*, *buttons: List[Dict[str, Any] | Tuple | List | str] | None = None*, *help_text: str | None = None*, *scope: str | None = None*, *position: int = -1*) → Output

输出一组 action 按钮。参见*pywebio.input.actions()*

不像 actions() 函数，put_actions() 不会提交任何表单，它只会设置 pin 组件的值。put_actions() 只接受‘submit’类型的按钮。

Added in version 1.4.

pywebio.pin.**put_file_upload**(*name: str*, *, *label: str = ''*, *accept: List | str | None = None*, *placeholder: str = 'Choose file'*, *multiple: bool = False*, *max_size: int | str = 0*, *max_total_size: int | str = 0*, *help_text: str | None = None*, *scope: str | None = None*, *position: int = -1*) → Output

Output a file uploading widget. Refer to: *pywebio.input.file_upload()*

4.6.3 Pin utils

pywebio.pin.**pin**

获取与设置 Pin 组件的值

You can use attribute or key index of pin object to get the current value of a pin widget. By default, when accessing the value of a widget that does not exist, it returns None instead of throwing an exception. You can enable the error raising by pin.use_strict() method.

还可以使用 pin 对象设置 pin 组件的值:

```
put_input('counter', type='number', value=0)

while True:
    pin.counter = pin.counter + 1  # Equivalent to: pin['counter'] = pin['counter
↪'] + 1
    time.sleep(1)
```

注：当使用基于协程的会话 时，你需要使用 await pin.name (或 await pin['name']) 语法来获取 pin 组件的值。

Use pin.pin.use_strict() to enable strict mode for getting pin widget value. An AssertionError will be raised when try to get value of pin widgets that are currently not in the page.

pywebio.pin.**pin_wait_change**(**names*, *timeout: int | None = None*)

pin_wait_change() 监听一组 pin 组件，当其中任意一个的值发生变化后，函数返回发生变化的组件的 name 和值。

参数

- **names** (*str*) –Pin 组件 name 列表
- **timeout** (*int/None*) –当 timeout 为正数时，pin_wait_change() 会最多阻塞 timeout 秒然后返回 None 如果这段时间内监听的 pin 组件的值没有发生变化的话。将 timeout 设置为 None （默认）来关闭超时。

Return dict/None

{"name": 发生变化的 pin 组件的 name, "value": 发生变化的 pin 组件的当前值}，当超时发生后，返回 None

Example:

```
put_input('a', type='number', value=0)
put_input('b', type='number', value=0)

while True:
    changed = pin_wait_change('a', 'b')
```

(续下页)

(接上页)

```
    with use_scope('res', clear=True):
        put_code(changed)
        put_text("a + b = %s" % (pin.a + pin.b))
```

这里 有一个使用*pin_wait_change()* 实现 markdown 实时预览的 demo。

注：使用*pin* 对象或*pin_update()* 更新 pin 组件的值不会引发*pin_wait_change()* 返回

当使用基于协程的会话 时，你需要使用 `await pin_wait_change()` 语法来调用此函数。

pywebio.pin.**pin_update**(*name: str*, ***spec*)

更新 pin 组件的输出属性。

参数

- **name** (*str*) –目标 pin 组件的 `name` 。
- **spec** –需要被更新的 pin 组件的参数。注意以下参数无法被更新：`type`, `name`, `code`, `multiple`

pywebio.pin.**pin_on_change**(*name: str*, *onchange: Callable[[Any], None] | None = None*, *clear: bool = False*, *init_run: bool = False*, ***callback_options*)

Bind a callback function to pin widget, the function will be called when user change the value of the pin widget.

The `onchange` callback is invoked with one argument, the changed value of the pin widget. You can bind multiple functions to one pin widget, those functions will be invoked sequentially (default behavior, can be changed by `clear` parameter).

参数

- **name** (*str*) –pin widget name
- **onchange** (*callable*) –callback function
- **clear** (*bool*) –whether to clear the previous callbacks bound to this pin widget. If you just want to clear callbacks and not set new callback, use `pin_on_change(name, clear=True)`.
- **init_run** (*bool*) –whether to run the `onchange` callback once immediately before the pin widget changed. This parameter can be used to initialize the output.
- **callback_options** –Other options of the `onclick` callback. Refer to the `callback_options` parameter of *put_buttons()*

Added in version 1.6.

4.7 高级特性

本部分介绍 PyWebIO 的高级特性。

4.7.1 使用 start_server() 启动多应用

start_server() 接收一个函数作为 PyWebIO 应用，另外，*start_server()* 还支持传入函数列表或字典，从而启动多个 PyWebIO 应用，应用之间可以通过*go_app()* 或*put_link()* 进行跳转:

```
def task_1():
    put_text('task_1')
    put_buttons(['Go task 2'], [lambda: go_app('task_2')])

def task_2():
    put_text('task_2')
    put_buttons(['Go task 1'], [lambda: go_app('task_1')])

def index():
    put_link('Go task 1', app='task_1')  # Use `app` parameter to specify the task␣
↪name
    put_link('Go task 2', app='task_2')

# equal to `start_server({'index': index, 'task_1': task_1, 'task_2': task_2})`
start_server([index, task_1, task_2])
```

当*start_server()* 的第一个参数的类型为字典时，字典键为应用名，类型为列表时，函数名为应用名。

可以通过 app URL 参数选择要访问的应用 (例如使用 `http://host:port/?app=foo` 来访问 `foo` 应用)，为提供了 app URL 参数时默认使用运行 `index` 应用，当 `index` 应用不存在时，PyWebIO 会提供一个默认的索引页作为 `index` 应用。

4.7.2 与 Web 框架整合

可以将 PyWebIO 应用集成到现有的 Python Web 项目中，PyWebIO 应用与 Web 项目共用一个 Web 框架。目前支持与 Flask、Tornado、Django、aiohttp 和 FastAPI(Starlette) Web 框架的集成。

不同 Web 框架的集成方法如下:

Tornado

Flask

Django

aiohttp

FastAPI/Starlette

Tornado

使用*pywebio.platform.tornado.webio_handler()* 来获取在 Tornado 中运行 PyWebIO 应用的 WebSocketHandler 类:

```
import tornado.ioloop
import tornado.web
from pywebio.platform.tornado import webio_handler

class MainHandler(tornado.web.RequestHandler):
    def get(self):
        self.write("Hello, world")

if __name__ == "__main__":
    application = tornado.web.Application([
```

(续下页)

(接上页)

```
        (r"/", MainHandler),
        (r"/tool", webio_handler(task_func)),  # `task_func` is PyWebIO task function
    ])
    application.listen(port=80, address='localhost')
    tornado.ioloop.IOLoop.current().start()
```

以上代码将 PyWebIO 应用的 `WebSocketHandler` 绑定到了 `/tool` 路径下。启动 Tornado 后，访问 `http://localhost/tool` 即可打开 PyWebIO 应用。

注意： 当使用 Tornado 后端时，PyWebIO 使用 WebSocket 协议和浏览器进行通讯，如果你的 Tornado 应用处在反向代理 (比如 Nginx) 之后，可能需要特别配置反向代理来支持 WebSocket 协议，这里有一个 Nginx 配置 WebSocket 的例子。

Flask

使用*pywebio.platform.flask.webio_view()* 来获取在 Flask 中运行 PyWebIO 应用的视图函数:

```
from pywebio.platform.flask import webio_view
from flask import Flask

app = Flask(__name__)

# `task_func` is PyWebIO task function
app.add_url_rule('/tool', 'webio_view', webio_view(task_func),
            methods=['GET', 'POST', 'OPTIONS'])  # need GET,POST and OPTIONS methods

app.run(host='localhost', port=80)
```

以上代码使用添加了一条路由规则将 PyWebIO 应用的视图函数绑定到 `/tool` 路径下。启动 Flask 应用后，访问 `http://localhost/tool` 即可打开 PyWebIO 应用

Django

使用*pywebio.platform.django.webio_view()* 来获取在 Django 中运行 PyWebIO 应用的视图函数:

```
# urls.py

from django.urls import path
from pywebio.platform.django import webio_view

# `task_func` is PyWebIO task function
webio_view_func = webio_view(task_func)

urlpatterns = [
    path(r"tool", webio_view_func),
]
```

以上代码使用添加了一条路由规则将 PyWebIO 应用的视图函数绑定到 `/tool` 路径下。启动 Django 应用后，访问 `http://localhost/tool` 即可打开 PyWebIO 应用

aiohttp

使用*pywebio.platform.aiohttp.webio_handler()* 来获取在 aiohttp 中运行 PyWebIO 应用的 Request Handler 协程:

```
from aiohttp import web
from pywebio.platform.aiohttp import webio_handler

app = web.Application()
# `task_func` is PyWebIO task function
app.add_routes([web.get('/tool', webio_handler(task_func))])

web.run_app(app, host='localhost', port=80)
```

启动 aiohttp 应用后，访问 `http://localhost/tool` 即可打开 PyWebIO 应用

注意： 当使用 aiohttp 后端时，PyWebIO 使用 WebSocket 协议和浏览器进行通讯，如果你的 aiohttp 应用处在反向代理 (比如 Nginx) 之后，可能需要特别配置反向代理来支持 WebSocket 协议，这里有一个 Nginx 配置 WebSocket 的例子。

FastAPI/Starlette

使用*pywebio.platform.fastapi.webio_routes()* 来获取在 FastAPI/Starlette 中运行 PyWebIO 应用的路由组件，你可以将其挂载在到 FastAPI/Starlette 应用中。

FastAPI:

```
from fastapi import FastAPI
from pywebio.platform.fastapi import webio_routes

app = FastAPI()

@app.get("/app")
def read_main():
    return {"message": "Hello World from main app"}

# `task_func` is PyWebIO task function
app.mount("/tool", FastAPI(routes=webio_routes(task_func)))
```

Starlette:

```
from starlette.applications import Starlette
from starlette.responses import JSONResponse
from starlette.routing import Route, Mount
from pywebio.platform.fastapi import webio_routes

async def homepage(request):
    return JSONResponse({'hello': 'world'})

app = Starlette(routes=[
    Route('/', homepage),
    Mount('/tool', routes=webio_routes(task_func))  # `task_func` is PyWebIO task␣
↪function
])
```

使用 `uvicorn <module>:app` 启动 server 后，访问 `http://localhost:8000/tool/` 即可打开 PyWebIO 应用

参见: FastAPI doc , Starlette doc

注意： 当使用 FastAPI 或 Starlette 后端时，PyWebIO 使用 WebSocket 协议和浏览器进行通讯，如果你的 aiohttp 应用处在反向代理 (比如 Nginx) 之后，可能需要特别配置反向代理来支持 WebSocket 协议，这里有一个 Nginx 配置 WebSocket 的例子。

Notes

生产环境部署

在生产环境中，你可能会使用一些 WSGI/ASGI 服务器（如 uWSGI、Gunicorn、Uvicorn）部署 Web 应用程序。由于 PyWebIO 应用程序会在进程中存储会话状态，当使用基于 HTTP 的会话（使用 Flask 和 Django 后端时）并生成多个进程来处理请求时，请求可能会被分发到错误的进程中。因此，在使用基于 HTTP 的会话时，只能启动一个进程来处理请求。

如果仍然希望使用多进程来提高并发，一种方式是使用 Uvicorn+FastAPI，或者你也可以启动多个 Tornado/aiohttp 进程，并在它们之前添加外部的负载均衡软件（如 HAProxy 或 nginx）。这些后端使用 WebSocket 协议与浏览器进行通信，所以不存在上述问题。

PyWebIO 静态资源的托管

PyWebIO 默认使用 CDN 来获取前端的静态资源，如果要将 PyWebIO 应用部署到离线环境中，需要自行托管静态文件，并将 `webio_view()` 或 `webio_handler()` 的 `cdn` 参数设置为 `False` 。

`cdn=False` 时需要将静态资源托管在和 PyWebIO 应用同级的目录下。同时，也可以通过 `cdn` 参数直接设置 PyWebIO 静态资源的 URL 目录。

PyWebIO 的静态文件的路径保存在 `pywebio.STATIC_PATH` 中，可使用命令 `python3 -c "import pywebio; print(pywebio.STATIC_PATH)"` 将其打印出来。

备注： 使用 `start_server()` 启动的应用，如果将 `cdn` 参数设置为 `False` ，会自动启动一个本地的静态资源托管服务，无需手动托管。

4.7.3 基于协程的会话

在大部分情况下，你并不需要使用基于协程的会话。PyWebIO 中所有仅用于协程会话的函数或方法都在文档中均有特别说明。

PyWebIO 的会话实现默认是基于线程的，用户每打开一个和服务端的会话连接，PyWebIO 会启动一个线程来运行任务函数。除了基于线程的会话，PyWebIO 还提供了基于协程的会话。基于协程的会话接受协程函数作为任务函数。

基于协程的会话为单线程模型，所有会话都运行在一个线程内。对于 IO 密集型的任务，协程比线程占用更少的资源同时又拥有媲美于线程的性能。另外，协程的上下文切换具有可预测性，能够减少程序同步与加锁的需要，可以有效避免大多数临界区问题。

使用协程会话

要使用基于协程的会话，需要使用 async 关键字将任务函数声明为协程函数，并使用 await 语法调用 PyWebIO 输入函数:

```
from pywebio.input import *
from pywebio.output import *
from pywebio import start_server

async def say_hello():
    name = await input("what's your name?")
    put_text('Hello, %s' % name)

start_server(say_hello, auto_open_webbrowser=True)
```

在协程任务函数中，也可以使用 await 调用其他协程或标准库 asyncio 中的可等待对象 (awaitable objects):

```
import asyncio
from pywebio import start_server

async def hello_word():
    put_text('Hello ...')
    await asyncio.sleep(1)  # await awaitable objects in asyncio
    put_text('... World!')

async def main():
    await hello_word()  # await coroutine
    put_text('Bye, bye')

start_server(main, auto_open_webbrowser=True)
```

注意: 在基于协程的会话中，*pywebio.input* 模块中的定义输入函数都需要使用 await 语法来获取返回值，忘记使用 await 将会是在使用基于协程的会话时常出现的错误。

其他在协程会话中也需要使用 await 语法来进行调用函数有:

- *pywebio.session.run_asyncio_coroutine(coro_obj)*
- *pywebio.session.eval_js(expression)*

警告: 虽然 PyWebIO 的协程会话兼容标准库 asyncio 中的 awaitable objects ，但 asyncio 库不兼容 PyWebIO 协程会话中的 awaitable objects .

也就是说，无法将 PyWebIO 中的 awaitable objects 传入 asyncio 中的接受 awaitable objects 作为参数的函数中，比如如下调用是 **不被支持的**

```
await asyncio.shield(pywebio.input())
await asyncio.gather(asyncio.sleep(1), pywebio.session.eval_js('1+1'))
task = asyncio.create_task(pywebio.input())
```

协程会话的并发

在基于协程的会话中，你可以启动线程，但是无法在其中调用 PyWebIO 交互函数（*register_thread()* 在协程会话中不可用）。但你可以使用*run_async(coro)* 来异步执行一个协程对象，新协程内可以使用 PyWebIO 交互函数:

```
from pywebio import start_server
from pywebio.session import run_async

async def counter(n):
    for i in range(n):
        put_text(i)
        await asyncio.sleep(1)

async def main():
    run_async(counter(10))
    put_text('Main coroutine function exited.')


start_server(main, auto_open_webbrowser=True)
```

run_async(coro) 返回一个*TaskHandler* ，通过该*TaskHandler* 可以查询协程运行状态和关闭协程。

会话的结束

和基于线程的会话一样，当用户关闭浏览器页面后，会话也随之关闭。

浏览器页面关闭后，当前会话内还未返回的 PyWebIO 输入函数调用将抛出 `SessionClosedException` 异常，之后对于 PyWebIO 交互函数的调用将会产生 `SessionNotFoundException` 或 `SessionClosedException` 异常。

协程会话也同样支持使用*defer_call(func)* 来设置会话结束时需要调用的函数。

协程会话与 Web 框架集成

基于协程的会话同样可以与 Web 框架进行集成，只需要在原来传入任务函数的地方改为传入协程函数即可。

但当前在使用基于协程的会话集成进 Flask 或 Django 时，存在一些限制：

一是协程函数内还无法直接通过 `await` 直接等待 asyncio 库中的协程对象，目前需要使用*run_asyncio_coroutine()* 进行包装。

二是，在启动 Flask/Django 这类基于线程的服务器之前需要启动一个单独的线程来运行事件循环。

使用基于协程的会话集成进 Flask 的示例:

```
import asyncio
import threading
from flask import Flask, send_from_directory
from pywebio import STATIC_PATH
from pywebio.output import *
from pywebio.platform.flask import webio_view
from pywebio.platform import run_event_loop
from pywebio.session import run_asyncio_coroutine

async def hello_word():
    put_text('Hello ...')
```

(续下页)

(接上页)

```
    await run_asyncio_coroutine(asyncio.sleep(1))  # can't just "await asyncio.
↪sleep(1)"
    put_text('... World!')

app = Flask(__name__)
app.add_url_rule('/hello', 'webio_view', webio_view(hello_word),
                 methods=['GET', 'POST', 'OPTIONS'])

# thread to run event loop
threading.Thread(target=run_event_loop, daemon=True).start()
app.run(host='localhost', port=80)
```

最后，使用 PyWebIO 编写的协程函数不支持 Script 模式，总是需要使用 `start_server` 来启动一个服务或者集成进 Web 框架来调用。

4.8 第三方库生态

4.8.1 构建 stand-alone App

PyInstaller 用于将一个 Python 应用及其依赖打包到文件夹或可执行文件中，用户可以在不安装 Python 解释器以及任何模块的情况下运行打包后的应用程序。

可以使用 PyInstaller 来将 PyWebIO 应用打包成一个单独的可执行文件或文件夹:

1. 创建 pyinstaller spec (specification) 文件:

   ```
   pyi-makespec <options> app.py
   ```

   你需要将 `app.py` 替换成你 PyWebIO 应用的文件名。

2. Only for PyWebIO before v1.8: Edit the spec file, change the `datas` parameter of `Analysis`:

   ```
   from pywebio.utils import pyinstaller_datas

   a = Analysis(
       ...
       datas=pyinstaller_datas(),
       ...
   ```

3. 使用 spec 文件来构建可执行文件:

   ```
   pyinstaller app.spec
   ```

如果你希望生成一个单独的可执行文件而不是文件夹，你需要在第一步时传入 `--onefile` 选项。

更多 PyInstaller 用法请见: https://pyinstaller.readthedocs.io/en/stable/spec-files.html

4.8.2 数据可视化

PyWebIO 支持使用第三方库进行数据可视化

Bokeh

Bokeh 是一个支持创建实时交互的数据可视化库。

在 PyWebIO 会话中调用 `bokeh.io.output_notebook(notebook_type='pywebio')` 来设置 Bokeh 输出到 PyWebIO:

```
from bokeh.io import output_notebook
from bokeh.io import show

output_notebook(notebook_type='pywebio')
fig = figure(...)
...
show(fig)
```

相应 demo 见 bokeh demo

除了创建普通图表，Bokeh 还可以通过启动 Bokeh server 来显示 Bokeh app，Bokeh app 支持向图表的添加按钮、输入框等交互组件，并向组件注册 Python 回调，从而创建可以与 Python 代码交互的图表。

在 PyWebIO 中，你也可以使用 `bokeh.io.show()` 来显示一个 Bokeh App，代码示例见 bokeh_app.py。

备注： Bokeh App 当前仅支持默认的 Tornado 后端

pyecharts

pyecharts 是一个使用 Python 创建 Echarts 可视化图表的库。

在 PyWebIO 中使用*put_html()* 可以输出 pyecharts 库创建的图表:

```
# `chart` is pyecharts chart instance
pywebio.output.put_html(chart.render_notebook())
```

相应 demo 见 pyecharts demo

plotly

plotly.py 是一个非常流行的 Python 数据可视化库，可以生成高质量的交互式图表。

PyWebIO 支持输出使用 plotly 库创建的图表。使用方式为在 PyWebIO 会话中调用:

```
# `fig` is plotly chart instance
html = fig.to_html(include_plotlyjs="require", full_html=False)
pywebio.output.put_html(html)
```

相应 demo 见 plotly demo

pyg2plot

pyg2plot 是一个使用 Python 创建 G2Plot 可视化图表的库。

PyWebIO 支持输出使用 pyg2plot 库创建的图表。使用方式为在 PyWebIO 会话中调用:

```
# `chart` 为 pyg2plot 图表实例
pywebio.output.put_html(chart.render_notebook())
```

相应 demo 见 plotly demo

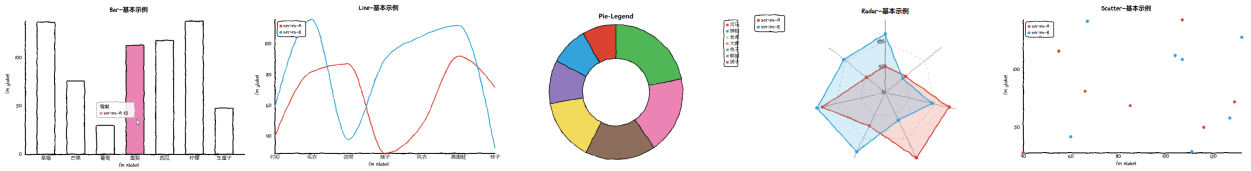
cutecharts.py

cutecharts.py 是一个可以创建具有卡通风格的可视化图表的 python 库。底层使用了 chart.xkcd Javascript 库。

在 PyWebIO 中使用`put_html()`可以输出 cutecharts.py 库创建的图表:

```
# `chart` is cutecharts chart instance
pywebio.output.put_html(chart.render_notebook())
```

相应 demo 见 cutecharts demo



4.9 Cookbook

参见:

PyWebIO Battery

- *Interaction related*
  - *Equivalent to "Press any key to continue"*
  - *Output pandas dataframe*
  - *Output Matplotlib figure*
  - *Add new syntax highlight for code output*
- *Web application related*
  - *Add Google AdSense/Analytics code*
  - *Refresh page on connection lost*

4.9.1 Interaction related

Equivalent to "Press any key to continue"

```
actions(buttons=["Continue"])
```

Output pandas dataframe

```
import numpy as np
import pandas as pd

df = pd.DataFrame(np.random.randn(6, 4), columns=list("ABCD"))
put_html(df.to_html(border=0))
```

参见:

pandas.DataFrame.to_html —pandas documentation

Output Matplotlib figure

Instead of using `matplotlib.pyplot.show()`, to show matplotlib figure in PyWebIO, you need to save the figure to in-memory buffer fist and then output the buffer via *`pywebio.output.put_image()`*:

```
import matplotlib
import matplotlib.pyplot as plt
import io
import pywebio

matplotlib.use('agg')  # required, use a non-interactive backend

fig, ax = plt.subplots()  # Create a figure containing a single axes.
ax.plot([1, 2, 3, 4], [1, 4, 2, 3])  # Plot some data on the axes.

buf = io.BytesIO()
fig.savefig(buf)
pywebio.output.put_image(buf.getvalue())
```

The `matplotlib.use('agg')` is required so that the server does not try to create (and then destroy) GUI windows that will never be seen.

When using Matplotlib in a web server (multiple threads environment), pyplot may cause some conflicts in some cases, read the following articles for more information:

- Multi Threading in Python and Pyplot | by Ranjitha Korrapati | Medium
- Embedding in a web application server (Flask) —Matplotlib documentation

Add new syntax highlight for code output

When output code via `put_markdown()` or `put_code()`, PyWebIO provides syntax highlight for some common languages. If you find your code have no syntax highlight, you can add the syntax highlighter by two following steps:

1. Go to prismjs CDN page to get your syntax highlighter link.
2. Use *`config(js_file=...)`* to load the syntax highlight module

```
@config(js_file="https://cdn.jsdelivr.net/npm/prismjs@1.23.0/components/prism-diff.
↪min.js")
def main():
    put_code("""
+ AAA
- BBB
CCC
    """.strip(), language='diff')

    put_markdown("""
    ```diff
    + AAA
    - BBB
    CCC
    ```
    """, lstrip=True)
```

4.9.2 Web application related

Add Google AdSense/Analytics code

When you setup Google AdSense/Analytics, you will get a javascript file and a piece of code that needs to be inserted into your application page, you can use *`pywebio.config()`* to inject js file and code to your PyWebIO application:

```
from pywebio import start_server, output, config

js_file = "https://www.googletagmanager.com/gtag/js?id=G-xxxxxxx"
js_code = """
window.dataLayer = window.dataLayer || [];
function gtag(){dataLayer.push(arguments);}
gtag('js', new Date());

gtag('config', 'G-xxxxxxx');
"""

@config(js_file=js_file, js_code=js_code)
def main():
    output.put_text("hello world")

start_server(main, port=8080)
```

Refresh page on connection lost

Add the following code to the beginning of your PyWebIO application main function:

```
session.run_js('WebIO._state.CurrentSession.on_session_close(()=>{setTimeout(()=>
↪location.reload(), 4000})')
```

4.10 Release notes

4.10.1 What's new in PyWebIO 1.8

2023/4/10

Highlights

- Add datatable widget (*put_datatable()*)
- Build reliable message transmission over HTTP-based backends (Flask and Django)

Backwards-incompatible changes

- When use *put_loading()* as context manager, the output inside the context will also been removed after the context block exits.

Detailed changes

- Add *put_file_upload()* pin widget.
- Add WPA support (via *config(manifest)*), so PyWebIO apps can be launched like a native app on mobile devices.
- Add type hints to all public functions (#501, thanks to 叶子)
- Add Uzbek language support for UI labels (#539, thanks to Ulugbek)
- Remove the `NullHandler()` logging handler added to `pywebio` logger, so the exception log from PyWebIO can be output by default.
- Add `max_payload_size` param to `start_server()` and `webio_handler()` for aiohttp and fastapi backends.
- When `tdata` of `put_table()` is list of dict, `header` parameter is not mandatory anymore.
- Add pyinstaller hook, so PyWebIO apps can be packaged to executable file with pyinstaller without any extra configuration.
- No traceback expose to user in production environment (`start_server(debug=False)`, the default setting).

Bug fix

- Fix memory leak after close session (#545)

v1.8.1 (2023/4/16)

- fix (#568) global `config(title)` don't work
- fix (#569) cell_content_bar shown error

v1.8.2 (2023/4/22)

- fix (#570) `flex` column config of `put_datatable()` don't work
- fix json encode error when use tuple as key in `column_args` of `put_datatable()`

v1.8.3 (2023/10/30)

- fix (#614) `input_group()` return invalid result when canceled

4.10.2 What's new in PyWebIO 1.7

2022/10/17

Highlights

- add session reconnect to aiohttp and fastapi backends (now, all platforms support session reconnect)

Detailed changes

- auto use local static when CND is not available
- refine *`use_scope(clear=True)`* to avoid page flashing

Bug fix

- fix: *`textarea(code=True, required=True)`* can't submit
- fix: auto hold don't work on script mode
- fix (#389): `put_select()` was hidden by `put_tabs()`
- fix: `input_update(datalist)` don't work when `datalist` is not provided in `input()`
- fix (#459): code textarea onchange fired when set value
- fix (#453): `put_table()` error when table data is empty with rich header
- fix load old static resource after version upgrade
- fix cancel type raise error in single `action()`
- fix (#377): error on nested `onchange` callback

- fix (#468): can't reset `select()`
- fix `set_env(output_animation=False)` don't work for image

4.10.3 What's new in PyWebIO 1.6

2022/3/23

Highlights

- add *pywebio.pin.pin_on_change()*

Detailed changes

- use bootstrap-select to provide more user-friendly select input
- add `pin.pin.use_strict()` to enable strict mode for getting pin widget value
- Persian language support for default labels, thanks to Pikhosh
- add color input type (#310)
- add input check on number and float type input

Bug fix

- fix: uncaught `SessionClosedException` in callback of thread-based session
- fix(#313): slider value label don't sync when set value

v1.6.1 (2022/5/22)

- fix (#380): `put_processbar()` don't work when name contains space
- fix (#385): bootstrap-select issue
- fix (#389): `put_select()` was hidden by `put_tabs()`
- fix auto hold don't work on script mode
- provide a fallback way when CDN is not available

v1.6.2 (2022/7/16)

- fix: plotly.js version error due to outdated CDN link

4.10.4 What's new in PyWebIO 1.5

2021/11/20

Highlights

- theme support via *pywebio.config()*, demo
- deprecate `pywebio.output.output()`, use *pywebio.output.use_scope()* instead (`output()` still work)

Detailed changes

- enable `lstrip` by default in *put_markdown()*, and the behavior of `lstrip` is more clever than previous version. Deprecate `strip_indent` since `lstrip` is sufficient.
- button disabled state support in *pywebio.output.put_buttons()* and *pywebio.output.put_button()*, and button value can be any type
- buttons in *pywebio.input.actions()* support color setting
- russian language support for frontend labels and messages. Thanks to @Priler.
- improve default index page of *pywebio.platform.path_deploy()*: improve pywebio app detection and show app title.
- compatible with latest aiohttp(v3.8)
- enable `websocket_ping_interval` by default in tornado server to avoid idle connections being close in some cloud platform (like heroku)
- exception traceback will be show in page when enable debug
- *slider* input add indicator to show its current value

Bug fix

- deep copy `options` and `buttons` parameters to avoid potential error - 81d57ba4, cb5ac8d5 - e262ea43
- fix page width exceeding screen width (mostly on mobile devices) - 536d09e3
- fix `put_buttons()` issue when buttons have same value - cb5ac8d5
- fix layout issue when use `put_markdown()` - 364059ae
- fix style issue in `put_tabs()` widget - f056f1ac
- fix sibling import issue in `path_deploy()` - 35209a7e
- fix "Address already in use" error when enable remote access in some cases - 8dd9877d

v1.5.1 (2021/12/21)

- fix setitem error of `pin.pin` object - 3f5cf1e5
- fix thread-based session tot closed properly - 22fbbf86..3bc7d36b>
- fix OverflowError on 32-bit Windows - 4ac7f0e5
- fix a sample error from cookbook - 99593db4
- fix spawn 2 remote access processes when enable debug in flask backed - 073f8ace

v1.5.2 (2021/12/30)

- fix #243: thread keep alive after session closed
- fix #247: can't use coroutine callback in `put_button()`

4.10.5 What's new in PyWebIO 1.4

2021/10/4

Highlights

- automatically hold session when needed
- support for binding onclick callback on any output widget

Detailed changes

- migrate to a open-source remote access service
- add `output_max_width` parameter to *`set_env()`*
- can use `Esc`/`F11` to toggle fullscreen of codemirror textarea
- *`pin_wait_change()`* support `timeout` parameter
- add *`pywebio.config()`*
- add *`pywebio.output.put_button()`*
- add *`pywebio.pin.put_actions()`*
- rearrange document

Bug fix

- fix(#148): form can't be submit after validation failed - e262ea43
- fix some codemirror issues: codemirror refresh and mode auto load - b7957891, 50cc41a9
- fix: `run_js()` return `None` when empty-value - 89ce352d
- fix: whole output crash when a sub output fail - 31b26d09

4.10.6 What's new in PyWebIO 1.3

2021/6/12

Highlights

- New module *pin* to provide persistent input support.
- Add a remote access service to `start_server()`. See *server mode - User Guide* for detail.
- Add *`input_update()`*, add `onchange` callback in input functions.
- Add support for FastAPI and Starlette.

Detailed changes

- *input* module
  - Add *`input_update()`*, add `onchange` callback in input functions.
  - Add *`pywebio.input.slider()`* to get range input.
- *output* module
  - Mark *`style()`* as deprecated, see *style - User Guide* for new method.
  - Add *`pywebio.output.put_tabs()`* to output tabs.
  - *`put_html()`* adds compatibility with ipython rich output.
  - Add `group` and `outline` parameters in *`put_buttons()`*.
- *session* module
  - Add promise support in *`eval_js()`*.
  - Support config input panel via *`set_env()`*.
- *platform* module
  - Add support for FastAPI and Starlette.
  - Add `wsgi_app()` / `asgi_app()` for Flask/Dgango/FastAPI backend.
  - Add remote access service to `start_server()`
  - Add `max_file_upload`/`payload_size_limit`/`upload_size_limit`/`max_payload_size` parameters to `start_server()`.
- So many other improvements.

Bug fix

- Fix table style.
- Fix large file uploading error.
- Fix server start error when enabled `auto_open_webbrowser`.
- Fix file names overflow in file input.
- Fix *`put_image()`* raise 'unknown file extension' error when use PIL Image as `src`.

- Sanitize the returned `filename` of *`file_upload()`* to avoid interpreting as path accidentally.
- So many other bugs fixed.

4.10.7 What's new in PyWebIO 1.2

2021 3/18

Highlights

- Websocket 连接可以通过在*`start_server()`* 中设定 `reconnect_timeout` 参数来支持连接重连。
- 添加*`path_deploy()`*, *`path_deploy_http()`* 和 `pywebio-path-deploy` 命令来实现从目录中加载运行 PyWebIO 应用。
- 所有的文档和示例都提供了英文版本。
- 为一些输出相关的函数提供上下文管理器的支持，参见*output functions list* 。

Detailed changes

- 添加 `put_info()`,`put_error()`,`put_warning()`,`put_success()` 来显示提示消息。
- 添加 `pywebio.utils.pyinstaller_datas()` 来获得使用 pyinstaller 打包 PyWebIO 应用所需要的数据文件。
- 添加使用 pyg2plot 进行数据可视化的文档。
- `output()` 的 `reset()`,`append()`,`insert()` 方法接受任意类型作为输出内容。
- `start_server()` 中添加 `static_dir` 参数来托管静态文件。
- 废弃 `pywebio.session.get_info()` ，使用*`pywebio.session.info`* 替代
- 当用户使用 IE 浏览器时提示浏览器不被支持。

4.10.8 What's new in PyWebIO 1.1

2021 2/7

距离写下 PyWebIO 的第一行代码过去已经整整一年了 ，2020 年发生了太多的事情，但对我来说又多了一份特殊的意义。新的一年继续努力 ，将 PyWebIO 做得越来越好。

Highlights

- 添加安全性支持: *`put_html()`*, *`put_markdown()`* 中支持使用 `sanitize` 参数开启防 XSS 攻击
- UI 国际化支持
- 添加 SEO 支持: 通过任务函数的注释或 `pywebio.platform.seo()` 来设置 SEO 信息
- CDN 支持，Web 框架整合更加方便，仅需引入一条路由即可
- 应用访问速度提升，不再使用探测请求的方式确定通信协议

Backwards-incompatible changes

- 移除使用 django 和 flask 框架 `start_server()` 中的 `disable_asyncio` 参数
- 废弃 `pywebio.session.data()`，使用*pywebio.session.local* 作为会话本地状态存储对象
- 整合到 Web 框架的应用，访问地址发生变化，参见*Web 框架整合文档*
- *put_scrollable()* 废弃 `max_height` 参数，使用 `height` 替代

Detailed changes

- *put_code()* 支持使用 `rows` 参数限制最大显示行数
- *put_scrollable()* 支持使用 `keep_bottom` 参数设定自动滚动到底部
- *put_markdown()* 支持配置 Markdown 解析参数
- 为*put_code()*, *put_image()*, *put_link()*, *put_row()*, *put_grid()* 中的参数添加转义
- `output()` 的 `reset()`, `append()`, `insert()` 方法接受字符串作为输出内容
- 修复: *file_upload()* 的 `max_size` and `max_total_size` 参数解析错误
- 修复: py3.6 自动打开浏览器失败

4.10.9 What's new in PyWebIO 1.0

2021 1/17

经过快一年的开发，PyWebIO 1.0 终于完成了。与上一版本 v0.3 相比有非常多的变化：

Highlights

- `start_server` 对多任务函数的支持，PyWebIO 应用可以包含多个任务函数，并提供了*go_app()* 用于任务函数之间的跳转
- 不再使用基于锚点的输出控制模型，改用基于 Scope 的模型
- 添加布局支持（*put_grid()*, *put_row()*, *put_column()*）和自定义样式支持（*style()*）
- 添加新的输出函数: *toast()*, *popup()*, *put_widget()*, *put_collapse()*, *put_link()*, *put_scrollable()*, *put_loading()*, `put_processbar()`
- 添加*span()*, `output()` 输出控制函数
- 添加 JS 执行函数: *run_js()*, *eval_js()*
- 更新 UI: 显示输入时，使用浮动式输入框；发生未捕获异常时，前端使用 console 日志记录异常

Backwards-incompatible changes

- 不再使用基于锚点的输出控制模型
- 不支持固定高度的输出区，移除 `pywebio.output.set_output_fixed_height()`
- 移除 `pywebio.output.set_title()` , `pywebio.output.set_auto_scroll_bottom()`，改用*pywebio.session.set_env()* 进行控制
- 移除 `pywebio.output.table_cell_buttons()`，使用*pywebio.output.put_buttons()* 替代

Detailed changes by module

- *input()* 支持 `action` 参数动态设置输入项的值
- *file_upload()* 支持多文件上传，支持限制上传文件大小，添加上传进度显示
- *put_buttons()* 支持指定按钮颜色
- *put_widget()* 、*popup()* 、*put_table()* 将字符串内容不再视作 Html，而是作为纯文本
- *put_text()* 支持输出多个对象
- *put_image()* 支持使用 Url 指定图片

4.10.10 What’s new in PyWebIO 0.3

2020 5/13

Highlights

- 支持输出 bokeh 数据可视化图表，文档
- 添加 `session.get_info()` 获取会话相关信息
- 前端 js 代码迁移 typescript
- *output.put_table()* 支持跨行/列单元格，单元格内容支持使用 `put_xxx` 类输出函数

Detailed changes by module

UI

- 当与服务器连接断开时，点击前端的交互式按钮会报错提示。

pywebio.output

- 锚点名字支持使用空格
- 弃用 `table_cell_buttons()`

4.10.11 What's new in PyWebIO 0.2

2020 4/30

Highlights

- 支持与 Django、aiohttp Web 框架整合
- 支持使用 plotly、pyecharts 等第三方库进行数据可视化
- 与 Web 框架整合时支持同时使用基于线程和协程的会话实现
- 添加*`defer_call()`* 、*`hold()`* 会话控制函数
- 添加*`put_image()`* 输出图像、*`remove(anchor)`* 移除内容
- 加入动画提升 UI 体验
- 添加测试用例，构建 CI 工作流

Detailed changes by module

UI

- 添加元素显示动画
- 页面底部添加 footer

pywebio.input

- *`input_group()`* 添加 `cancelable` 参数来允许用户取消输入
- *`actions()`* 函数 `button` 参数支持 `reset` 和 `cancel` 按钮类型

pywebio.output

- 输出函数使用 `anchor` 参数指定输出锚点时，若锚点已经存在，则将锚点处的内容替换为当前内容。
- `clear_range()` 添加添加锚点存在检查
- *`scroll_to(anchor, position)`* 添加 `position` 参数精细化控制滚动位置

pywebio.platform

- start_server 和 webio_view 、webio_handle 添加跨域支持

pywebio.session

- Session 关闭时，清理更彻底：任何还在进行的 PyWebIO 调用都会抛出 SessionClosedException 异常
- fix: Session 对象构造函数无法识别 functools.partial 处理的任务函数

4.11 pywebio_battery —PyWebIO battery

Utilities that help write PyWebIO apps quickly and easily.

备注： pywebio_battery is an extension package of PyWebIO, you must install it before using it. To install this package, run pip3 install -U pywebio-battery

4.11.1 Functions index

Interaction related

Function name	Description
file_picker	Local file picker
confirm	Confirmation modal
popup_input	Show a form in popup window
redirect_stdout	redirecting stdout to pywebio
run_shell	Run command in shell
put_logbox, *logbox_append*, *logbox_clear*	Logbox widget
put_video	Output video
put_audio	Output audio
wait_scroll_to_bottom	Wait the page is scrolled to bottom

Web application related

Function name	Description
get_all_query, *get_query*	Get URL parameter
set_localstorage, *get_localstorage*	User browser storage
set_cookie, *get_cookie*	Web Cookie
basic_auth, *custom_auth*, *revoke_auth*	Authentication

pywebio_battery.**file_picker**(*path: str*, *multiple: bool = False*, *accept: str | List[str] = ''*, *cancelable: bool = False*, *title: str = 'File Picker'*, *show_hidden_files: bool = False*) → str | List[str] | None

A file picker widget that allows you to select files from the local file system where PyWebIO is running.

参数

- **path** (*str*) –The root path of the file picker. ~ can be used to represent the user's home directory.
- **multiple** (*bool*) –Whether to allow multiple files to be selected.
- **accept** (*str*) –Acceptable file type, case-insensitive. Can be a string or a list of strings. e.g. `accept='pdf'`, `accept=['jpg', 'png']`. Default is to accept any file.
- **cancelable** (*bool*) –Whether to allow the user to cancel the file picker. By default, the user can only close the file picker by selecting the file.
- **title** (*str*) –The title of the file picker popup.
- **show_hidden_files** –Whether to show hidden files/folders.

返回

The selected file path or a list of file paths. `None` if the user cancels the file picker.

```
files = file_picker('.', multiple=True, accept='py')
put_text(files)
```

pywebio_battery.**confirm**(*title: str*, *content: str | Output | Sequence[str | Output] | None = None*, ***, *timeout: int | None = None*) → bool | None

Show a confirmation modal.

参数

- **title** (*str*) –Model title.
- **content** (*list/put_xxx()/str*) –The content of the confirmation modal. Can be a string, the put_xxx() calls , or a list of them.
- **timeout** (*None/float*) –Seconds for operation time out.

返回

Return `True` when the "CONFIRM" button is clicked, return `False` when the "CANCEL" button is clicked, return `None` when a timeout is given and the operation times out.

```
choice = confirm("Delete File", "Are you sure to delete this file?")
put_text("Your choice", choice)
```

pywebio_battery.**popup_input**(*pins: Sequence[Output] | Output*, *title='Please fill out the form below'*, *validate: Callable[[Dict], Tuple[str, str] | None] | None = None*, *popup_size: str = 'normal'*, *cancelable: bool = False*) → dict | None

Show a form in popup window.

参数

- **pins** (*list*) –*pin* widget list. It can also contain ordinary output widgets.
- **title** (*str*) –model title.
- **validate** (*callable*) –validation function for the form. Same as `validate` parameter in *input_group()*
- **popup_size** (*str*) –popup window size. See `size` parameter of *popup()*
- **cancelable** (*bool*) –Whether the form can be cancelled. Default is `False`. If `cancelable=True`, a "Cancel" button will be displayed at the bottom of the form.

返回

return the form value as dict, return `None` when user cancel the form.

```
def check_password(form):
    if len(form['password']) < 6:
        return 'password', 'password length must greater than 6'

form = popup_input(
    [
        put_input("username", label="Username"),
        put_input("password", type=PASSWORD, label="Password"),
        put_info("If you forget your password, please contact the administrator.
↪"),
    ],
    title="Login",
    validate=check_password
)
put_text("Login info:", form)
```

pywebio_battery.**redirect_stdout**(*output_func=functools.partial(<function put_text>, inline=True)*)

Context manager for temporarily redirecting stdout to pywebio.

```
with redirect_stdout():
    print("Hello world.")
```

pywebio_battery.**run_shell**(*cmd: str*, *output_func=functools.partial(<function put_text>, inline=True)*, *encoding='utf8'*) → int

Run command in shell and output the result to pywebio

参数

- **cmd** (*str*) –command to run
- **output_func** (*callable*) –output function, default to `put_text()`. the function should accept one argument, the output text of command.
- **encoding** (*str*) –command output encoding

返回

shell command return code

在 0.4 版本发生变更: add `encoding` parameter and return code

```
cmd = "ls -l"
put_logbox('shell_output')
run_shell(cmd, output_func=lambda msg: logbox_append('shell_output', msg))
```

pywebio_battery.**put_logbox**(*name: str*, *height=400*, *keep_bottom=True*) → Output

Output a logbox widget

```
import time

put_logbox("log")
while True:
    logbox_append("log", f"{time.time()}\n")
    time.sleep(0.2)
```

参数

- **name** (*str*) –the name of the widget, must unique in session-wide.
- **height** (*int*) –the height of the widget in pixel

- **keep_bottom** (*bool*) –Whether to scroll to bottom when new content is appended (via *logbox_append()*).

在 0.3 版本发生变更: add `keep_bottom` parameter

`pywebio_battery.`**logbox_append**(*name: str*, *text: str*)

Append text to a logbox widget

`pywebio_battery.`**logbox_clear**(*name: str*)

Clear all contents of a logbox widget

`pywebio_battery.`**put_video**(*src: str | bytes*, *autoplay: bool = False*, *loop: bool = False*, *height: int | None = None*, *width: int | None = None*, *muted: bool = False*, *poster: str | None = None*, *scope: str | None = None*, *position: int = -1*) → Output

Output video

参数

- **src** (*str/bytes*) –Source of video. It can be a string specifying video URL, a bytes-like object specifying the binary content of the video.
- **autoplay** (*bool*) –Whether to autoplay the video. In some browsers (e.g. Chrome 70.0) autoplay doesn't work if not enable `muted`.
- **loop** (*bool*) –If True, the browser will automatically seek back to the start upon reaching the end of the video.
- **width** (*int*) –The width of the video's display area, in CSS pixels. If not specified, the intrinsic width of the video is used.
- **height** (*int*) –The height of the video's display area, in CSS pixels. If not specified, the intrinsic height of the video is used.
- **muted** (*bool*) –If set, the audio will be initially silenced.
- **poster** (*str*) –A URL for an image to be shown while the video is downloading. If this attribute isn't specified, nothing is displayed until the first frame is available, then the first frame is shown as the poster frame.
- **position** (*int scope,*) –Those arguments have the same meaning as for *put_text()*

Example:

```
url = "https://interactive-examples.mdn.mozilla.net/media/cc0-videos/flower.mp4"
put_video(url)
```

Added in version 0.4.

`pywebio_battery.`**put_audio**(*src: str | bytes*, *autoplay: bool = False*, *loop: bool = False*, *muted: bool = False*, *scope: str | None = None*, *position: int = -1*) → Output

Output audio

参数

- **src** (*str/bytes*) –Source of audio. It can be a string specifying video URL, a bytes-like object specifying the binary content of the audio.
- **autoplay** (*bool*) –Whether to autoplay the audio.

备注: Web browsers typically don't allow autoplaying audio without user interaction. If you want to autoplay audio, one way is to call `put_audio(autoplay=True)` in a call-

back function of a button. See also: https://developer.mozilla.org/en-US/docs/Web/Media/Autoplay_guide

- **loop** (*bool*) –If True, the browser will automatically seek back to the start upon reaching the end of the audio.
- **muted** (*bool*) –If set, the audio will be initially silenced.
- **scope** –The scope of the video. It can be `"session"` or `"page"`. If not specified, the video will be automatically removed when the session is closed.
- **position** (*int scope,*) –Those arguments have the same meaning as for *put_text()*

Example:

```
url = "https://interactive-examples.mdn.mozilla.net/media/cc0-audio/t-rex-roar.mp3
↪"
put_audio(url)
```

Added in version 0.4.

pywebio_battery.**wait_scroll_to_bottom**(*threshold: float = 10*, *timeout: float | None = None*) → bool

Wait until the page is scrolled to bottom.

This function is useful to achieve infinite scrolling.

参数

- **threshold** (*float*) –If the distance (in pixels) of the browser's viewport from the bottom of the page is less than the threshold, it is considered to reach the bottom
- **timeout** (*float*) –Timeout in seconds. The maximum time to wait for the page to scroll to bottom. Default is None, which means no timeout.

返回

Return `True` if the page is scrolled to bottom, return `False` only when timeout.

Example:

```
put_text('This is long text. Scroll to bottom to continue.\n' * 100)
while True:
    wait_scroll_to_bottom()
    put_text("New generated content\n"*20)
```

Added in version 0.5.

pywebio_battery.**get_all_query**()

Get URL parameter (also known as "query strings" or "URL query parameters") as a dict

pywebio_battery.**get_query**(*name: str*)

Get URL parameter value

pywebio_battery.**set_localstorage**(*key: str*, *value: str*)

Save data to user's web browser

The data is specific to the origin (protocol+domain+port) of the app. Different origins use different web browser local storage.

参数

- **key** –the key you want to create/update.

- **value** –the value you want to give the key you are creating/updating.

You can read the value by using *get_localstorage(key)*

pywebio_battery.**get_localstorage**(*key: str*) → str

Get the key’s value in user’s web browser local storage

pywebio_battery.**set_cookie**(*key: str*, *value: str*, *days=7*)

Set cookie

pywebio_battery.**get_cookie**(*key: str*)

Get cookie

pywebio_battery.**basic_auth**(*verify_func: Callable[[str, str], bool]*, *secret: str | bytes*, *expire_days=7*, *token_name='pywebio_auth_token'*) → str

Persistence authentication with username and password.

You need to provide a function to verify the current user based on username and password. The `basic_auth()` function will save the authentication state in the user’s web browser, so that the authed user does not need to log in again.

参数

- **verify_func** (*callable*) –User authentication function. It should receive two arguments: username and password. If the authentication is successful, it should return `True`, otherwise return `False`.
- **secret** (*str*) –HMAC secret for the signature. It should be a long, random str.
- **expire_days** (*int*) –how many days the auth state can keep valid. After this time, authed users need to log in again.
- **token_name** (*str*) –the name of the token to store the auth state in user browser.

Return str

username of the current authed user

Example:

```
user_name = basic_auth(lambda username, password: username == 'admin' and␣
↪password == '123',
                       secret="__TODO:_GENERATE_YOUR_OWN_RANDOM_VALUE_HERE__")
put_text("Hello, %s. You can refresh this page and see what happen" % user_name)
```

Added in version 0.4.

pywebio_battery.**custom_auth**(*login_func: Callable[[], str]*, *secret=typing.Union[str, bytes]*, *expire_days=7*, *token_name='pywebio_auth_token'*) → str

Persistence authentication with custom logic.

You need to provide a function to determine the current user and return the username. The `custom_auth()` function will save the authentication state in the user’s web browser, so that the authed user does not need to log in again.

参数

- **login_func** (*callable*) –User login function. It should receive no arguments and return the username of the current user. If fail to verify the current user, it should return `None`.
- **secret** (*str*) –HMAC secret for the signature. It should be a long, random str.
- **expire_days** (*int*) –how many days the auth state can keep valid. After this time,authed users need to log in again.

- **token_name** (*str*) –the name of the token to store the auth state in user browser.

Return str
username of the current authed user.

Added in version 0.4.

pywebio_battery.**revoke_auth**(*token_name='pywebio_auth_token'*)

Revoke the auth state of current user

参数
token_name (*str*) –the name of the token to store the auth state in user browser.

Added in version 0.4.

4.12 服务器-客户端通信协议

PyWebIO 采用服务器-客户端架构，服务端运行任务代码，通过网络与客户端（也就是用户浏览器）交互。本章介绍 PyWebIO 服务端与客户端通信的协议。

服务器与客户端有两种通信方式：WebSocket 和 Http 通信。

使用 Tornado 或 aiohttp 后端时，服务器与客户端通过 WebSocket 通信，使用 Flask 或 Django 后端时，服务器与客户端通过 Http 通信。

WebSocket 通信：

服务器与客户端通过 WebSocket 连接发送 json 序列化之后的 PyWebIO 消息

Http 通信：

- The client polls the backend through Http GET requests, and the backend returns a list of PyWebIO messages serialized in json
- When the user submits the form or clicks the button, the client submits data to the backend through Http POST request

为方便区分，下文将由服务器向客户端发送的数据称作 command，将客户端发向服务器的数据称作 event

以下介绍 command 和 event 的格式

4.12.1 Command

command 由服务器-> 客户端，基本格式为:

```
{
    "command": ""
    "task_id": ""
    "spec": {}
}
```

各字段含义如下:

- `command` 字段表示指令名
- `task_id` 字段表示发送指令的 Task id，客户端对于此命令的响应事件都会传递 task_id
- `spec` 字段为指令的参数，不同指令参数不同

需要注意，以下不同命令的参数和 PyWebIO 的对应函数的参数大部分含义一致，但是也有些许不同。
以下分别对不同指令的 `spec` 字段进行说明：

input_group

显示一个输入表单

表 2: `spec` 可用字段

字段	是否必选	类型	字段说明
label	False	str	表单标题
inputs	True	list	输入项
cancelable	False	bool	表单是否可以取消 若 `cancelable=True` 则会在表单底部显示一个”取消”按钮， 用户点击取消按钮后，触发 `from_cancel` 事件

`inputs` 字段为输入项组成的列表，每一输入项为一个 `dict`，字段如下：

- label: 输入标签名。必选
- type: 输入类型。必选
- name: 输入项 id。必选
- onchange: bool, whether to push input value when input change
- onbulr: bool, whether to push input value when input field `onblur`
- auto_focus: 自动获取输入焦点. 输入项列表中最多只能由一项的 auto_focus 为真
- help_text: 帮助文字
- 输入项 HTML 元素额外的 HTML 属性
- Other attributes of different input types

输入类型目前有：

- text: 文本输入
- number: 数字输入
- password: 密码输入
- checkbox: 多选项
- radio: 单选项
- select: 下拉选择框 (可单选/多选)
- textarea: 大段文本输入
- file: 文件上传
- actions: Actions selection.

输入类型与 html 输入元素的对应关系:

- text: input[type=text]
- number: input[type=number]
- float: input[type=text], and transform input value to float
- password: input[type=password]
- checkbox: input[type=checkbox]
- radio: input[type=radio]
- select: select https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/select
- textarea: textarea https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/textarea
- file: input[type=file]
- actions: button[type=submit] https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/button

Unique attributes of different input types:

- text,number,password:
  - action: Display a button on the right of the input field. The format of `action` is `{label: button label, callback_id: button click callback id}`
- textarea:
  - code: Codemirror 参数或 bool, 同*`pywebio.input.textarea()`* 的 `code` 参数
- select:
  - options: `{label:, value: , [selected:,] [disabled:]}`
- checkbox:
  - options: `{label:, value: , [selected:,] [disabled:]}`
  - inline
- radio:
  - options: `{label:, value: , [selected:,] [disabled:]}`
  - inline
- actions
  - buttons: `{label:, value:, [type: 'submit'/'reset'/'cancel'], [disabled:], [color:]}`.
- file:
  - 是否允许多文件上传
  - 单个文件的最大大小（字节），超过限制将会禁止上传
  - 所有文件的最大大小（字节），超过限制将会禁止上传
- slider
  - min_value: The minimum permitted value.
  - max_value: The maximum permitted value.
  - step: The stepping interval.
  - float: If need return a float value

update_input

更新输入项，用于对当前显示表单中输入项的 `spec` 进行更新

命令 `spec` 可用字段：

- target_name: str The name of the target input item.
- target_value: str, optional. Used to filter item in checkbox, radio
- attributes: dist, fields need to be updated
  - valid_status: When it is bool, it means setting the state of the input value, pass/fail; when it is 0, it means clear the valid_status flag
  - value: Set the value of the item
  - label
  - placeholder
  - invalid_feedback
  - valid_feedback
  - help_text
  - options: only available in checkbox, radio and select type
  - other fields of item’s `spec` // not support the `inline` field

close_session

指示服务器端已经关闭连接。`spec` 为空

set_session_id

将当前会话 id 发送至客户端，客户端可以使用此 id 来重连会话 (仅在 websocket 连接中可用)。`spec` 字段为会话 id

destroy_form

Destroy the current form. `spec` of the command is empty.

表单在页面上提交之后不会自动销毁，需要使用此命令显式销毁

output

Output content

The `spec` fields of `output` commands:

- type: content type
- style: str, Additional css style
- container_selector: The css selector of output widget’s content slot. If empty(default), use widget self as container
- container_dom_id: The dom id need to be set to output widget’s content slot.

- scope: str, 内容输出的域的 css 选择器。若 CSS 选择器匹配到页面上的多个容器，则内容会输出到每个匹配到的容器
- int, 在输出域中输出的位置, 见*输出函数的 scope 相关参数*
- click_callback_id:
- 不同 type 时的特有字段

`container_selector` and `container_dom_id` is used to implement output context manager.

`type` 的可选值及特有字段：

- type: markdown
  - content: str
  - options: dict, marked.js options
  - sanitize: bool, 是否使用 DOMPurify 对内容进行过滤来防止 XSS 攻击。
- type: html
  - content: str
  - sanitize: bool, 是否使用 DOMPurify 对内容进行过滤来防止 XSS 攻击。
- type: text
  - content: str
  - inline: bool, Use text as an inline element (no line break at the end of the text)
- type: buttons
  - callback_id:
  - buttons:[{value:, label:, [color:], [disabled:]},…]
  - small: bool，是否显示为小按钮样式
  - group: bool, Whether to group the buttons together
  - link: bool，是否显示为链接样式
  - outline: bool, Whether enable outline style.
- type: file
  - name: 下载保存为的文件名
  - content: 文件 base64 编码的内容
- type: table
  - data: Table data, which is a two-dimensional list, the first row is table header.
  - span: cell span info. Format: { “[row id],[col id]” : { “row” :row span, “col” :col span }}
- type: pin
  - input: input spec, same as the item of `input_group.inputs`
- type: scope
  - dom_id: the DOM id need to be set to this widget
  - contents list: list of output spec
- type: scrollable

  - contents:
  - min_height:
  - max_height:
  - keep_bottom:
  - border:
- type: tabs
  - tabs:
- type: custom_widget
  - template:
  - data:

pin_value

命令 `spec` 可用字段：

- name

pin_update

命令 `spec` 可用字段：

- name
- attributes: dist, fields need to be updated

pin_wait

命令 `spec` 可用字段：

- names: list,
- timeout: int,

pin_onchange

set a callback which is invoked when the value of pin widget is changed

The `spec` fields of `pin_onchange` commands:

- name: string
- callback_id: string, if `None`, not set callback
- clear: bool

popup

Show popup

The `spec` fields of `popup` commands:

- title
- content
- size: `large`, `normal`, `small`
- implicit_close
- closable
- dom_id: DOM id of popup container element

toast

Show a notification message

The `spec` fields of `popup` commands:

- content
- duration
- position: `'left'` / `'center'` / `'right'`
- color: hexadecimal color value starting with '#'
- callback_id

close_popup

Close the current popup window.

`spec` of the command is empty.

set_env

Config the environment of current session.

The `spec` fields of `set_env` commands:

- title (str)
- output_animation (bool)
- auto_scroll_bottom (bool)
- http_pull_interval (int)
- input_panel_fixed (bool)
- input_panel_min_height (int)
- input_panel_init_height (int)
- input_auto_focus (bool)

output_ctl

Output control

The `spec` fields of `output_ctl` commands:

- set_scope: scope name
  - container: 新创建的 scope 的父 scope 的 css 选择器
  - position: int, 在父 scope 中创建此 scope 的位置.
  - scope 已经存在时如何操作:
    - null/不指定: 表示立即返回不进行任何操作
    - `'remove'`：先移除旧 scope 再创建新 scope
    - `'clear'`：将旧 scope 的内容清除，不创建新 scope
    - `'blank'`: Clear the contents of the old scope and keep the height, don't create a new scope
- loose: css selector of the scope, set the scope not to keep the height (i.e., revoke the effect of `set_scope(if_exist='blank')`)
- clear: 需要清空的 scope 的 css 选择器
- clear_before
- clear_after
- clear_range:[,]
- **scroll_to**
  - position: top/middle/bottom 与 scroll_to 一起出现, 表示滚动页面，让 scope 位于屏幕可视区域顶部/中部/底部
- remove: 将给定的 scope 连同 scope 处的内容移除

run_script

run javascript code in user's browser

The `spec` fields of `run_script` commands:

- code: 字符串格式的要运行的 js 代码
- args: 传递给代码的局部变量。字典类型，字典键表示变量名，字典值表示变量值 (变量值需要可以被 json 序列化)
- eval: bool, whether to submit the return value of javascript code

download

Send file to user

The `spec` fields of `download` commands:

- name: str, File name when downloading
- content: str, File content in base64 encoding.

4.12.2 Event

Event 消息由客户端发往服务端。基本格式:

```
{
    event: event name
    task_id: ""
    data: object/str
}
```

`event` 表示事件名称。`data` 为事件所携带的数据，其根据事件不同内容也会不同，不同事件对应的 `data` 字段如下:

input_event

表单发生更改时触发

- event_name: 目前可用值 ' `blur`'，表示输入项失去焦点
- name: 输入项 name
- value: 输入项值

注意：checkbox radio 不产生 blur 事件

callback

用户点击显示区的按钮时触发

在 `callback` 事件中，`task_id` 为对应的 `button` 组件的 `callback_id` 字段；事件的 `data` 为被点击 button 的 `value`

from_submit

用户提交表单时触发

事件 `data` 字段为表单 `name` -> 表单值的字典

from_cancel

表单取消输入

The `data` of the event is `None`

js_yield

submit data from js. It’s a common event to submit data to backend.

事件 `data` 字段为相应的数据

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

CHAPTER 6

Discussion and support

- Need help when use PyWebIO? Make a new discussion on Github Discussions.
- Report bugs on the GitHub issue.

Python 模块索引

p

索引

P

R

S

T

U

W