3.4 Асинхронные приложения

Асинхронные приложения

Приложение Flet может быть написано как асинхронное и использовать и

asyncio

другие асинхронные библиотеки Python. Вызов корутин естественным образом поддерживается в Flet, поэтому вам не нужно оборачивать их для синхронного выполнения.

По умолчанию Flet использует библиотеку для выполнения

threading

пользовательских сессий и обработки событий в отдельных потоках, но иногда это может быть неэффективным использованием CPU, так как он ничего не делает, ожидая ответа HTTP или выполняя sleep() .

Asyncio, с другой стороны, позволяет реализовывать параллельность в одном потоке, переключая контекст выполнения между "корутинами". Это особенно важно для приложений, которые будут публиковаться в виде статических веб-сайтов с использованием Pyodide. Pyodide - это среда выполнения Python, созданная как WebAssembly (WASM) и работающая в браузере. На момент написания этого текста она еще не поддерживает многопоточность.

# С чего начать?

Чтобы начать работу с асинхронным приложением Flet, вы должны сделать

основной метод асинхронным:

main()

import flet as ft

async def main(page: ft.Page):

await page.add\_async(ft.Text("Привет, асинхронный мир!"))

ft.app(main)

Вы можете использовать await ft.app\_async(main) , если приложение Flet является частью большего приложения и вызывается из асинхронного кода.

Обратите внимание на использование для

await page.add\_async(...)

добавления новых элементов управления на страницу. В асинхронном приложении

вы больше не можете использовать или другие синхронные методы

page.add()

страницы - вы должны использовать их асинхронные аналоги, заканчивающиеся на \_async , везде в коде:



page.update() → await page.update\_async()

page.add() → await page.add\_async()





page.clean() → await page.clean\_async()

* и т.д.

# Обработчики событий элементов управления

Обработчики событий элементов управления могут быть как синхронными, так и асинхронными.

Если обработчик не вызывает никаких асинхронных методов, он может быть обычным синхронным методом:

def page\_resize(e):

print("Новый размер страницы:", page.window\_width,

page.window\_height)

page.on\_resize = page\_resize

Однако, если обработчик вызывает асинхронную логику, он также должен быть асинхронным:

async def button\_click(e):

page.controls.append(ft.Text("Привет!"))

await page.update\_async()

ft.ElevatedButton("Скажи привет!", on\_click=button\_click)

**Асинхронные лямбды**

В Python нет асинхронных лямбд. Это вполне нормально иметь обработчик событий в виде лямбда-функции в асинхронном приложении для простых вещей:

page.on\_error = lambda e: print("Ошибка страницы:", e.data)

но у вас не может быть асинхронной лямбды, поэтому должен использоваться асинхронный обработчик событий.

# Задержка выполнения

Чтобы задержать выполнение кода в асинхронном приложении Flet, вы должны

использовать вместо time.sleep() . Например:

asyncio.sleep()

import asyncio

import flet as ft

async def main(page: ft.Page):

async def button\_click(e):

await asyncio.sleep(1)

await page.add\_async(ft.Text("Привет!"))

await page.add\_async(

ft.ElevatedButton("Скажи привет с задержкой!",

on\_click=button\_click)

)

ft.app(main)

# Многопоточность

Технически, никто не запретит вам использовать библиотеку

threading

асинхронном приложении, но это будет плохой идеей. Версии

asyncio

блокировок, очередей и задач, используемые в Flet API, не являются потокобезопасными и, например, вызов

await page.update\_async()

в для

из

нескольких потоков приведет к непредсказуемым результатам. Кроме того,

библиотека не поддерживается Pyodide, если вы решите развернуть

threading

ваше приложение как статический веб-сайт.

Чтобы запустить что-то в фоновом режиме, используйте asyncio.create\_task() . Например, асинхронная версия элемента управления "countdown" из руководства по пользовательским элементам управления будет выглядеть так:

import asyncio

import flet as ft

class Countdown(ft.UserControl):

def init (self, seconds):

super(). init ()

self.seconds = seconds

async def did\_mount\_async(self):

self.running = True

asyncio.create\_task(self.update\_timer())

async def will\_unmount\_async(self):

self.running = False

async def update\_timer(self):

while self.seconds and self.running:

mins, secs = divmod(self.seconds, 60)

self.countdown.value = "{:02d}:{:02d}".format(mins, secs)

await self.update\_async()

await asyncio.sleep(1)

self.seconds -= 1

def build(self):

self.countdown = ft.Text()

return self.countdown

async def main(page: ft.Page):

await page.add\_async(Countdown(120), Countdown(60))

ft.app(target=main)

3.4 Асинхронные приложения

Какой из методов следует использовать для добавления новых элементов управления на страницу в асинхронном приложении Flet?