



ХОЧУ ПОМОЧЬ  
ПРОЕКТУ

 mango-office.ru

РЕКЛАМА

⋮



### Виртуальная АТС Расширенная


До 120 функций. Когда нужно распределять звонки по группам сотрудников и по типам...

Узнать больше

Статьи

Углубленный обзор и доступ к состоянию локального контекста

Модуль contextvars в Python 3.7. и предоставляет API для управления, хранения и доступа к состоянию локального контекста



Реклама. ПАО «Группа Ренессанс Страхование». Подробности и лицензии на [repln.ru](#). 18+

Код пр

[Модуль contextvars](#) не имеет строгого отношения к [контекстным менеджерам Python](#). Хотя он предоставляет механизм, который может быть использован контекстными менеджерами для хранения своего состояния.

Локальные переменные, используемые в потоках недостаточны для асинхронных задач, т.к. асинхронный код выполняется в одном потоке. Любой контекстный менеджер, который сохраняет и восстанавливает значение контекста с помощью передачи [threading.local\(\)](#) будет иметь свои контекстные значения, которые неожиданно будут передаваться в другой код при использовании [синтаксиса async/await](#).

Чтобы предотвратить такое поведение в асинхронном коде, диспетчеры контекста, у которых есть состояние, должны использовать переменные контекста [модуля contextvars](#) вместо [threading.local\(\)](#).

Несколько примеров, когда желательно иметь рабочее локальное хранилище контекста для [асинхронного кода](#):

- Менеджеры контекста, такие как [контексты модуля decimal](#);
- Данные, связанные с запросами, такие как токены безопасности и данные запроса в веб-приложениях, языковой контекст для gettext и т. д.;
- Профилирование, трассировка и [ведение журнала](#) в больших базах кода.

Предлагаемый механизм доступа к контекстным переменным, использует [класс contextvars.ContextVar\(\)](#). Модуль (например [decimal](#)), желающий использовать новый механизм, должен:

- объявить глобальную переменную модуля, содержащую ContextVar в качестве ключа;
- получить доступ к текущему значению ключевой переменной с помощью метода ContextVar.get();
- изменить текущее значение ключевой переменной с помощью метода ContextVar.set().

Понятие "текущее значение" заслуживает особого рассмотрения: разные асинхронные задачи, которые существуют и выполняются одновременно, могут иметь разные значения для одного и того же ключа. Эта идея хорошо известна из локального хранилища потока, но в этом случае местоположение значения не обязательно привязано к потоку. Вместо этого существует понятие "текущий контекст", который хранится в локальном хранилище потока. За управление текущим контекстом отвечает структура задачи, например: [asyncio](#).

Контекст - это словарь, который отображает объекты [ContextVar](#) на их значения. Сам контекст предоставляет интерфейс abc.Mapping (неизменяемый словарь), поэтому его нельзя изменить напрямую. Чтобы установить новое значение для переменной контекста в объекте контекста, пользователю необходимо:

- сделать объект [Context](#) "текущим" с помощью метода Context.run();
- использовать ContextVar.set(), чтобы установить новое значение для переменной контекста.

Метод ContextVar.get() ищет переменную в текущем объекте Context, используя self в качестве ключа.

Вверх

Получить прямую ссылку на текущий объект `contextvars.Context` невозможно, но можно получить его частичную копию с помощью [функции `contextvars.copy\_context\(\)`](#). Это гарантирует, что вызывающий `Context.run()` является единственным владельцем его объекта `Context`.

Примеры использования модуля contextvars:

```
import contextvars

var = contextvars.ContextVar('var')
var.set('spam')

def main():
    # 'var' был установлен в 'spam' перед вызовами
    # 'copy_context()' и 'ctx.run(main)', так:
    # var.get() == ctx[var] == 'spam'

    var.set('ham')

    # Теперь, после установки 'var' в 'ham':
    # var.get() == ctx[var] == 'ham'

ctx = contextvars.copy_context()

# Любые изменения, которые функция 'main()'
# вносит в 'var', будут содержаться в 'ctx'.
ctx.run(main)

# Функция 'main()' была запущена в контексте 'ctx',
# следовательно 'main()' исполниться с измененной 'var':
# ctx[var] == 'ham'

# Однако, за пределами контекста 'ctx', переменная
# 'var' по-прежнему имеет значение 'spam':
# var.get() == 'spam'
```

## Поддержка асинхронного кода.

Переменные контекста [модуля contextvars](#) изначально поддерживаются в [asyncio](#) и готовы к использованию без какой-либо дополнительной настройки.

Пример простого echo-сервера, который использует переменную контекста, делающую адрес удаленного клиента доступным в Task, которая обрабатывает этого клиента.

Для тестирования кода, представленного ниже используйте соединение telnet на порту 8081: `telnet 127.0.0.1 8081`

```
import asyncio, contextvars

client_addr_var = contextvars.ContextVar('client_addr')

def render_goodbye():
    # Можно получить доступ к адресу текущего обработанного
    # клиента, не передавая его явно этой функции.

    client_addr = client_addr_var.get()
    return f'Good bye, client @ {client_addr}\n'.encode()

async def handle_request(reader, writer):
    addr = writer.transport.get_extra_info('socket').getpeername()
    client_addr_var.set(addr)

    # В любом коде, который вызывается, теперь можно
    # получить адрес клиента, вызвав 'client_addr_var.get()'.

    while True:
        line = await reader.readline()
        print(line)
        if not line.strip():
```

13.09.2023, 21:58

Модуль contextvars, локальный контекст в Python

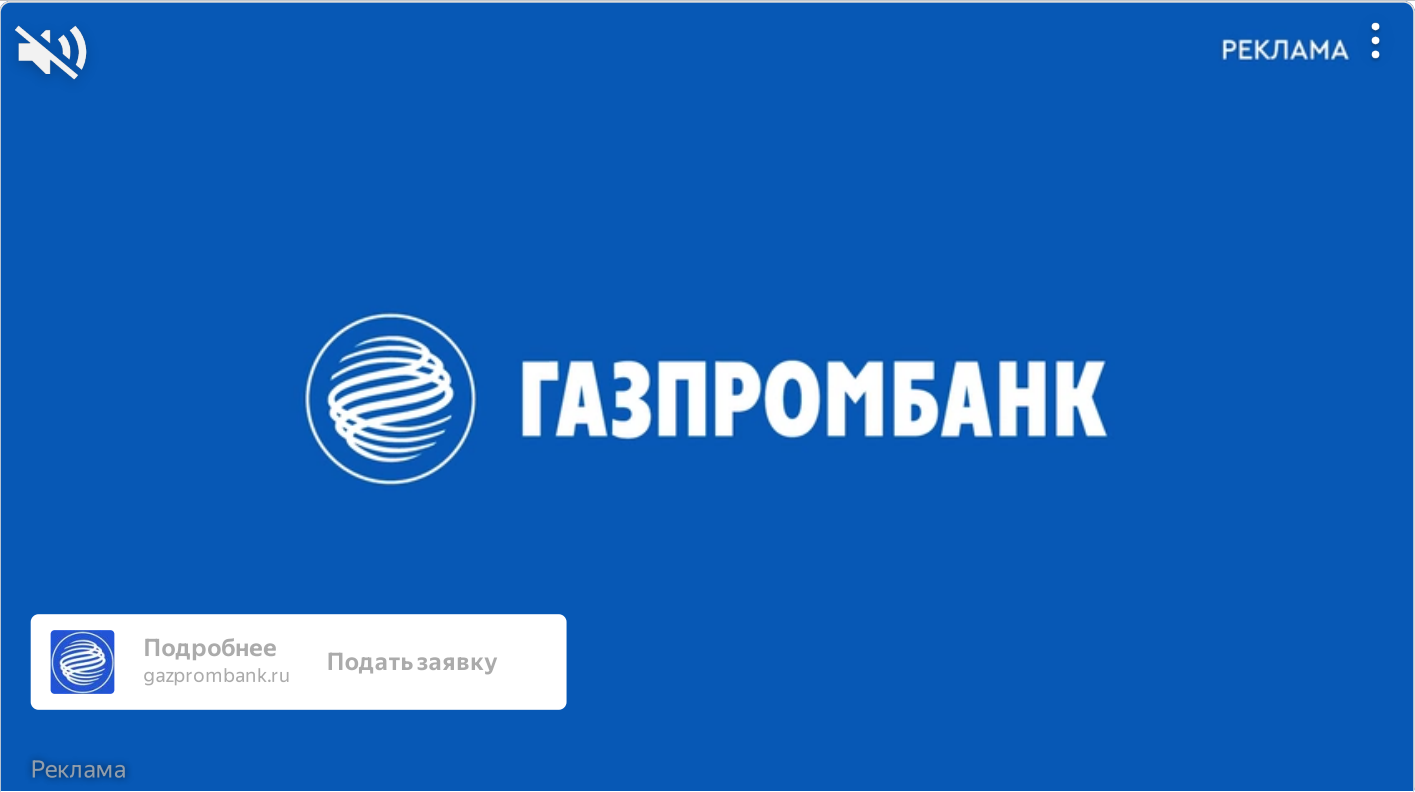
```
        break
    writer.write(line)

    writer.write(render_goodbye())
    writer.close()
    :

async def main():
    srv = await asyncio.start_server(
        handle_request, '127.0.0.1', 8081)

    async with srv:
        await srv.serve_forever()

asyncio.run(main())
```



Содержание раздела:

- [КРАТКИЙ ОБЗОР МАТЕРИАЛА.](#)
- [Класс ContextVar\(\) модуля contextvars](#)
- [Объект Token\(\) модуля contextvars](#)
- [Объект Context модуля contextvars](#)
- [Функция copy\\_context\(\) модуля contextvars](#)