PyCon China 2024

For Good . For fun.

2024/11/23 中国 上海

PYCHINA
Python 中国社区

PyCon China 2024

>> 2024/11/23 上海

>> For good . For fun.

同时编写同步 IO 和异步 IO 的代码

—— Aber

Asynchronous

```python
def worker():
    ...
    time.sleep(1)
    ...
```

```python
async def worker():
    ...
    await asyncio.sleep(1)
    ...
```

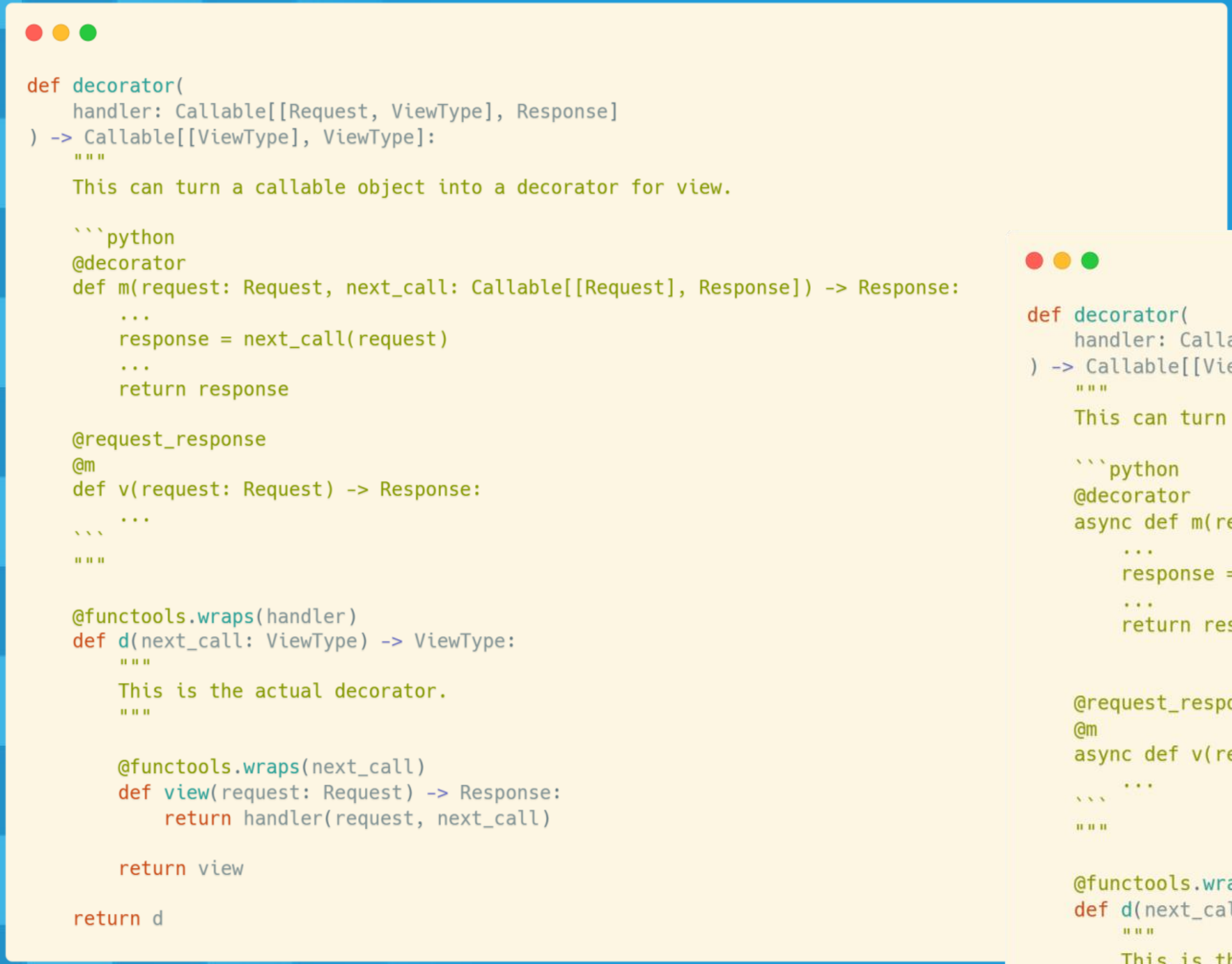
```python
def socket_connect():
    ...
    sock = socket.connect(addr)
    ...
```

```python
async def socket_connect():
    ...
    reader, writer = await asyncio.open_connection(addr)
    ...
```

```python
def read_file() -> bytes:
    with open(...) as fp:
        return fp.read()
```

```python
async def read_file() -> bytes:
    async with async_open(...) as fp:
        return fp.read()
```

大巧不工

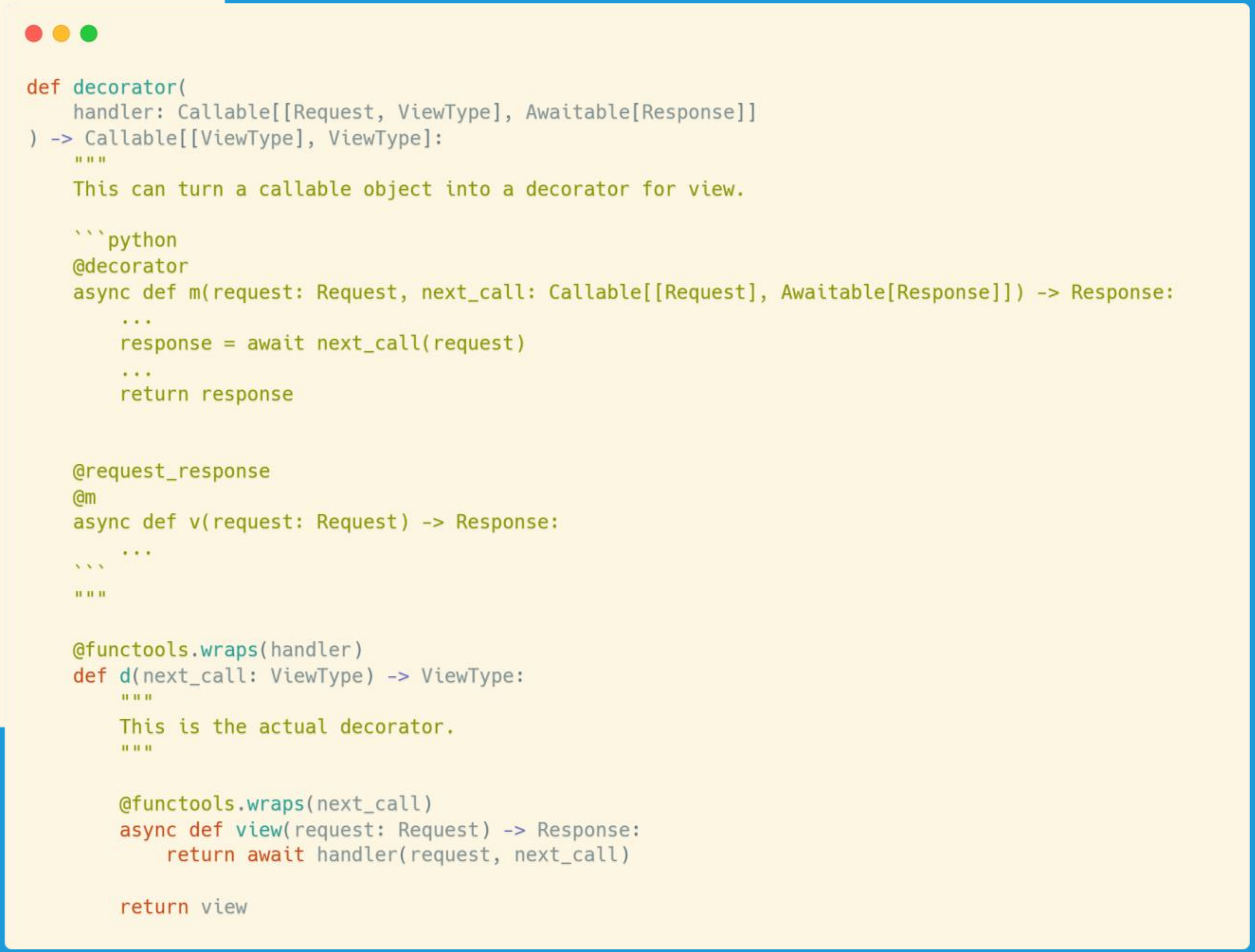


~~~python
def decorator(
    handler: Callable[[Request, ViewType], Response]
) -> Callable[[ViewType], ViewType]:
    """
    This can turn a callable object into a decorator for view.

    ```python
    @decorator
    def m(request: Request, next_call: Callable[[Request], Response]) -> Response:
        ...
        response = next_call(request)
        ...
        return response

    @request_response
    @m
    def v(request: Request) -> Response:
        ...
    ~~~
    """

    @functools.wraps(handler)
    def d(next_call: ViewType) -> ViewType:
        """
        This is the actual decorator.
        """

        @functools.wraps(next_call)
        def view(request: Request) -> Response:
            return handler(request, next_call)

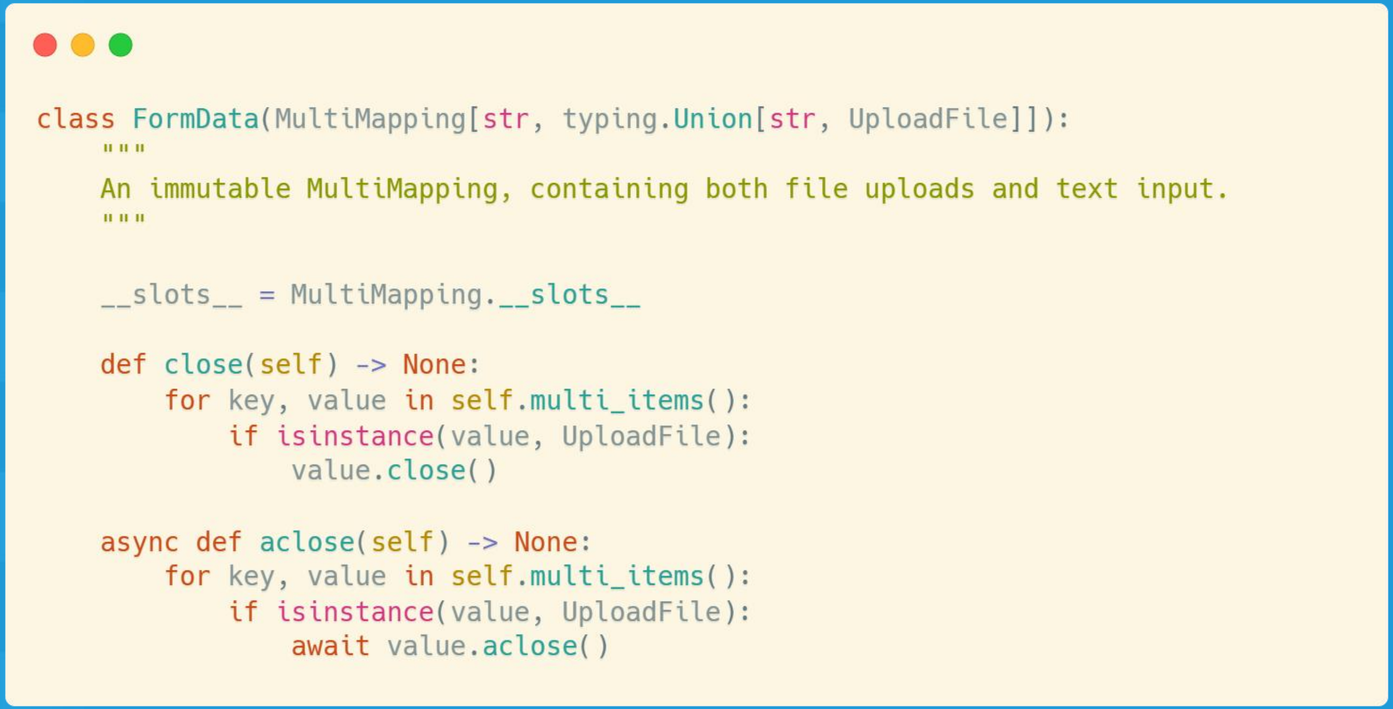
        return view

    return d
~~~



```python
def decorator(
    handler: Callable[[Request, ViewType], Awaitable[Response]]
) -> Callable[[ViewType], ViewType]:
    """
    This can turn a callable object into a decorator for view.

    ```python
    @decorator
    async def m(request: Request, next_call: Callable[[Request], Awaitable[Response]]) -> Response:
        ...
        response = await next_call(request)
        ...
        return response


    @request_response
    @m
    async def v(request: Request) -> Response:
        ...
    ~~~
    """

    @functools.wraps(handler)
    def d(next_call: ViewType) -> ViewType:
        """
        This is the actual decorator.
        """

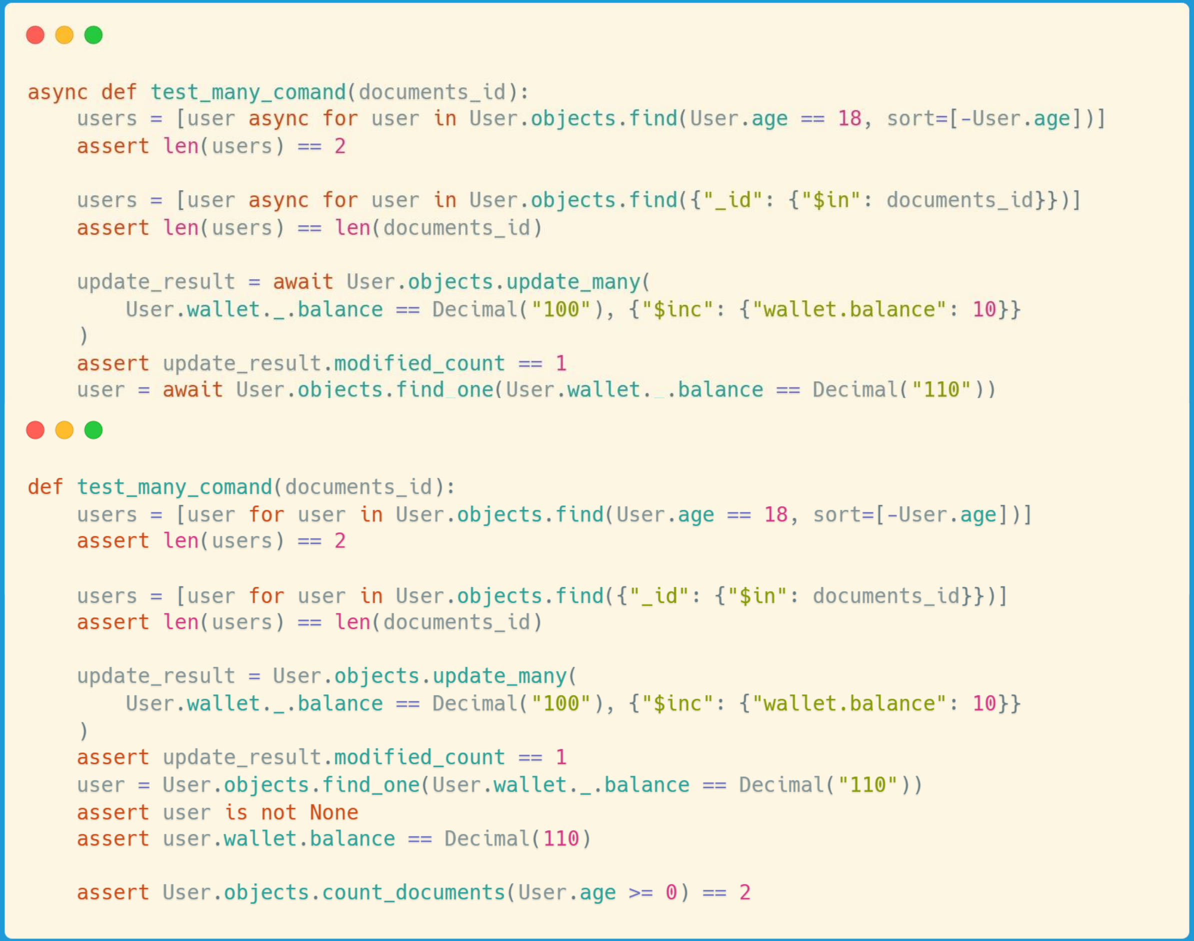
        @functools.wraps(next_call)
        async def view(request: Request) -> Response:
            return await handler(request, next_call)

        return view
```
```



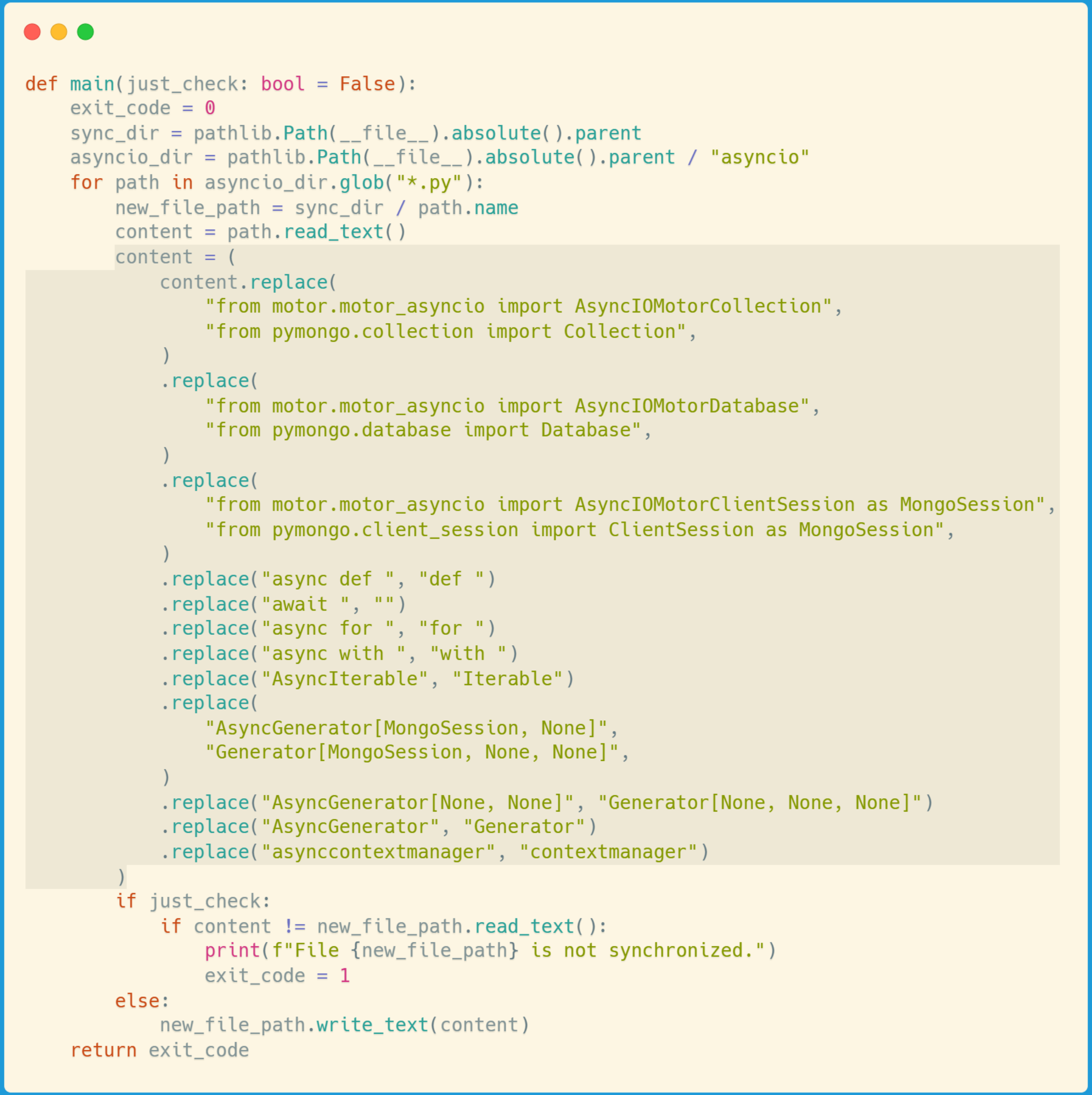
```python
class FormData(MultiMapping[str, typing.Union[str, UploadFile]]):
    """
    An immutable MultiMapping, containing both file uploads and text input.
    """

    __slots__ = MultiMapping.__slots__

    def close(self) -> None:
        for key, value in self.multi_items():
            if isinstance(value, UploadFile):
                value.close()

    async def aclose(self) -> None:
        for key, value in self.multi_items():
            if isinstance(value, UploadFile):
                await value.aclose()
```

手动的全自动生成



```python
async def test_many_comand(documents_id):
    users = [user async for user in User.objects.find(User.age == 18, sort=[-User.age])]
    assert len(users) == 2

    users = [user async for user in User.objects.find({"_id": {"$in": documents_id}})]
    assert len(users) == len(documents_id)

    update_result = await User.objects.update_many(
        User.wallet._.balance == Decimal("100"), {"$inc": {"wallet.balance": 10}}
    )
    assert update_result.modified_count == 1
    user = await User.objects.find_one(User.wallet._.balance == Decimal("110"))
```

```python
def test_many_comand(documents_id):
    users = [user for user in User.objects.find(User.age == 18, sort=[-User.age])]
    assert len(users) == 2

    users = [user for user in User.objects.find({"_id": {"$in": documents_id}})]
    assert len(users) == len(documents_id)

    update_result = User.objects.update_many(
        User.wallet._.balance == Decimal("100"), {"$inc": {"wallet.balance": 10}}
    )
    assert update_result.modified_count == 1
    user = User.objects.find_one(User.wallet._.balance == Decimal("110"))
    assert user is not None
    assert user.wallet.balance == Decimal(110)

    assert User.objects.count_documents(User.age >= 0) == 2
```

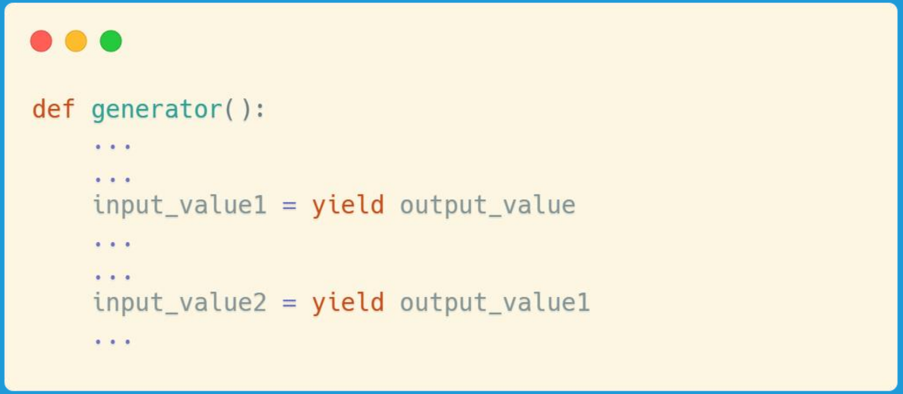


```python
def main(just_check: bool = False):
    exit_code = 0
    sync_dir = pathlib.Path(__file__).absolute().parent
    asyncio_dir = pathlib.Path(__file__).absolute().parent / "asyncio"
    for path in asyncio_dir.glob("*.py"):
        new_file_path = sync_dir / path.name
        content = path.read_text()
        content = (
            content.replace(
                "from motor.motor_asyncio import AsyncIOMotorCollection",
                "from pymongo.collection import Collection",
            )
            .replace(
                "from motor.motor_asyncio import AsyncIOMotorDatabase",
                "from pymongo.database import Database",
            )
            .replace(
                "from motor.motor_asyncio import AsyncIOMotorClientSession as MongoSession",
                "from pymongo.client_session import ClientSession as MongoSession",
            )
            .replace("async def ", "def ")
            .replace("await ", "")
            .replace("async for ", "for ")
            .replace("async with ", "with ")
            .replace("AsyncIterable", "Iterable")
            .replace(
                "AsyncGenerator[MongoSession, None]",
                "Generator[MongoSession, None, None]",
            )
            .replace("AsyncGenerator[None, None]", "Generator[None, None, None]")
            .replace("AsyncGenerator", "Generator")
            .replace("asynccontextmanager", "contextmanager")
        )
        if just_check:
            if content != new_file_path.read_text():
                print(f"File {new_file_path} is not synchronized.")
                exit_code = 1
        else:
            new_file_path.write_text(content)
    return exit_code
```

元编程

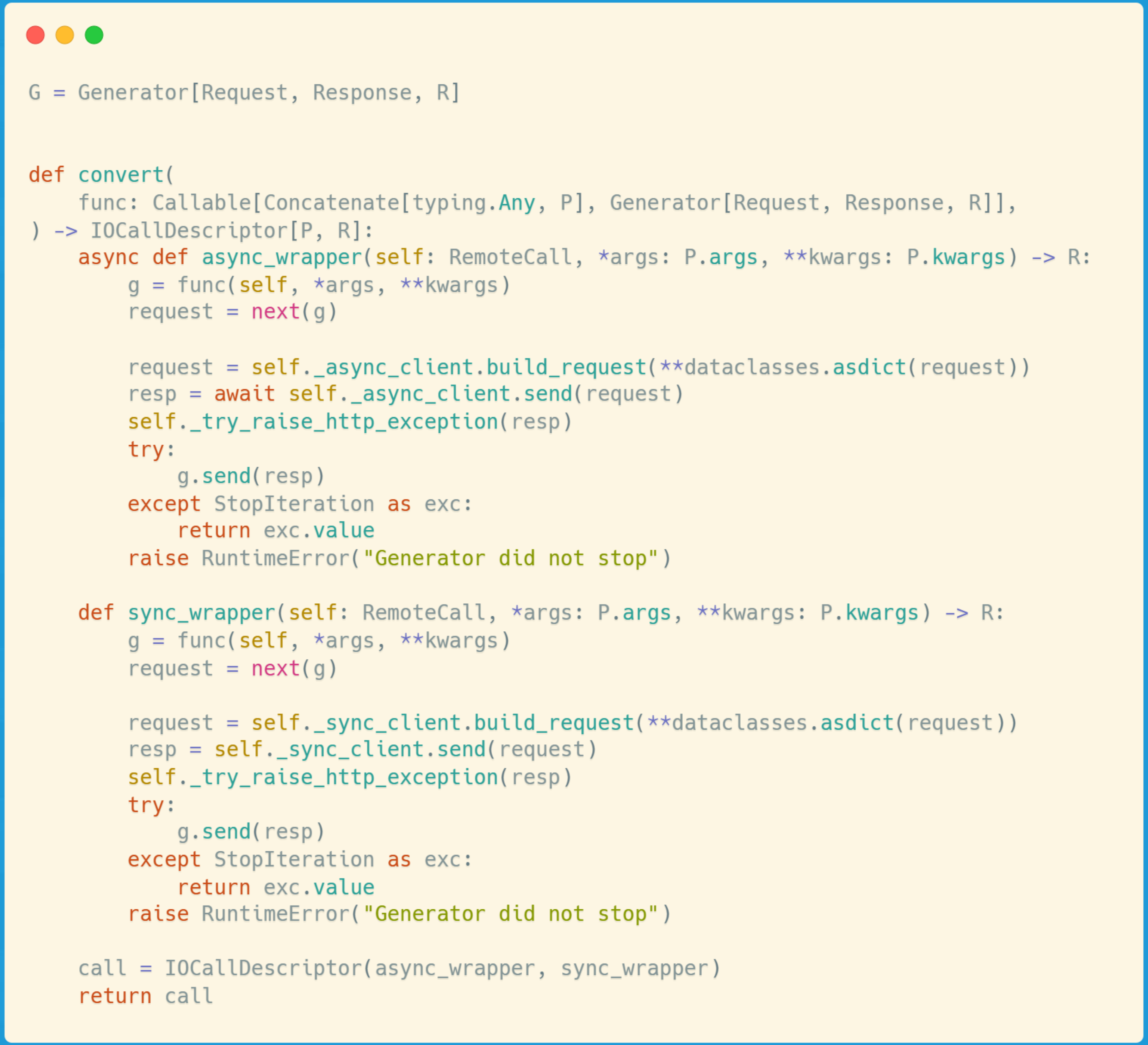
Generator pipe

```
g = generator()

output_value = next(g)

output_value1 = g.send(input_value1)

...
```

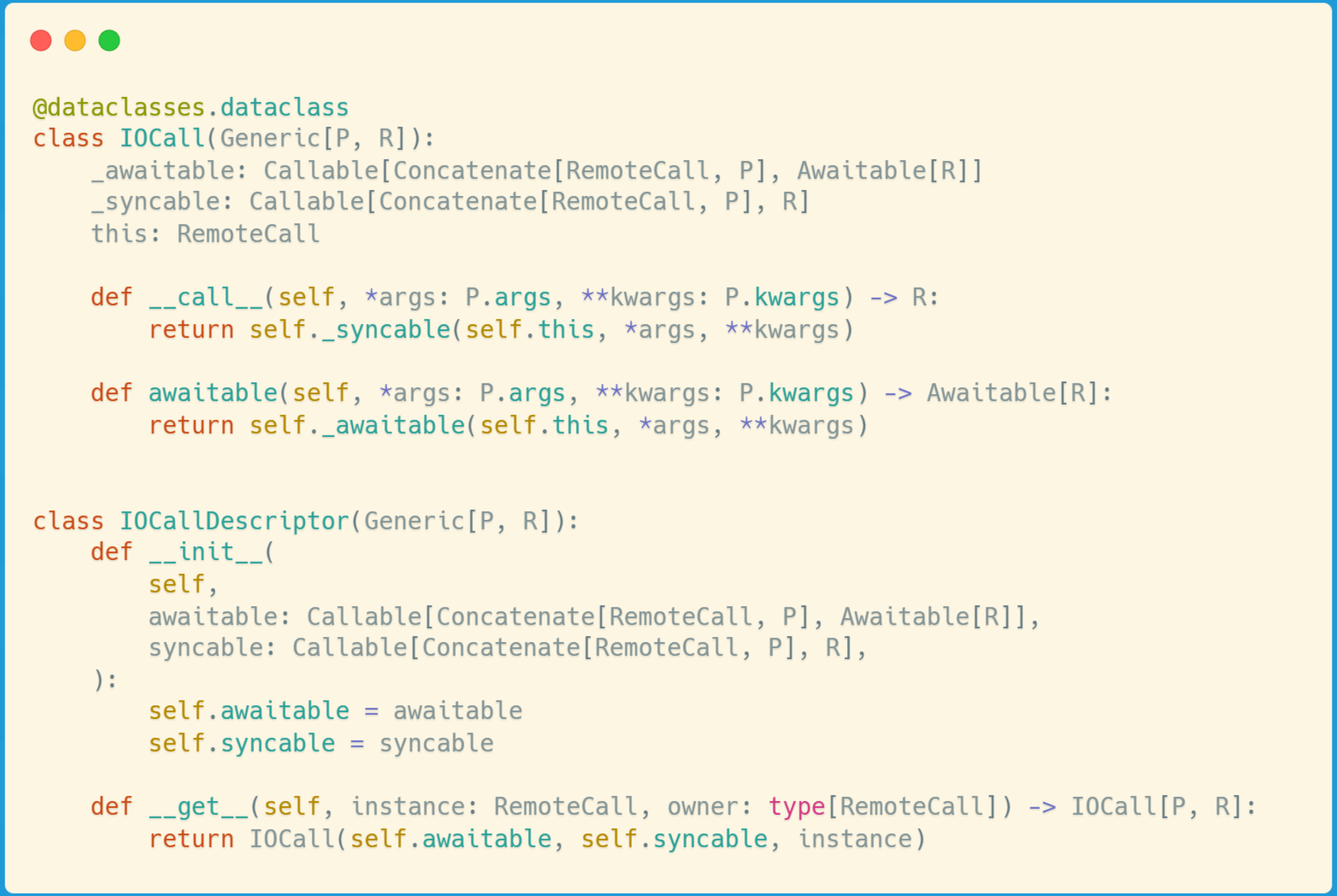


```
def generator():
    ...
    ...
    input_value1 = yield output_value
    ...
    ...
    input_value2 = yield output_value1
    ...
```

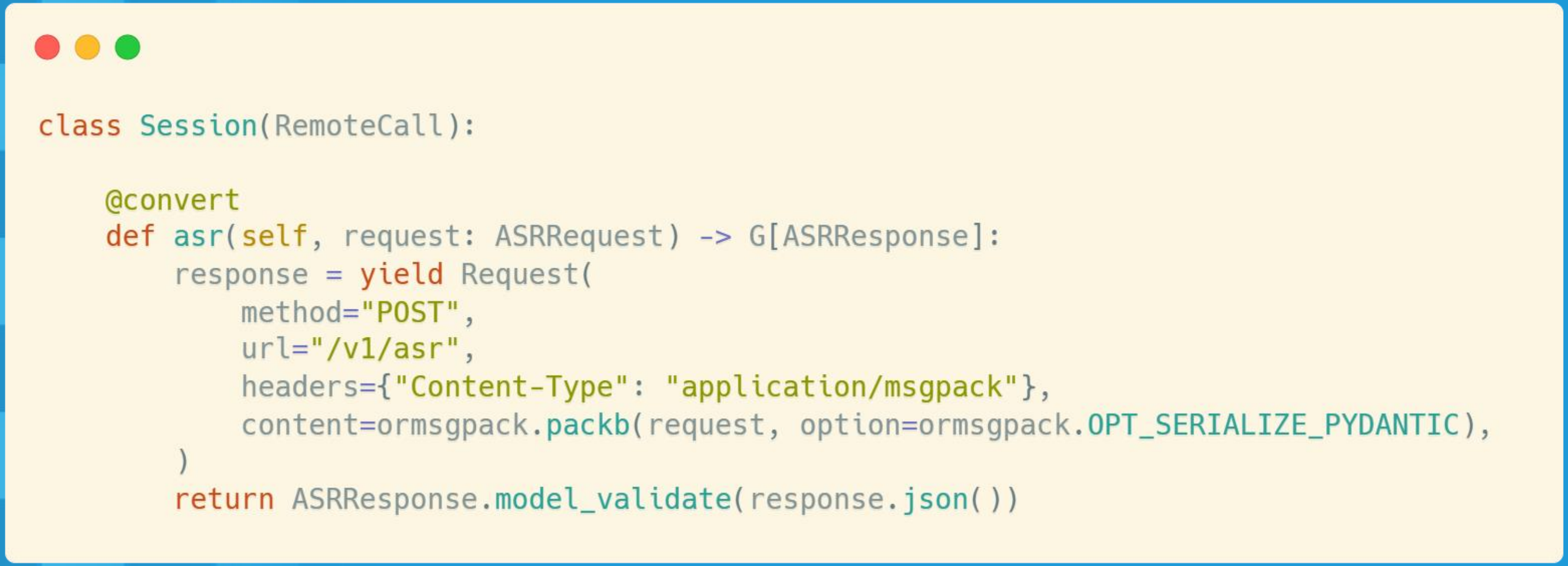
```python
class Session(RemoteCall):

    def asr(self, request: ASRRequest) -> ASRResponse:
        response = self.send(Request(
            method="POST",
            url="/v1/asr",
            headers={"Content-Type": "application/msgpack"},
            content=ormsgpack.packb(request, option=ormsgpack.OPT_SERIALIZE_PYDANTIC),
        ))
        return ASRResponse.model_validate(response.json())
        return ASRResponse.model_validate(response.json())
```



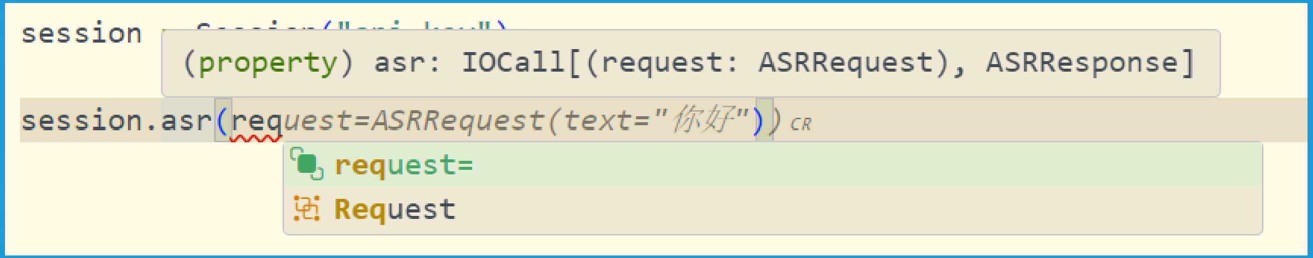
```python
G = Generator[Request, Response, R]


def convert(
    func: Callable[Concatenate[typing.Any, P], Generator[Request, Response, R]],
) -> IOCallDescriptor[P, R]:
    async def async_wrapper(self: RemoteCall, *args: P.args, **kwargs: P.kwargs) -> R:
        g = func(self, *args, **kwargs)
        request = next(g)

        request = self._async_client.build_request(**dataclasses.asdict(request))
        resp = await self._async_client.send(request)
        self._try_raise_http_exception(resp)
        try:
            g.send(resp)
        except StopIteration as exc:
            return exc.value
        raise RuntimeError("Generator did not stop")

    def sync_wrapper(self: RemoteCall, *args: P.args, **kwargs: P.kwargs) -> R:
        g = func(self, *args, **kwargs)
        request = next(g)

        request = self._sync_client.build_request(**dataclasses.asdict(request))
        resp = self._sync_client.send(request)
        self._try_raise_http_exception(resp)
        try:
            g.send(resp)
        except StopIteration as exc:
            return exc.value
        raise RuntimeError("Generator did not stop")

    call = IOCallDescriptor(async_wrapper, sync_wrapper)
    return call
```



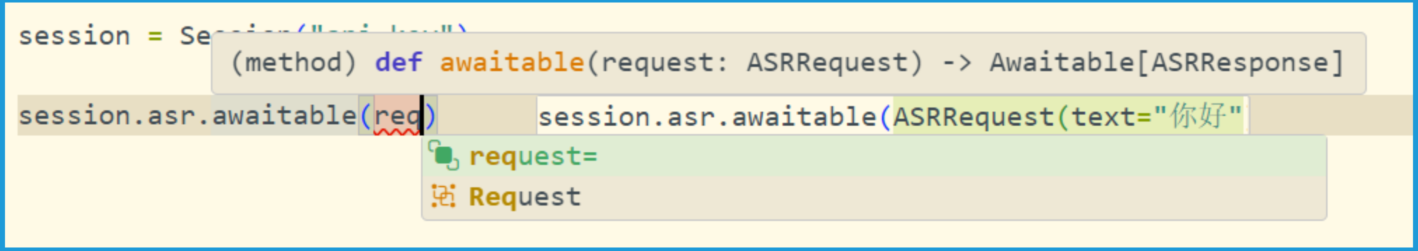
```python
@dataclasses.dataclass
class IOCall(Generic[P, R]):
    _awaitable: Callable[Concatenate[RemoteCall, P], Awaitable[R]]
    _syncable: Callable[Concatenate[RemoteCall, P], R]
    this: RemoteCall

    def __call__(self, *args: P.args, **kwargs: P.kwargs) -> R:
        return self._syncable(self.this, *args, **kwargs)

    def awaitable(self, *args: P.args, **kwargs: P.kwargs) -> Awaitable[R]:
        return self._awaitable(self.this, *args, **kwargs)


class IOCallDescriptor(Generic[P, R]):
    def __init__(
        self,
        awaitable: Callable[Concatenate[RemoteCall, P], Awaitable[R]],
        syncable: Callable[Concatenate[RemoteCall, P], R],
    ):
        self.awaitable = awaitable
        self.syncable = syncable

    def __get__(self, instance: RemoteCall, owner: type[RemoteCall]) -> IOCall[P, R]:
        return IOCall(self.awaitable, self.syncable, instance)
```



```python
class Session(RemoteCall):

    @convert
    def asr(self, request: ASRRequest) -> G[ASRResponse]:
        response = yield Request(
            method="POST",
            url="/v1/asr",
            headers={"Content-Type": "application/msgpack"},
            content=ormsgpack.packb(request, option=ormsgpack.OPT_SERIALIZE_PYDANTIC),
        )
        return ASRResponse.model_validate(response.json())
```



```
session
(property) asr: IOCall[(request: ASRRequest), ASRResponse]
session.asr(request=ASRRequest(text="你好")) CR
request=
Request
```



```
session = Se
(method) def awaitable(request: ASRRequest) -> Awaitable[ASRResponse]
session.asr.awaitable(req)      session.asr.awaitable(ASRRequest(text="你好"
request=
Request
```

?

最佳方案

代码全部出自我的三个开源项目：

- github.com/abersheeran/baize
- github.com/abersheeran/typedmongo
- github.com/fishaudio/fish-audio-python

终