asyncio在Web中的应用

李俱顺

关于我

李俱顺 杭州云柚科技有限公司

Python/Golang开发者，杭州云柚科技联合创始人。先后就职于百度、支付宝等公司。

Blog: https://ipfans.github.io/
E-mail: kevin [at] yeeuu.com

目 录

asyncio简介

PyCon 2016 China

Python的并发问题

一个请求占用了大量的CPU时间，导致程序吞吐量下降。**都怪万恶的GIL?**

常见解决方案

多进程　多线程　green threads

异步编程 ← asyncio 是一个解决方案。

asyncio简介

简单来讲：

异步可以解决什么问题?

IO导致的阻塞问题，常常见于网络通讯过程(Socket/HTTP)中。

asyncio不能解决什么问题?

大量运算导致的CPU时间占用问题。

asyncio简介

asyncio是Python官方的一种异步编程实现。

语法糖支持，与容易混淆的yield写法说再见。

asyncio简介

最早在Python 3.4版本引入标准库。通过yield from语法（pep-380)实现。

```python
# python 3.4
import asyncio

@asyncio.coroutine
def hello():
    print("Hello world!")
    r = yield from asyncio.sleep(1)
    print("Hello again!")

loop = asyncio.get_event_loop()
loop.run_until_complete(hello())
loop.close()
```

可是yield from有点奇怪?

和生成器怎么区分呢?

asyncio简介

然后PEP-0492出现了...

```python
# python 3.5
import asyncio

async def hello():
    print("Hello world!")
    r = await asyncio.sleep(1)
    print("Hello again!")

loop = asyncio.get_event_loop()
loop.run_until_complete(hello())
loop.close()
```

async/await不仅仅是asyncio
可以使用，任何异步循环框架
均可以使用该语法。比如Tornado。

asyncio简介

PyCon 2016 China

PEP-0492适用于多种语法

```
async def read_data(db):
        data = await db.fetch('SELECT ...')

async with session.transaction():
        ...
        await session.update(data)

async for row in Cursor():
        print(row)
```

asyncio简介

async with 实现

将同步的__enter__和__exit__升级为异步__aenter__和__aexit__：

```
class AsyncContextManager:
    async def __aenter__(self):
        await log('entering context')
    async def __aexit__(self, exc_type, exc, tb):
        await log('exiting context')
```

asyncio简介

PyCon China 2016

async for 实现

将同步的__iter__和__next__升级为异步__aiter__和__anext__：

```
class AsyncIterable:
    async def __aiter__(self):
        return self
    async def __anext__(self):
        data = await self.fetch_data()
        if data:
            return data
        else:
            raise StopAsyncIteration
async def fetch_data(self):.....
```

目 录

asyncio的应用落地

PyCon China 2016

项目背景

- 智能硬件管理控制平台。
- 操作同步响应，及时反馈用户。
- 基于MongoDB的简单Map-Reduce。

我们遇到的问题

- 大部分通讯不是实时，导致大量IO通讯等待。
- 数据库进行大量数据处理会导致长时间等待。
- 当使用者数量上升时，应用性能下降明显。

asyncio的应用落地

asyncio的应用落地

PyCon China 2016

asyncio生态

Web框架	数据库	KV存储
aiohttp / Tornado	aiomysql(MySQL) /aiopg / asyncpg	aioredis / ...

更多类库： https://github.com/aio-libs

asyncio的应用落地

PyCon China 2016

我们希望

- 一个类似flask风格的框架
- 可以方便的上手，没有较高学习门槛
- 高自定义度
- 生态相对丰富，想偷懒的时候可以直接伸手

我们的选择: aiohttp/motor/aioredis

asyncio的应用落地

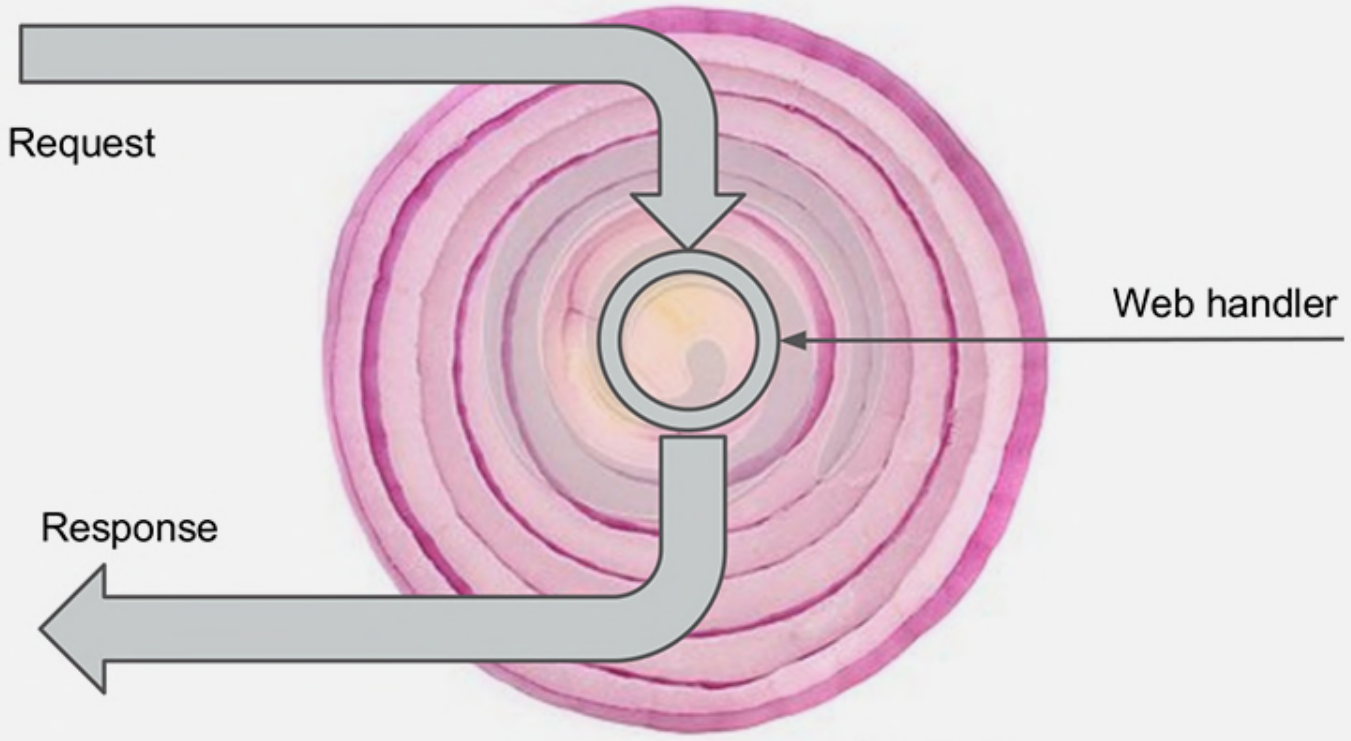
PyCon China 2016

一个简单的例子

```
from aiohttp import web
async def handler(request):
        return web.Response(text="Hello, world")

app = web.Application(loop=loop) app.router.ad–
d_route('GET', '/', handler)
web.run_app(app)
```

asyncio的应用落地



asyncio的应用落地

PyCon China 2016

全局变量保存

```
app = web.Application(loop=loop, logger=logger)
app['db'] = conn
```

使用

```
async with app['db'].acquire() as conn:
    await conn.execute(...)
```

asyncio的应用落地

PyCon China 2016

我们希望

- 一个类似flask风格的框架
- 可以方便的上手，没有较高学习门槛
- 高自定义度
- 生态相对丰富，想偷懒的时候可以直接伸手

我们的选择: aiohttp/motor/aioredis

asyncio的应用落地

PyCon China 2016

项目开发部署

测试环境(单进程测试):

python –m aiohttp.web –H localhost –P 8000

app.app:debug

正式环境(多进程):

gunicorn –b 127.0.0.1:8000 –k aiohttp.worker.–

GunicornWebWorker –w 9 –t 60 app.app

asyncio的应用落地

PyCon China 2016

性能测试

Name	50% (ms)	75% (ms)	Avg (ms)	Req/s	Timeouts
Aiohttp	412.08	433.48	406.53	483	615.20
Muffin	451.77	477.42	449.69	439.1	617.90
Tornado	1044.8	1068.02	1036.97	187.8	103.65
WSGI	2607.26	4677.82	3578.6	18.4	16.15
Falcon	2944.88	4842.57	3629.31	18.35	14.20
Wheezy.Wed	3125.03	4346.63	3573.43	18.35	13.70
Bottle	2885.36	3964.56	3207.14	18.3	15.85
Weppy	2593.71	4492.56	3468.05	18.25	16.15
Flask	2679.56	3990.17	3344.27	18.25	16.15
Django	2908.53	4165.52	3477.36	18.1	14.30
Pyramid	2980.71	3870.58	3305.18	18	14.10

asyncio的应用落地

PyCon China 2016

一些经验

1. 低频率下异步表现有限，高并发下才能体现价值，需根据自己情况选择。

2. asyncio跑分不一定优于gevent，但是坑也不多。

3. aiohttp相对成熟，坑也不多，适合作为线上服务支撑。

4. aiohttp周边基础扩展也相对其他asyncio框架丰富。

目 录

asyncio的一些技巧

PyCon China 2016

PYTHONASYNCIODEBUG=1

开启asyncio的调试模式，等同于 Loop.set_debug()。
可以提供更加详细的错误信息提示

asyncio的一些技巧

异常例子

```
import asyncio
async def bug():
        raise Exception(”not consumed”)
loop = asyncio.get_event_loop()
asyncio.ensure_future(bug())
loop.run_forever()
loop.close()
```

asyncio的一些技巧

```
➜  Desktop python3 async_debug.py
Task exception was never retrieved
future: <Task finished coro=<bug() done, defined at async_debug.py:3> exception=
Exception('not consumed',)>
Traceback (most recent call last):
  File "/Users/kevin/.pyenv/versions/3.5.2/lib/python3.5/asyncio/tasks.py", line
 239, in _step
    result = coro.send(None)
  File "async_debug.py", line 4, in bug
    raise Exception("not consumed")
Exception: not consumed
```

```
➜  Desktop PYTHONASYNCIODEBUG=1 python3 async_debug.py
Task exception was never retrieved
future: <Task finished coro=<bug() done, defined at async_debug.py:3> exception=
Exception('not consumed',) created at async_debug.py:7>
source_traceback: Object created at (most recent call last):
  File "async_debug.py", line 7, in <module>
    asyncio.ensure_future(bug())
Traceback (most recent call last):
  File "/Users/kevin/.pyenv/versions/3.5.2/lib/python3.5/asyncio/tasks.py", line
 239, in _step
    result = coro.send(None)
  File "async_debug.py", line 4, in bug
    raise Exception("not consumed")
Exception: not consumed
```

asyncio的应用落地

PyCon China 2016

显示定义Loop

```
def request(method, url, *, loop=None):
        ...
loop = asyncio.get_event_loop()
await request('GET', 'http://python.org', loop=loop)
```

asyncio的一些技巧

使用asyncio.Semaphore控制并发连接

```
async def bound_fetch(semaphore, url, loop=None):
    async with semaphore:
        await fetch(url, loop)


async def fetch_all():
    tasks = []
    sem = asyncio.Semaphore(100)
    for i in range(10000):
        task = await bound_fetch(sem, "http://www.yeeuu.com", loop)
        tasks.append(task)
    return await asyncio.gather(*tasks)
```

asyncio的应用落地

PyCon China 2016

善用替代loop

Uvloop(https://github.com/MagicStack/uvloop)是一个基于libev和cython的高效异步框架，与asyncio API基本一致，可以无缝替换。

注意： 在某些细节处理上与asyncio有区别，尤其是私有函数。

asyncio的一些技巧

PyCon China 2016

Graceful Restart

```
async def cleanup():
    await app['db'].close()


app.on_shutdown.append(cleanup())
try:
    loop.run_forever()
except KeyboardInterrupt:
    pass
finally:
    # something cleanup
    loop.run_until_complete(app.shutdown())
    loop.run_until_complete(app.cleanup())
loop.close()
```

asyncio的应用落地

Python 3是Future，让我们拥抱它吧！

考虑性能？pypy 3.5正在开发，第一个版本有希望明年发布。

更多的asyncio库正在等待你的Port。XD

FAQ

谢 谢 观 看

K3v1n_lee

kevin@yeeuu.com