asyncio

3.4版本加入标准库。
asyncio底层基于selectors实现，看似库，其实就是个框架，包含异步IO、事件循环、协程、任务等内容。

问题的引出

```
def a():
    for x in range(3):
        print(x)

def b():
    for x in "abc":
        print(x)

a()
b()
# 运行结果一定是
0
1
2
a
b
c
```

这是一个串行的程序，单线程中根本没有做任何

能否并行？

```
import threading
import time

def a():
    for x in range(3):
        time.sleep(0.001) #
        print(x)

def b():
    for x in "abc":
        time.sleep(0.001)
        print(x)

threading.Thread(target=a, name='a').start()
threading.Thread(target=b, name='b').start()

# 运行结果
```

```
0
a
1
b
2
c
```

多进程版本

```
import multiprocessing
import time

def a():
    for x in range(3):
        time.sleep(0.001)
        print(x)

def b():
    for x in "abc":
        time.sleep(0.001)
        print(x)

if __name__ == "__main__":
    multiprocessing.Process(target=a, name='a').start()
    multiprocessing.Process(target=b, name='b').start()

# 运行结果
0
a
1
b
2
c
```

生成器版本

```
def a():
    for x in range(3):
        print(x)
        yield

def b():
    for x in "abc":
        print(x)
        yield

x = a()
y = b()

for i in range(3):
    next(x)
```

```
    next(y)
```

上例在一个线程内通过生成器完成了**调度**，让两个函数都有机会执行，这样的调度不是操作系统的进程、线程完成的，而是用户自己设计的。

这个程序编写：
需要使用yield来让出控制权
需要循环帮助交替执行

事件循环

事件循环是asyncio提供的核心运行机制。

column	column
asyncio.get_event_loop()	返回一个事件循环对象，是asyncio.BaseEventLoop的实例
AbstractEventLoop.stop()	停止运行事件循环
AbstractEventLoop.run_forever()	一直运行，直到stop()
AbstractEventLoop.run_until_complete(future)	运行直至Future对象运行完
AbstractEventLoop.close()	关闭事件循环
AbstractEventLoop.is_running()	返回事件循环的是否运行
AbstractEventLoop.close()	关闭事件循环

协程

- 协程不是进程、也不是线程，它是用户空间调度的完成并发处理的方式。
- 进程、线程由操作系统完成调度，而协程是线程内完成调度。它不需要更多的线程，自然也没有多线程切换带来的开销。
- 协程是非抢占式调度，只有一个协程主动让出控制权，另一个协程才会被调度。
- 协程也不需要使用锁机制，因为是在同一个线程中执行。
- 多CPU下，可以使用多进程和协程配合，既能进程并发又能发挥协程在单线程中的优势。
- Python中协程是基于生成器的。

协程的使用

3.4引入asyncio，使用装饰器

```
import asyncio

@asyncio.coroutine
def sleep(x): # 协程函数
    for i in range(3):
        print('sleep {}'.format(i))
        yield from asyncio.sleep(x)

loop = asyncio.get_event_loop()
loop.run_until_complete(sleep(3))
loop.close()
```

将生成器函数转换成协程函数，就可以在事件循环中执行了。

3.5版本开始，Python提供关键字async、await，在语言上原生支持协程。

```
import asyncio

async def sleep(x):
    for i in range(3):
        print('sleep {}'.format(i))
        await asyncio.sleep(x)

loop = asyncio.get_event_loop()
loop.run_until_complete(sleep(3))
loop.close()
```

async def用来定义协程函数，iscoroutinefunction()返回True。协程函数中可以不包含await、async关键字，但不能使用yield关键字。
如同生成器函数调用返回生成器对象一样，协程函数调用也会返回一个对象称为协程对象，iscoroutine()返回True。

再来做个例子

```
import asyncio
import threading

async def sleep(x):
    for i in range(3):
        print('sleep {}'.format(i))
        await asyncio.sleep(x)

async def showthread(x):
    for i in range(3):
        print(threading.enumerate())
        await asyncio.sleep(2)

loop = asyncio.get_event_loop()
tasks = [sleep(3), showthread(3)]
loop.run_until_complete(asyncio.wait(tasks))
loop.close()
```

```
# 协程版本
import asyncio
import threading

@asyncio.coroutine
def a():
    for x in range(3):
        print('a.x', x)
        yield

@asyncio.coroutine
def b():
    for x in 'abc':
        print('b.x', x)
        yield


print(asyncio.iscoroutinefunction(a))
print(asyncio.iscoroutinefunction(b))

# 大循环
loop = asyncio.get_event_loop()
tasks = [a(), b()]
loop.run_until_complete(asyncio.wait(tasks))
loop.close()
```

TCP Echo Server举例

```
import asyncio

# TCP Echo Server 举例
async def handle(reader, writer):
    while True:
        data = await reader.read(1024)
        print(dir(reader))
        print(dir(writer))
        client = writer.get_extra_info('peername')
        message = "{} Your msg {}".format(client, data.decode()).encode()
        writer.write(message)
        await writer.drain()


loop = asyncio.get_event_loop()
ip = '127.0.0.1'
port = 9999
crt = asyncio.start_server(handle, ip, port, loop=loop)
```

```
server = loop.run_until_complete(crt)
print(server) # server是监听的socket对象
try:
    loop.run_forever()
except KeyboardInterrupt:
    pass
finally:
    server.close()
    loop.close()
```

aiohttp库

安装
$ pip install aiohttp

开发

HTTP Server

```
from aiohttp import web

async def indexhandle(request:web.Request):
    return web.Response(text=request.path, status=201)

async def handle(request:web.Request):
    print(request.match_info)
    print(request.query_string) # http://127.0.0.1:8080/1?name=12301
    return web.Response(text=request.match_info.get('id', '0000'), status=200)

app = web.Application()
app.router.add_get("/", indexhandle) # http://127.0.0.1:8080/
app.router.add_get("/{id}", handle) # http://127.0.0.1:8080/12301

web.run_app(app, host='0.0.0.0', port=9977)
```

HTTP Client

```
import asyncio
from aiohttp import ClientSession

async def get_html(url:str):
    async with ClientSession() as session:
        async with session.get(url) as res:
            print(res.status)
            print(await res.text())


url = 'http://127.0.0.1/ziroom-web/'

loop = asyncio.get_event_loop()
```

```
loop.run_until_complete(get_html(url))
loop.close()
```