

目录

Python教程

Python简介

安装Python

第一个Python程序

Python基础

函数

高级特性

函数式编程

模块

面向对象编程

面向对象高级编程

错误、调试和测试

IO编程

进程和线程

正则表达式

常用内建模块

常用第三方模块

virtualenv

图形界面

网络编程

电子邮件

访问数据库

Web开发

异步IO

协程

asyncio

async/await

aiohttp

实战

FAQ

期末总结

关于作者



廖雪峰

北京 朝阳区

+ 加关注

廖雪峰

自己的Python课程

Python商业爬虫全解码

让天下没有爬不到的数据！

Python爬虫

+

数据分析

Python机器学习

+

深度学习

.....

找廖雪峰老师

廖雪峰老师

自己的Java课程

Java高级架构师

更专业 更权威

源码分析专题

+

微服务架构专题

高并发分布式专题

+

性能优化专题

.....

找廖雪峰老师

async/await

阅读: 151714

用 `asyncio` 提供的 `@asyncio.coroutine` 可以把一个generator标记为coroutine类型，然后在coroutine内部用 `yield from` 调用另一个coroutine实现异步操作。

为了简化并更好地标识异步IO，从Python 3.5开始引入了新的语法 `async` 和 `await`，可以让coroutine的代码更简洁易读。

请注意，`async` 和 `await` 是针对coroutine的新语法，要使用新的语法，只需要做两步简单的替换：

1. 把 `@asyncio.coroutine` 替换为 `async`；
2. 把 `yield from` 替换为 `await`。

让我们对比一下上一节的代码：

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

用新语法重新编写如下：

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

剩下的代码保持不变。

小结

Python从3.5版本开始为 `asyncio` 提供了 `async` 和 `await` 的新语法；

注意新语法只能用在Python 3.5以及后续版本，如果使用3.4版本，则仍需使用上一节的方案。

练习

将上一节的异步获取sina、sohu和163的网站首页源码用新语法重写并运行。

参考源码

[async_hello2.py](#)

[async_wget2.py](#)

读后有收获可以请作者喝咖啡，读后有疑问请加群讨论：



还可以分享给朋友：

分享到微博

廖雪峰

官方 独家

Python

商业爬虫全解码

找廖雪峰老师

广告 X

ACM金牌得主

全球顶尖名企一线数据科学家倾力指导

人工智能与自然语言/计算机视觉课程培训

Artificial Intelligence For NLP/CV Courses

无offer退全款

廖雪峰推荐

JAVA进阶教程

原价1599元

0元领取

阿里云

高性能云服务器首台5折

企业上云事半功倍，最大20Gbps内网带宽，450万PPS

立即选购

5折

python免费公开课

编程学习网

授课模式：在线直播+课后视频，从零基础到中高级开发工程师

查看详情


python免费公开课

编程学习网

授课模式：在线直播+课后视频，从零基础到中高级开发工程师

查看详情


评论



await后面只能加asyncio.xxx的函数？

暗忧殇eee created at February 13, 2018 1:57 PM, Last updated at April 1, 2019 9:57 PM

await后面只能加asyncio.xxx的函数？我尝试自己写了个生成器函数，但是放进去不能用，假如不能自定义，这个携程会不会显得有点鸡肋？



Mr_RightMen

Created at February 28, 2018 5:19 PM

```
async def outer():
    print('in outer')
    print('waiting for result1')
    result1 = await phase1()
    print('waiting for result2')
    result2 = await phase2(result1)
    return (result1,result2)

async def phase1():
    print('in phase1')
    return 'result1'

async def phase2(arg):
    print('in phase2')
    return 'result2 derived from {}'.format(arg)

loop = asyncio.get_event_loop()
tasks = [outer()]
```