关于作者

廖雪峰 V 北京 朝阳区

+加关注

asyncio

阅读: 298326

`asyncio` 是Python 3.4版本引入的标准库，直接内置了对异步IO的支持。

`asyncio` 的编程模型就是一个消息循环。我们从 `asyncio` 模块中直接获取一个 `EventLoop` 的引用，然后把需要执行的协程扔到 `EventLoop` 中执行，就实现了异步IO。

用 `asyncio` 实现 `Hello world` 代码如下：

```
import asyncio

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

# 获取EventLoop:
loop = asyncio.get_event_loop()
# 执行coroutine
loop.run_until_complete(hello())
loop.close()
```

`@asyncio.coroutine` 把一个generator标记为coroutine类型，然后，我们就把这个 `coroutine` 扔到 `EventLoop` 中执行。

`hello()` 会首先打印出 `Hello world!` ，然后，`yield from` 语法可以让我们方便地调用另一个 `generator` 。由于 `asyncio.sleep()` 也是一个 `coroutine` ，所以线程不会等待 `asyncio.sleep()` ，而是直接中断并执行下一个消息循环。当 `asyncio.sleep()` 返回时，线程就可以从 `yield from` 拿到返回值（此处是 `None` ），然后接着执行下一行语句。

把 `asyncio.sleep(1)` 看成是一个耗时1秒的IO操作，在此期间，主线程并未等待，而是去执行 `EventLoop` 中其他可以执行的 `coroutine` 了，因此可以实现并发执行。

我们用Task封装两个 `coroutine` 试试：

```
import threading
import asyncio

@asyncio.coroutine
def hello():
    print('Hello world! (%s)' % threading.currentThread())
    yield from asyncio.sleep(1)
    print('Hello again! (%s)' % threading.currentThread())

loop = asyncio.get_event_loop()
tasks = [hello(), hello()]
loop.run_until_complete(asyncio.wait(tasks))
loop.close()
```

观察执行过程：

```
Hello world! (<_MainThread(MainThread, started 140735195337472)>)
Hello world! (<_MainThread(MainThread, started 140735195337472)>)
(暂停约1秒)
Hello again! (<_MainThread(MainThread, started 140735195337472)>)
Hello again! (<_MainThread(MainThread, started 140735195337472)>)
```

由打印的当前线程名称可以看出，两个 `coroutine` 是由同一个线程并发执行的。

如果把 `asyncio.sleep()` 换成真正的IO操作，则多个 `coroutine` 就可以由一个线程并发执行。

我们用 `asyncio` 的异步网络连接来获取sina、sohu和163的网站首页：

```
import asyncio

@asyncio.coroutine
def wget(host):
    print('wget %s...' % host)
    connect = asyncio.open_connection(host, 80)
    reader, writer = yield from connect
    header = 'GET / HTTP/1.0\r\nHost: %s\r\n\r\n' % host
    writer.write(header.encode('utf-8'))
    yield from writer.drain()
    while True:
        line = yield from reader.readline()
        if line == b'\r\n':
            break
        print('%s header > %s' % (host, line.decode('utf-8').rstrip()))
    # Ignore the body, close the socket
    writer.close()

loop = asyncio.get_event_loop()
tasks = [wget(host) for host in ['www.sina.com.cn', 'www.sohu.com', 'www.163.com']]
loop.run_until_complete(asyncio.wait(tasks))
loop.close()
```

执行结果如下：

```
wget www.sohu.com...
wget www.sina.com.cn...
wget www.163.com...
(等待一段时间)
(打印出sohu的header)
www.sohu.com header > HTTP/1.1 200 OK
www.sohu.com header > Content-Type: text/html
...
(打印出sina的header)
www.sina.com.cn header > HTTP/1.1 200 OK
www.sina.com.cn header > Date: Wed, 20 May 2015 04:56:33 GMT
...
(打印出163的header)
www.163.com header > HTTP/1.0 302 Moved Temporarily
www.163.com header > Server: Cdn Cache Server V2.0
...
```

可见3个连接由一个线程通过 `coroutine` 并发完成。

小结

`asyncio` 提供了完善的异步IO支持；

异步操作需要在 `coroutine` 中通过 `yield from` 完成；

多个 `coroutine` 可以封装成一组Task然后并发执行。

参考源码

async_hello.py

async_wget.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

上一页 下一页

广告 X
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

广告 X
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

报错

思想上的独行侠1 created at February 28, 2019 4:09 PM, Last updated at May 5, 2019 9:06 PM

例子都能运行，也能得出相应的结果，但是每次都报错： RuntimeError: This event loop is already running 这是怎么回事呢？

半九半十

Created at May 5, 2019 9:06 PM

我也是，请问解决了吗

全部讨论 回复