Python协程(gevent)

- 讲师：pansir

简介

协程/微线程/纤程，执行函数A时，可以随时中断，去执行函数B。

优势

1. 执行效率极高
2. 不需要处理多线程的锁机制

gevent

简介

genvent主要模式是greenlet，它是以C扩展模块形式接入Python的轻量级协程。Greenlet全部运行在主程序操作系统进程的内部，但是通过协作式调度。

- 安装：`pip install gevent`

同步与异步执行

子任务之间的切换就是上下文的切换，gevent通过yield进行上下文切换

```python
import gevent

def foo():
    print('开始执行 foo')
    gevent.sleep(0)
    print('切换上下文 foo')


def bar():
    print('开始执行 bar')
    gevent.sleep(0)
    print('切换上下文  bar')

gevent.joinall([
    gevent.spawn(foo),
    gevent.spawn(bar),
])
```

- 模拟IO阻塞调用，协程异步

```python
import time
import gevent
from gevent import select

start = time.time()
tic = lambda: '时间差为:  %1.1f' % (time.time() - start)


def gr1():
    print('gr1开始轮询: %s' % tic())
    select.select([], [], [], 2)
    print('gr1结束轮询: %s' % tic())


def gr2():
    print('gr2开始轮询时间: %s' % tic())
    select.select([], [], [], 2)
    print('gr2结束轮询: %s' % tic())


def gr3():
    print("gr3执行时间, %s" % tic())
    gevent.sleep(1)
    print("gr3执行完毕, %s" % tic())

gevent.joinall([
    gevent.spawn(gr1),
    gevent.spawn(gr2),
    gevent.spawn(gr3),
])
```

- 同步执行与异步执行

```python
import time
import gevent
import random


def task(pid):
    # gevent.sleep(random.randint(0, 2) * 0.001)
    gevent.sleep(0.1)
    print('任务完成: %s' % pid)


def synchronous():
    start = time.time()
    for i in range(0, 10):
        task(i)
    end = time.time()
    print("同步任务执行时间: {}".format(end - start))


def asynchronous():
    start = time.time()
    threads = [gevent.spawn(task, i) for i in range(10)]
    gevent.joinall(threads)
    end = time.time()
    print("异步任务执行时间: {}".format(end - start))

print('执行同步任务:')
synchronous()

print('执行异步任务:')
asynchronous()
```

- 同步与异步发http请求

```
import time

import gevent.monkey

gevent.monkey.patch_socket()

import gevent
import requests


def fetch(pid):
    response = requests.get('http://httpbin.org/get', timeout=50)
    print('请求结果 %s: %s' % (pid, response.status_code))
    return response.text


def synchronous():
    start = time.time()
    for i in range(1, 10):
        fetch(i)
    end = time.time()
    print("同步任务执行时间: {}".format(end - start))


def asynchronous():
    start = time.time()
    threads = []
    for i in range(1, 10):
        threads.append(gevent.spawn(fetch, i))
    gevent.joinall(threads)
    end = time.time()
    print("异步任务执行时间: {}".format(end - start))


print('同步执行:')
synchronous()

print('异步执行:')
asynchronous()
```

创建Greenlets

```
import gevent
from gevent import Greenlet


def foo(message, n):
    gevent.sleep(n)
    print(message)

thread1 = Greenlet.spawn(foo, "我是 Greenlet", 1)
thread2 = gevent.spawn(foo, "我是 gevent", 0.5)
thread3 = gevent.spawn(lambda x: (x + 1), 2)
threads = [thread1, thread2, thread3]
gevent.joinall(threads)
```

- 重写greenlet

```
import gevent
from gevent import Greenlet

class MyGreenlet(Greenlet):

    def __init__(self, message, n):
        Greenlet.__init__(self)
        self.message = message
        self.n = n

    def _run(self):
        print(self.message)
        gevent.sleep(self.n)

g1 = MyGreenlet("重写Greenlet", 3)
g2 = MyGreenlet("我也是Greenlet", 3)
g1.start()
g2.start()
```

```python
g1.join()
g2.join()
```

监控greenlets状态

```python
import gevent

def win():
    return 'success'

def fail():
    raise Exception('failed')


winner = gevent.spawn(win)
loser = gevent.spawn(fail)
print("winner启动状态", winner.started)  # True
print("loser启动状态", loser.started)  # True

try:
    gevent.joinall([winner, loser])
except Exception as e:
    print('协程执行失败')

print("winner是否停止:", winner.ready())  # True
print("loser是否停止:", loser.ready())  # True

print("winner返回值:", winner.value)  # success
print("loser返回值:", loser.value)  # None

print("winner执行结果:", winner.successful())  # True
print("loser执行结果:", loser.successful())  # False

print("loser异常:", loser.exception)
```

超时处理

```python
import time

import gevent
from gevent import Timeout

timeout = Timeout(1)
timeout.start()

def wait():
    gevent.sleep(2)
    print("任务完成")

try:
    gevent.spawn(wait).join()
except Timeout:
    print('任务中断')
```

```
任务中断
```

协程编程

数据消费

通过队列，防止数据被重复使用

```
import gevent
from gevent.queue import Queue

q1 = Queue()

def consumer(name):
    while not q1.empty():
        print('%s 从队列获取数据 %s' % (name, q1.get()))
        gevent.sleep(0)
    print(name, '退出')


def producer():
    for i in range(1, 10):
        q1.put(i)

gevent.joinall([
    gevent.spawn(producer),
    gevent.spawn(consumer, 'liudan'),
    gevent.spawn(consumer, 'shangcl'),
    gevent.spawn(consumer, 'hhc'),
])
```

协程局部变量

通过local限制协程变量访问

```
import gevent
from gevent.local import local

data = local()
```

```
def f1():
    data.x = 1
    print(data.x)


def f2():
    try:
        print(data.x)
    except AttributeError:
        print("访问异常")


gevent.joinall([
    gevent.spawn(f1),
    gevent.spawn(f2)
])
```

猴子补丁

因为python标准库的socket是阻塞式的，可以使用猴子补丁，会给所有的python库打上补丁(替换)，从而变成非阻塞的

- 获取IP地址

```
import time

from gevent import monkey;monkey.patch_all()
import gevent
import socket

urls = ['www.baidu.com', 'www.gevent.org', 'www.python.org']

def get_time(url):
    start = time.time()
    ip = socket.gethostbyname(url)
    time.sleep(1)
    end = time.time()
    print("耗时", end - start, url)
    return ip

start = time.time()
jobs = [gevent.spawn(get_time, url) for url in urls]
gevent.joinall(jobs, timeout=5)
end = time.time()
print("总耗时", end - start)
print([job.value for job in jobs])
```