12.13 多个线程队列轮询¶

问题¶

你有一个线程队列集合，想为到来的元素轮询它们， 就跟你为一个客户端请求去轮询一个网络连接集合的方式一样。

解决方案¶

对于轮询问题的一个常见解决方案中有个很少有人知道的技巧，包含了一个隐藏的回路网络连接。 本质上讲其思想就是：对于每个你想要轮询的队列，你创建一对连接的套接字。 然后你在其中一个套接字上面编写代码来标识存在的数据， 另外一个套接字被传给 `select()` 或类似的一个轮询数据到达的函数。下面的例子演示了这个思想：

```
import queue
import socket
import os

class PollableQueue(queue.Queue):
    def __init__(self):
        super().__init__()
        # Create a pair of connected sockets
        if os.name == 'posix':
            self._putsocket, self._getsocket = socket.socketpair()
        else:
            # Compatibility on non-POSIX systems
            server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            server.bind(('127.0.0.1', 0))
            server.listen(1)
            self._putsocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            self._putsocket.connect(server.getsockname())
            self._getsocket, _ = server.accept()
            server.close()

    def fileno(self):
        return self._getsocket.fileno()

    def put(self, item):
        super().put(item)
        self._putsocket.send(b'x')

    def get(self):
        self._getsocket.recv(1)
        return super().get()
```

在这个代码中，一个新的 `Queue` 实例类型被定义，底层是一个被连接套接字对。 在Unix机器上的 `socketpair()` 函数能轻松的创建这样的套接字。 在Windows上面，你必须使用类似代码来模拟它。 然后定义普通的 `get()` 和 `put()` 方法在这些套接字上面来执行I/O操作。 `put()` 方法再将数据放入队列后会写一个单字节到某个套接字中去。 而 `get()` 方法在从队列中移除一个元素时会从另外一个套接字中读取到这个单字节数据。

`fileno()` 方法使用一个函数比如 `select()` 来让这个队列可以被轮询。 它仅仅只是暴露了底层被 `get()` 函数使用到的socket的文件描述符而已。

下面是一个例子，定义了一个为到来的元素监控多个队列的消费者：

```
import select
import threading

def consumer(queues):
    '''
    Consumer that reads data on multiple queues simultaneously
    '''
    while True:
        can_read, _, _ = select.select(queues,[],[])
        for r in can_read:
            item = r.get()
```

```
            print('Got:', item)

q1 = PollableQueue()
q2 = PollableQueue()
q3 = PollableQueue()
t = threading.Thread(target=consumer, args=([q1,q2,q3],))
t.daemon = True
t.start()

# Feed data to the queues
q1.put(1)
q2.put(10)
q3.put('hello')
q2.put(15)
...
```

如果你试着运行它，你会发现这个消费者会接受到所有的被放入的元素，不管元素被放进了哪个队列中。

讨论¶

对于轮询非类文件对象，比如队列通常都是比较棘手的问题。 例如，如果你不使用上面的套接字技术， 你唯一的选择就是编写代码来循环遍历这些队列并使用一个定时器。像下面这样：

```
import time
def consumer(queues):
    while True:
        for q in queues:
            if not q.empty():
                item = q.get()
                print('Got:', item)

        # Sleep briefly to avoid 100% CPU
        time.sleep(0.01)
```

这样做其实不合理，还会引入其他的性能问题。 例如，如果新的数据被加入到一个队列中，至少要花10毫秒才能被发现。 如果你之前的轮询还要去轮询其他对象，比如网络套接字那还会有更多问题。 例如，如果你想同时轮询套接字和队列，你可能要像下面这样使用：

```
import select

def event_loop(sockets, queues):
    while True:
        # polling with a timeout
        can_read, _, _ = select.select(sockets, [], [], 0.01)
        for r in can_read:
            handle_read(r)
        for q in queues:
            if not q.empty():
                item = q.get()
                print('Got:', item)
```

这个方案通过将队列和套接字等同对待来解决了大部分的问题。 一个单独的 `select()` 调用可被同时用来轮询。 使用超时或其他基于时间的机制来执行周期性检查并没有必要。 甚至，如果数据被加入到一个队列，消费者几乎可以实时的被通知。 尽管会有一点点底层的I/O损耗，使用它通常会获得更好的响应时间并简化编程。