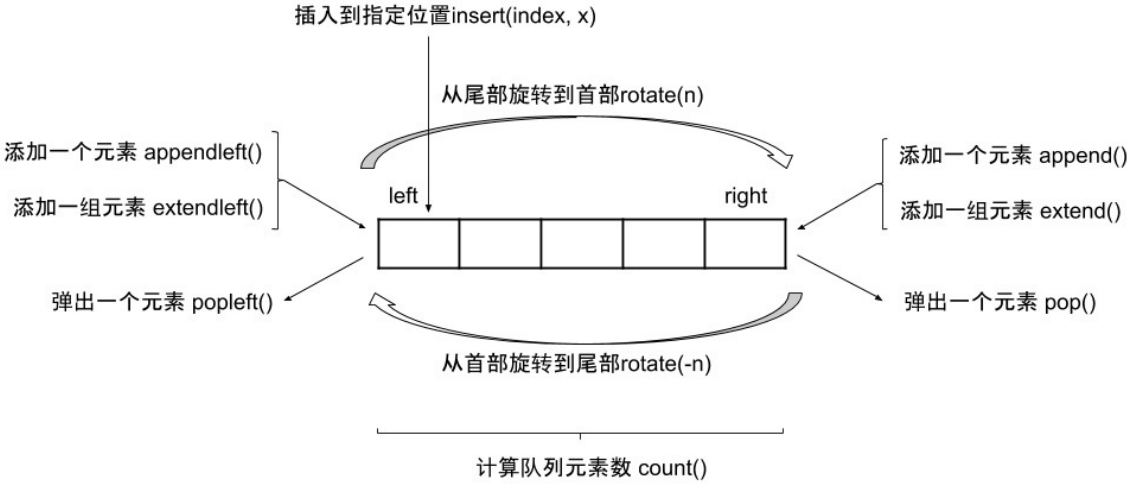
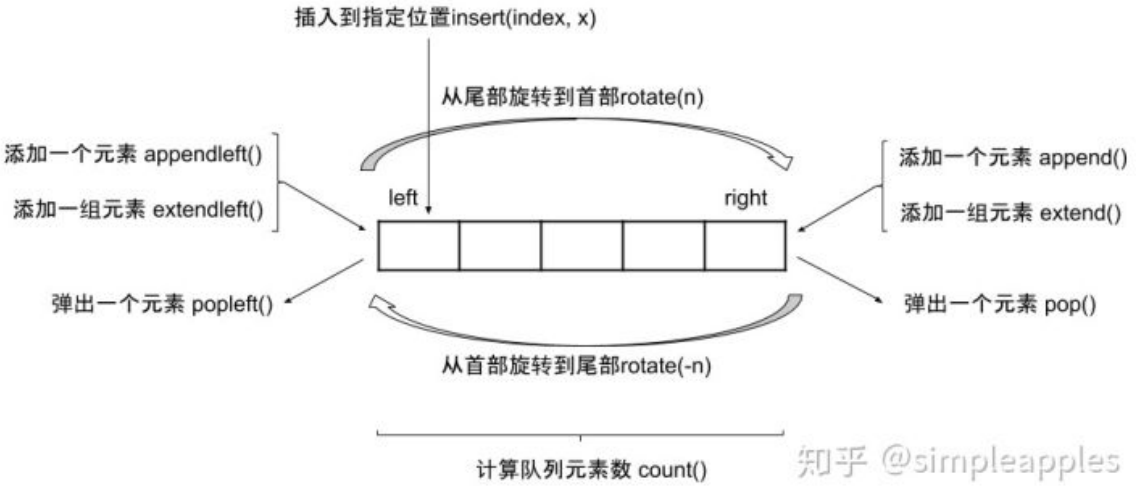


简析Python中的四种队列

队列是一种只允许在一端进行插入操作，而在另一端进行删除操作的线性表。
在Python文档中搜索队列（queue）会发现，Python标准库中包含了四种队列，分别是queue.Queue / asyncio.Queue / multiprocessing.Queue / collections.deque。

collections.deque

deque是双端队列（double-ended queue）的缩写，由于两端都能编辑，deque既可以用来实现栈（stack）也可以用来实现队列（queue）。
deque支持丰富的操作方法，主要方法如图：



相比于list实现的队列，deque实现拥有更低的时间和空间复杂度。list实现在出队（pop）和插入（insert）时的空间复杂度大约为O(n)，deque在出队（pop）和入

队（append）时的时间复杂度是O(1)。
deque也支持in操作符，可以使用如下写法：

```
q = collections.deque([1, 2, 3, 4])
print(5 in q)  # False
print(1 in q)  # True
```

deque还封装了顺逆时针的旋转的方法：rotate。

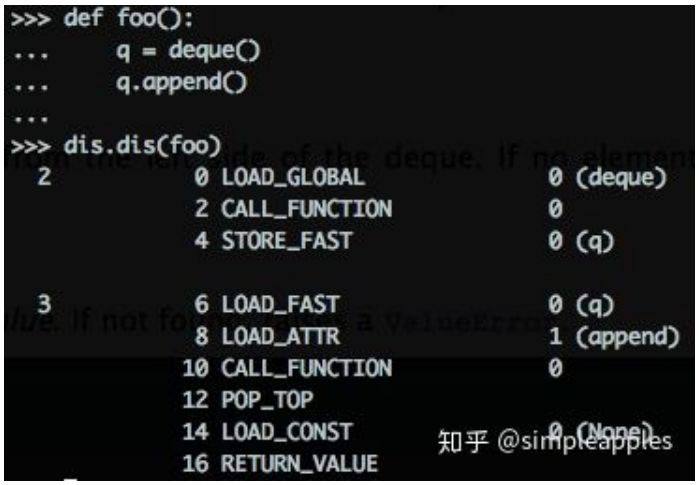
```
# 顺时针
q = collections.deque([1, 2, 3, 4])
q.rotate(1)
print(q)  # [4, 1, 2, 3]
q.rotate(1)
print(q)  # [3, 4, 1, 2]

# 逆时针
q = collections.deque([1, 2, 3, 4])
q.rotate(-1)
print(q)  # [2, 3, 4, 1]
q.rotate(-1)
print(q)  # [3, 4, 1, 2]
```

线程安全方面，通过查看collections.deque中的append()、pop()等方法的源码可以知道，他们都是原子操作，所以是GIL保护下的线程安全方法。

```
static PyObject *
deque_append(dequeobject *deque, PyObject *item) {
    Py_INCREF(item);
    if (deque_append_internal(deque, item, deque->maxlen) < 0)
        return NULL;
    Py_RETURN_NONE;
}
```

通过dis方法可以看到，append是原子操作（一行字节码）。



综上，collections.deque是一个可以方便实现队列的数据结构，具有线程安全的特性，并且有很高的性能。

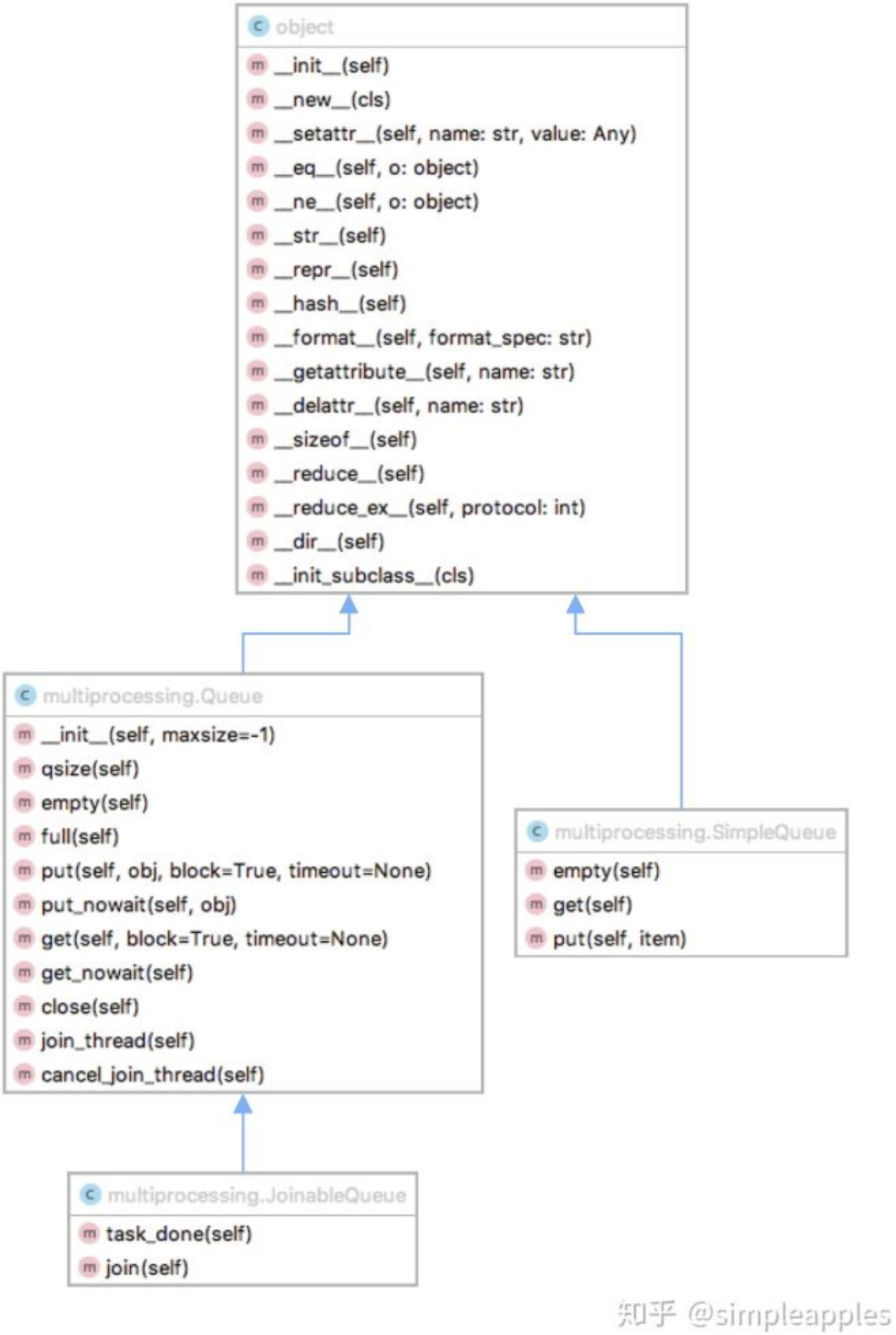
queue.Queue & asyncio.Queue

queue.Queue和asyncio.Queue都是支持多生产者、多消费者的队列，基于collections.deque，他们都提供了Queue（FIFO队列）、PriorityQueue（优先级队列）、LifoQueue（LIFO队列），接口方面也相同。
区别在于queue.Queue适用于多线程的场景，asyncio.Queue适用于协程场景下的通信，由于asyncio的加成，queue.Queue下的阻塞接口在asyncio.Queue中则是以返回协程对象的方式执行，具体差异如下表：

	queue.Queue	asyncio.Queue
介绍	同步队列	asyncio队列
线程安全	是	否
超时机制	通过timeout参数实现	通过asyncio.wait_for()方法实现
qsize()	预估的队列长度（获取qsize到下一个操作之间，queue有可能被其它的线程修改，导致qsize大小发生变化）	准确的队列长度（由于是单线程，所以queue不会被其它线程修改）
put() / set()	put(item, block=True, timeout=None)，可以通过设置block是否为True来配置put和set方法是否为阻塞，并且可以为阻塞操作设置最大时长timeout，block为False时行为和put_nowait()方法一致。	put()方法会返回一个协程对象，所以没有block参数和timeout参数，如果需要非阻塞方法，可以使用put_nowait()，如果需要对阻塞方法应用超时，可以使用coroutine asyncio.wait_for()。

multiprocessing.Queue

multiprocessing提供了三种队列，分别是Queue、SimpleQueue、JoinableQueue。



multiprocessing.Queue既是线程安全也是进程安全的，相当于queue.Queue的多进程克隆版。和threading.Queue很像，multiprocessing.Queue支持put和get操作，

底层结构是multiprocessing.Pipe。
multiprocessing.Queue底层是基于Pipe构建的，但是数据传递时并不是直接写入Pipe，而是写入进程本地buffer，通过一个feeder线程写入底层Pipe，这样做是为了实现超时控制和非阻塞put/get，所以Queue提供了join_thread、cancel_join_thread、close函数来控制feeder的行为，close函数用来关闭feeder线程、join_thread用来join feeder线程，cancel_join_thread用来在控制在进程退出时，不自动join feeder线程，使用cancel_join_thread有可能导致部分数据没有被feeder写入Pipe而导致的数据丢失。
和threading.Queue不同的是，multiprocessing.Queue默认不支持join()和task_done操作，这两个支持需要使用mp.JoinableQueue对象。
SimpleQueue是一个简化的队列，去掉了Queue中的buffer，没有了使用Queue可能出现的问题，但是put和get方法都是阻塞的并且没有超时控制。

总结

通过对比可以发现，上述四种结构都实现了队列，但是用处却各有偏重，collections.deque在数据结构层面实现了队列，但是并没有应用场景方面的支持，可以看做是一个基础的数据结构。queue模块实现了面向多生产线程、多消费线程的队列，asyncio.queue模块则实现了面向多生产协程、多消费协程的队列，而multiprocessing.queue模块实现了面向多成产进程、多消费进程的队列。

参考

https://docs.python.org/3/library/collections.html#collections.deque
https://docs.python.org/3/library/queue.html
https://docs.python.org/3/library/asyncio-queue.html
https://docs.python.org/3/library/multiprocessing.html#multiprocessing.Queue
https://bugs.python.org/issue15329
http://blog.ftofficer.com/2009/12/python-multiprocessing-3-about-queue/
http://cyrusin.github.io/2016/04/27/python-gil-implementaion/