Python

**Queue 的分析**

队列模块实现多生产者，多消费者队列。当必须在多个线程之间安全地交换信息时，它在线程编程中特别有用。

队列模块定义以下类和异常：

Queue.Queue类（maxsize = 0）

FIFO队列的构造函数。 maxsize是一个整数，用于设置可以放入队列中的对象数的上限。一旦达到此大小，插入将被阻塞，直到消耗队列对象为止。如果maxsize小于或等于零，则队列大小为无限。

Queue.PriorityQueue（maxsize = 0）类

优先级队列的构造函数。 maxsize是一个整数，用于设置可以放入队列中的对象数的上限。一旦达到此大小，插入将被阻塞，直到消耗队列对象为止。如果maxsize小于或等于零，则队列大小为无限。

最低值的条目首先被检索（最低值的条目是sorted（list（entries））[0]返回的条目）。条目的典型模式是元组，其形式为：（priority\_number，数据）。

异常Queue.Empty

在空的Queue对象上调用非阻塞get（）（或get\_nowait（））时引发的异常。

异常Queue.Full

在已满的Queue对象上调用非阻塞put（）（或put\_nowait（））时引发的异常。

队列对象（Queue或PriorityQueue）提供了下面描述的公共方法。

Queue.qsize（）

返回队列的大概大小。请注意，qsize（）> 0不能保证后续的get（）不会阻塞，qsize（）<maxsize也不能保证put（）不会阻塞。

Queue.empty（）

如果队列为空，则返回True，否则返回False。如果empty（）返回True，则不能保证后续对put（）的调用都不会阻塞。同样，如果empty（）返回False，则不能保证后续对get（）的调用都不会阻塞。

Queue.full（）

如果队列已满，则返回True，否则返回False。如果full（）返回True，则不能保证后续对get（）的调用都不会阻塞。同样，如果full（）返回False，则不能保证后续对put（）的调用都不会阻塞。

Queue.put（item [，block [，timeout]]）

将对象放入队列。如果可选的args块为true，并且超时为None（默认值），则在必要时进行阻塞，直到有可用的插槽可用为止。如果超时为正数，则在最多超时秒内阻塞，并在该时间内没有可用插槽的情况下引发Full异常。否则（块为false），如果有空闲插槽立即可用，则将对象放在队列中，否则引发Full异常（在这种情况下，超时将被忽略）。

Queue.put\_nowait（对象）

等效于put（item，False）。

Queue.get（[block [，timeout]]）

从队列中删除并返回一个对象。如果可选的args块为true，并且超时为None（默认值），则在必要时进行阻塞，直到有可用项为止。如果超时为正数，则它最多会阻塞超时秒，如果在该时间内没有可用的对象，则会引发Empty异常。否则（块为false），如果一项立即可用，则返回一个对象，否则引发Empty异常（在这种情况下将忽略超时）。

Queue.get\_nowait（）

等效于get（False）。

提供了两种方法来支持跟踪排队任务是否已由守护程序使用者线程完全处理。

Queue.task\_done（）

表示先前排队的任务已完成。由队列使用者线程使用。对于用于提取任务的每个get（），随后对task\_done（）的调用将告诉队列该任务的处理已完成。

如果join（）当前正在阻塞，它将在所有对象都已处理完毕后恢复（这意味着已将每个put（）放入队列的对象都收到了task\_done（）调用）。

如果调用的次数超过队列中放置的对象的次数，则引发ValueError。

queue.join（）

阻塞直到队列中的所有对象都已获得并处理。

每当将对象添加到队列时，未完成任务的数量就会增加。每当使用者线程调用task\_done（）表示已检索到该对象并且该对象的所有工作已完成时，该计数就会减少。当未完成的任务数降至零时，join（）会解除阻止。

**Priority queue 的分析**

Python 的优先级队列是采用堆队列的算法，堆是一个二叉树，这个从0计数元素的操作对所有的范围在head[k] <= heap[2\*(k+1)] 和 heap[k]<= heap [2\*k+2]是用数组（array）实现的。 而且为了比较，不存在无限大的元素，最小的堆元素是根节点，即heap[0]。

堆算法有两点：1.以0为基准的索引；2.pop 方法返回最小的对象。

可以把堆看做是常规的list, heap[0]是最小的对象，heap.sort() 维持堆不变

创建堆，用list初始化，即[], 或者通过heapify() 把一个填充好的list 转化成heap

heap 的函数：

heapq.heappush(heap, item)

将值项推入堆，保持堆不变。

heapq.heappop(heap)

弹出并从堆中返回最小的项，以保持堆不变。如果堆为空，则会引发IndexError。要访问最小的对象而不弹出它，请使用heap [0]。

heapq.heappushpop(heap, item)

将对象推入堆中，然后弹出并从堆中返回最小的对象。组合的操作比heappush（）更有效地运行，随后分别调用heappop（）

New in version 2.6.

heapq.heapify(x)

将列表x在线性时间内就地转换为堆。

heapq.heapreplace(heap, item)

弹出并从堆中返回最小的对象，然后推送新对象。堆大小不变。如果堆为空，则会引发IndexError。

一步操作比heapush（）和heappush（）效率更高，并且在使用固定大小的堆时可能更合适。弹出/推入组合始终从堆中返回一个元素，并将其替换为item。

返回的值可能大于添加的对象。如果不需要，请考虑改用heappushpop（）。它的推/弹出组合返回两个值中较小的一个，而将较大的值保留在堆上。

heapq.merge(\*iterables)

将多个排序的输入合并到一个排序的输出中（例如，合并多个日志文件中带有时间戳的条目）。返回排序后的值的迭代器。

类似于sorted（itertools.chain（\* iterables）），但返回一个iterable，不会一次将数据全部拉入内存，并假定每个输入流都已排序（最小到最大）。

heapq.nlargest(n, iterable[, key])

从iterable定义的数据集中返回包含n个最大元素的列表。 key（如果提供）指定一个参数的功能，该参数用于从iterable中的每个元素中提取比较键：key = str.lower等效于：sorted（iterable，key = key，reverse = True）[：n]

heapq.nsmallest(n, iterable[, key])

从iterable定义的数据集中返回包含n个最小元素的列表。 key（如果提供）指定一个参数的功能，该参数用于从iterable中的每个元素中提取比较键：key = str.lower等效于：sorted（iterable，key = key）[：n]

Changed in version 2.5: Added the optional key argument.

对于n的较小值，后两个函数表现最佳。对于较大的值，使用sorted（）函数更为有效。同样，当n == 1时，使用内置的min（）和max（）函数会更有效。如果需要重复使用这些功能，请考虑将可迭代对象转换为实际堆。