Сообщить об ошибке.

Модуль queue, очереди в Python

Потоко-безопасные очереди FIFO, LIFO и очереди с приотитетом

Модуль queue реализует очереди с несколькими производителями и несколькими потребителями. Это особенно полезно в потоковом программировании, когда информация должна безопасно обмениваться между несколькими потоками. Класс queue.Queue() в этом модуле реализует всю необходимую семантику блокировки.

Модуль реализует три типа очереди, которые отличаются только порядком, в котором извлекаются записи:

- В очереди FIFO первые добавленные задачи являются первыми извлеченными.
- В очереди LIFO самая последняя добавленная запись является первой извлеченной (работающей как стек).
- В очереди с приоритетами записи сохраняются отсортированными с использованием модуля heapq и сначала извлекается запись с наименьшим значением.

Внутренне эти три типа очередей используют блокировки для временного блокирования конкурирующих потоков, однако они не предназначены для обработки повторного входа в поток.

Кроме того, модуль реализует простой тип очереди FIFO - queue.SimpleQueue(), специфическая реализация которого обеспечивает дополнительные гарантии в обмен на меньшую функциональность.

Примеры использования модуля queue.

Очередь FIFO:

Класс queue.Queue() реализует базовый контейнер типа FIFO - "первым пришел - первым вышел". Элементы добавляются к одному концу очереди с помощью метода put(), а удаляются с другого конца с помощью метода get().

```
import queue

q = queue.Queue()

for i in range(5):
    q.put(i)

while not q.empty():
    print(q.get(), end=' ')

# 0 1 2 3 4
```

Очередь LIFO:

В отличие от стандартной реализации очереди FIFO, в queue.LifoQueue() используется порядок "последним пришел - первым вышел", который обычно связан со структурой данных стека.

```
import queue

q = queue.LifoQueue()

for i in range(5):
    q.put(i)

while not q.empty():
    print(q.get(), end=' ')
```

```
# 4 3 2 1 0
```

Очередь с приоритетом:

Иногда порядок обработки элементов в очереди должен основываться на характеристиках этих элементов, а не только на порядке их создания или добавления в очередь. Например, задания на печать из финансового отдела могут иметь приоритет над списком заданий из отдела технической поддержки. Класс модуля queue.PriorityQueue() использует порядок сортировки содержимого очереди, чтобы решить, какой элемент получить.

```
import functools
import queue
import threading


@functools.total_ordering
class Job:

    def __init__(self, priority, description):
        self.priority = priority
        self.description = description
        return

    def __eq__(self, other):
        try:
            return self.priority == other.priority
        except AttributeError:
            return NotImplemented

    def __lt__(self, other):
        try:
            return self.priority < other.priority
        except AttributeError:
            return NotImplemented

def process_job(q):
    while True:
        next_job = q.get()
        print('Processing job:', next_job.description)
        q.task_done()

q = queue.PriorityQueue()
q.put(Job(3, 'Mid-level'))
q.put(Job(10, 'Low-level'))
q.put(Job(1, 'Important'))

workers = [
    threading.Thread(target=process_job, args=(q,)),
    threading.Thread(target=process_job, args=(q,)),
]
for w in workers:
    w.setDaemon(True)
    w.start()

q.join()
```

Этот пример имеет несколько потоков, потребляющих задания, которые обрабатываются на основе приоритета элементов в очереди на момент вызова get(). Порядок обработки элементов, добавляемых в очередь во время работы потоков-потребителей, зависит от переключения контекста потока.

```
Processing job: Important job
Processing job: Mid-level job
Processing job: Low-level job
```

Содержание раздела: