18.5.8. Queues

Source code: Lib/asyncio/queues.py

Queues:

- `Queue`
- `PriorityQueue`
- `LifoQueue`

asyncio queue API was designed to be close to classes of the `queue` module (`Queue`, `PriorityQueue`, `LifoQueue`), but it has no *timeout* parameter. The `asyncio.wait_for()` function can be used to cancel a task after a timeout.

18.5.8.1. Queue

class `asyncio.`**`Queue`**(*maxsize=0*, ***, *loop=None*)

A queue, useful for coordinating producer and consumer coroutines.

If *maxsize* is less than or equal to zero, the queue size is infinite. If it is an integer greater than `0`, then `yield from put()` will block when the queue reaches *maxsize*, until an item is removed by `get()`.

Unlike the standard library `queue`, you can reliably know this Queue's size with `qsize()`, since your single-threaded asyncio application won't be interrupted between calling `qsize()` and doing an operation on the Queue.

This class is not thread safe.

Changed in version 3.4.4: New `join()` and `task_done()` methods.

`empty`()

Return `True` if the queue is empty, `False` otherwise.

`full`()

Return `True` if there are `maxsize` items in the queue.

> **Note:** If the Queue was initialized with `maxsize=0` (the default), then `full()` is never `True`.

coroutine **`get`**()

Remove and return an item from the queue. If queue is empty, wait until an item is available.

This method is a coroutine.

> **See also:** The `empty()` method.

`get_nowait`()

Remove and return an item from the queue.

Return an item if one is immediately available, else raise `QueueEmpty`.

coroutine `join`()

Block until all items in the queue have been gotten and processed.

The count of unfinished tasks goes up whenever an item is added to the queue. The count goes down whenever a consumer thread calls `task_done()` to indicate that the item was retrieved and all work on it is complete. When the count of unfinished tasks drops to zero, `join()` unblocks.

This method is a coroutine.

New in version 3.4.4.

coroutine `put`(*item*)

Put an item into the queue. If the queue is full, wait until a free slot is available before adding item.

This method is a coroutine.

> **See also:** The `full()` method.

`put_nowait`(*item*)

Put an item into the queue without blocking.

If no free slot is immediately available, raise `QueueFull`.

`qsize`()

Number of items in the queue.

`task_done`()

Indicate that a formerly enqueued task is complete.

Used by queue consumers. For each `get()` used to fetch a task, a subsequent call to `task_done()` tells the queue that the processing on the task is complete.

If a `join()` is currently blocking, it will resume when all items have been processed (meaning that a `task_done()` call was received for every item that had been `put()` into the queue).

Raises `ValueError` if called more times than there were items placed in the queue.

New in version 3.4.4.

`maxsize`

Number of items allowed in the queue.

18.5.8.2. PriorityQueue

class `asyncio.`**`PriorityQueue`**

A subclass of `Queue`; retrieves entries in priority order (lowest first).

Entries are typically tuples of the form: (priority number, data).

18.5.8.3. LifoQueue

class `asyncio.`**`LifoQueue`**

A subclass of `Queue` that retrieves most recently added entries first.

18.5.8.3.1. Exceptions

exception `asyncio.`**`QueueEmpty`**

Exception raised when the `get_nowait()` method is called on a `Queue` object which is empty.

exception `asyncio.`**`QueueFull`**

Exception raised when the `put_nowait()` method is called on a `Queue` object which is full.