《QuecPython队列应用之自动锁&条件变量&事件》

一、前言

鉴于QuecPython模组中没有可用的事件、自动锁和条件变量等应用（与cpython标准库类似）。

二、自动锁

所谓的自动锁，其实就是提供一种自动加锁和自动解锁的机制。用户在使用的使用，就可以不需要关注加锁解锁流程这样就可以大大的减少了如忘记解锁从而导致的死锁等一系列问题。

> 加锁：即请求锁。解锁：即释放锁。

实现原理：通过python的上下文管理器，实现自动加锁和解锁。

```
import _thread


class AutoLock(object):
    def __init__(self):
        self.__lock = _thread.allocate_lock()

    def acquire(self):
        self.__lock.acquire()

    def release():
        self.__lock.release()

    def __enter__(self):
        self.__lock.acquire()

    def __exit__(self, exc_type, exc_val, exc_tb):
        self.__lock.release()


if __name__ == "__main__":
    lock = AutoLock()
    with lock:
        # TODO: do something
        pass
```

四、条件变量

条件变量允许一个或多个线程等待，直到它们由另一个线程通知。使用对象的`wait()`方法产生阻塞，指导任意线程调用`notify()`方法解除阻塞。

> 此处使用模组中的阻塞队列实现。

```
import _thread
from queue import Queue


```

```
class Condition(object):
    def __init__(self, queue=Queue(maxsize=1)):
    self.__block_queue = queue

    def wait(self):
        # return a *notify party*(`identity`), see .notify() method
        return self.__block_queue.get()

    def notify(self, identity=None):
        # `identity` is a param defined by customer, means *notify party*
        self.__block_queue.put(identity)
```

三、事件

所谓的事件，其实就是线程同步的一种方式。比如，A线程可以判定B线程是否产生的某一个事件。事件对象管理一个标志变量，该标志可以使用`set()`方法设置为true并重置使用`clear()`方法设置为false。`wait()`方法阻塞，直到标志为真的。该标志最初为false。

```
import _thread


class Event(object):
    def __init__(self):
    self.__cond = Condition()
    self.__lock = AutoLock()
    self.flag = False

    def set(self):
        with self.__lock():
            self.flag = True
            self.__cond.notify()

    def clear(self):
        with self.__lock:
            self.flag = False

    def isSet(self):
        return self.flag

    def wait(self):
        self.__cond.wait()
        return self.flag
```