82 QWaitCondition

82.1 睡眠与唤醒

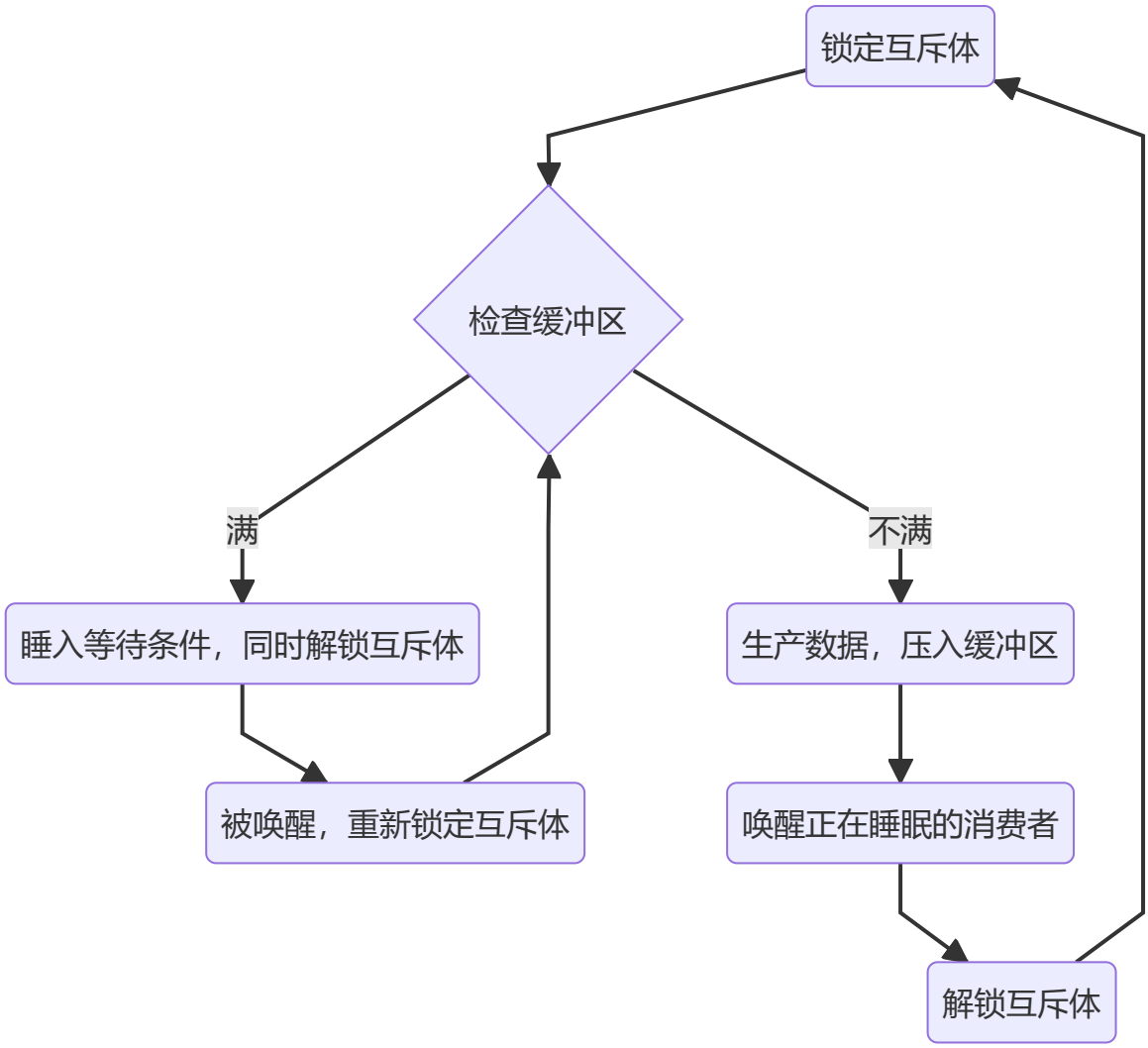
QWaitCondition类所表示的等待条件，可以与QMutex类所表示的互斥体结合使用，在执行操作的条件不满足时，陷入睡眠，同时解锁所持有的互斥体，以使其它线程得以执行为其创造条件的操作，待条件满足时再被唤醒，重新锁定互斥体后，执行那些有赖于该条件的操作。

```
bool QWaitCondition::wait(QMutex* lockedMutex, // 陷入睡眠，同时解锁lockedMutex，
    unsigned long time = ULONG_MAX);           // 被唤醒后，重新锁定lockedMutex，
并返回
void QWaitCondition::wakeAll();                // 唤醒所有在该等待条件中睡眠的线
程，但只有一个线程能成功锁定互斥体，并从wait中返回
void QWaitCondition::wakeOne();                // 唤醒一个在该等待条件中睡眠的线
程，该线程在重新锁定互斥体后从wait调用中返回
```

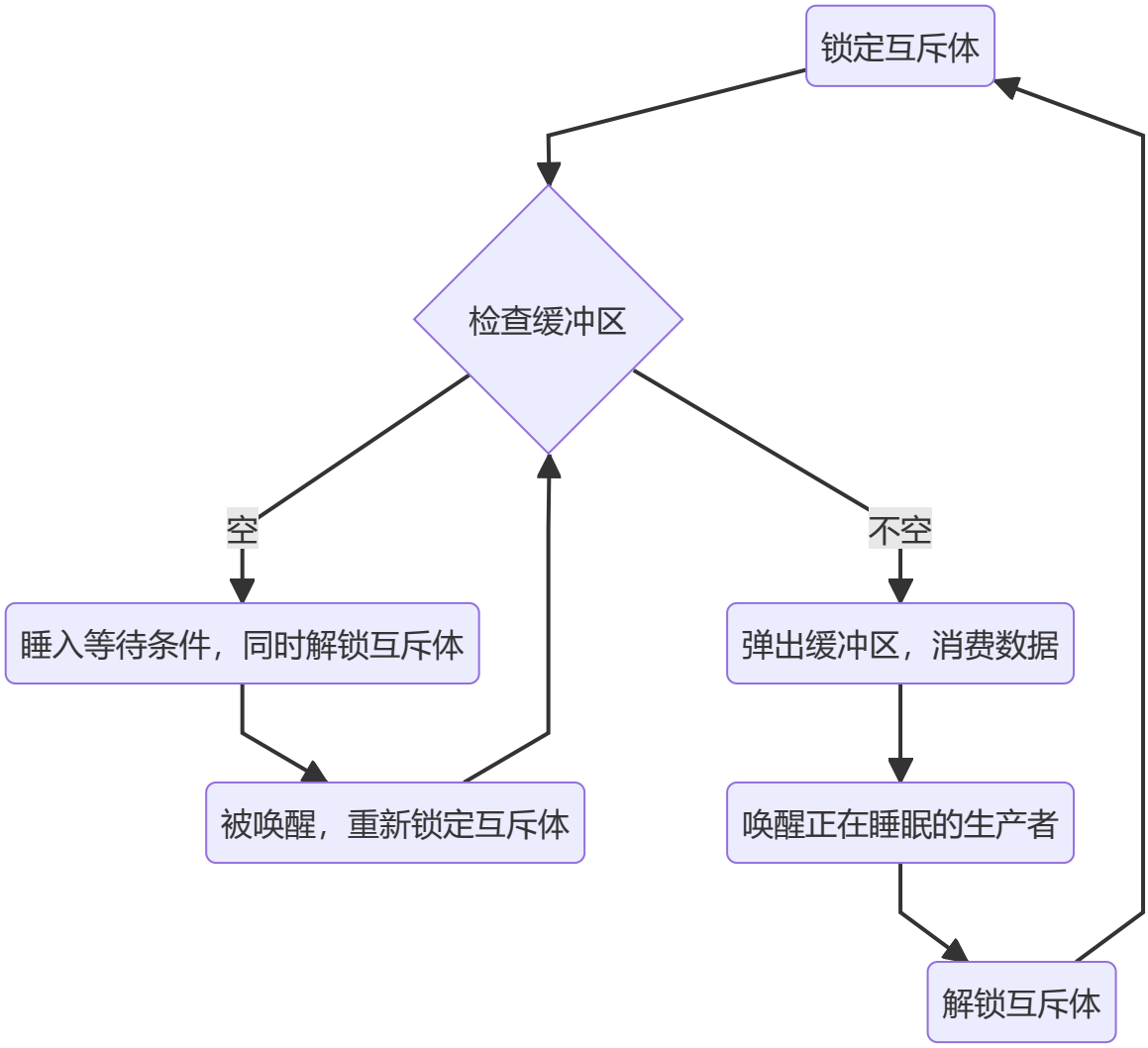
82.2 生产者/消费者模型

生产者负责生产数据，消费者负责消费数据，二者共享同一个数据缓冲区。

82.2.1 生产者线程



82.2.2 消费者线程



82.3 案例

82.3.1 创建项目

通过QtCreator，在C:\Users\Minwei\Projects\Qt路径下，创建名为WaitCondition的控制台（Console）项目。

82.3.2 实现功能

C:\Users\Minwei\Projects\Qt\WaitCondition\main.cpp：

```
#include <iostream>
using namespace std;

#include <QCoreApplication>
#include <QThread>
#include <QMutex>
#include <QWaitCondition>
#include <QRandomGenerator>

#define MAX_STOCK 10

char g_storage[MAX_STOCK];
```

```
size_t g_stock = 0;

QMutex g_mutex;
QWaitCondition g_nonFull;
QWaitCondition g_nonEmpty;

void show (char const* name, char const* op, char prod) {
    cout << name << "> ";

    for (size_t i = 0; i < g_stock; ++i)
        cout << g_storage[i];

    cout << op << prod << endl;
}

class Producer: public QThread {
public:
    Producer(char const* name)
        : m_name(name)
        , m_speed(time(NULL))
        , m_prod(time(NULL))
    {
    }

protected:
    void run()
    {
        for (;;)
        {
            g_mutex.lock();

            while (g_stock >= MAX_STOCK)
            {
                cout << m_name << "> FULL!" << endl;
                g_nonFull.wait(&g_mutex);
            }

            char prod = m_prod.bounded('A', 'Z' + 1);
            show(m_name, "<-", prod);
            g_storage[g_stock++] = prod;

            g_nonEmpty.wakeAll();

            g_mutex.unlock();

            msleep(m_speed.bounded(1, 101));
        }
    }

private:
    char const* m_name;
    QRandomGenerator m_speed, m_prod;
};

class Consumer: public QThread {
public:
```

```
69      Consumer(char const* name)
70          : m_name(name)
71          , m_speed(time(NULL))
72      {
73      }
74
75  protected:
76      void run()
77      {
78          for (;;)
79          {
80              g_mutex.lock();
81
82              while (!g_stock)
83              {
84                  cout << m_name << "> EMPTY!" << endl;
85                  g_nonEmpty.wait(&g_mutex);
86              }
87
88              char prod = g_storage[--g_stock];
89              show(m_name, "->", prod);
90
91              g_nonFull.wakeAll();
92
93              g_mutex.unlock();
94
95              msleep(m_speed.bounded(1, 101));
96          }
97      }
98
99  private:
100     char const* m_name;
101     QRandomGenerator m_speed;
102 };
103
104 int main(int argc, char *argv[])
105 {
106     QCoreApplication a(argc, argv);
107
108     Producer p1("PRODUCER-1"), p2("PRODUCER-2");
109     p1.start();
110     p2.start();
111
112     Consumer c1("CONSUMER-1"), c2("CONSUMER-2");
113     c1.start();
114     c2.start();
115
116     return a.exec();
117 }
```

82.3.3 测试验证

运行效果如图所示：

```
PRODUCER-1> <-F
PRODUCER-2> F<-F
CONSUMER-1> F->F
CONSUMER-2> ->F
CONSUMER-2> EMPTY!
CONSUMER-1> EMPTY!
PRODUCER-2> <-A
PRODUCER-1> A<-A
...
PRODUCER-1> ACIRMEVS<-D
PRODUCER-2> ACIRMEVSD<-F
CONSUMER-1> ACIRMEVSD->F
PRODUCER-1> ACIRMEVSD<-T
PRODUCER-2> FULL!
CONSUMER-1> ACIRMEVSD->T
PRODUCER-2> ACIRMEVSD<-F
PRODUCER-1> FULL!
CONSUMER-2> ACIRMEVSD->F
PRODUCER-2> ACIRMEVSD<-V
...
```