带有消息队列和计时器的 C++ std::thread 事件循环

大卫·拉弗尼尔

2020 年 9 月 14 日 警察

评价我: 4.77/5 (38 票)

使用 C++11 线程支持库创建一个带有事件循环、消息队列和计时器的工作线程

std::thread 可用于分离线程，但没有线程安全队列和计时器——大多数操作系统提供的服务。在本文中，我将展示如何使用 C++ 标准库来创建这些"缺失"的功能，并提供许多程序员熟悉的事件处理循环。

下载 StdWorkerThread.zip - 33.9 KB

介绍

事件循环，有时也称为消息循环，是一个等待和分派传入事件的线程。线程阻塞等待请求到达，然后将事件分派给事件处理函数。循环通常使用消息队列来保存传入的消息。每条消息依次出列、解码，然后执行一个动作。事件循环是实现进程间通信的一种方式。

所有操作系统都提供对多线程应用程序的支持。每个操作系统都有独特的函数调用来创建线程、消息队列和计时器。随着 C++11 线程支持库的出现，现在可以创建可移植代码并避免特定于操作系统的函数调用。本文提供了一个简单的示例，说明如何在仅依赖 C++ 标准库的情况下创建线程事件循环、消息队列和计时器服务。任何支持线程库的 C++11 编译器都应该能够编译附加的源代码。

背景

通常，我需要一个线程作为事件循环运行。传入的消息由线程出列，数据根据唯一的消息标识符分派到适当的函数处理程序。能够调用函数的计时器支持对于低速轮询或在预期时间内没有发生某些事情时生成超时非常方便。很多时候，工作线程是在启动时创建的，并且在应用程序终止之前不会被销毁。

实现的一个关键要求是传入的消息必须在同一个线程实例上执行。虽然 say`std::async` 可能会使用池中的临时线程，但此类可确保所有传入消息都使用相同的线程。例如，一个子系统可以用非线程安全的代码来实现。单个`WorkerThread` 实例用于安全地将函数调用分派到子系统中。

乍一看，C++ 线程支持似乎缺少一些关键功能。是的，`std::thread` 可用于分离线程，但没有线程安全队列和计时器——大多数操作系统提供的服务。我将展示如何使用 C++ 标准库来创建这些"缺失"的特性，并提供许多程序员熟悉的事件处理循环。

工作线程

本`WorkerThread` 类封装了所有必要的事件循环机制。一个简单的类接口允许创建线程、将消息发布到事件循环以及最终线程终止。界面如下图所示：

C++ 缩小▲ 复制代码

```
class WorkerThread
{
public:
    /// Constructor
    WorkerThread(const char* threadName);

    /// Destructor
    ~WorkerThread();

    /// Called once to create the worker thread
    /// @return True if thread is created. False otherwise.
    bool CreateThread();

    /// Called once a program exit to exit the worker thread
    void ExitThread();

    /// Get the ID of this thread instance
    /// @return The worker thread ID
    std::thread::id GetThreadId();

    /// Get the ID of the currently executing thread
    /// @return The current thread ID
    static std::thread::id GetCurrentThreadId();

    /// Add a message to the thread queue
    /// @param[in] data - thread specific message information
    void PostMsg(std::shared_ptr<UserData> msg);

private:
    WorkerThread(const WorkerThread&) = delete;
    WorkerThread& operator=(const WorkerThread&) = delete;

    /// Entry point for the worker thread
    void Process();

    /// Entry point for timer thread
    void TimerThread();

    std::unique_ptr<std::thread> m_thread;
    std::queue<std::shared_ptr<ThreadMsg>> m_queue;
    std::mutex m_mutex;
    std::condition_variable m_cv;
    std::atomic<bool> m_timerExit;
    const char* THREAD_NAME;
};
```

首先要注意的是，`std::thread` 它用于创建一个主工作线程。主要的工作线程函数是`Process()`.

C++ 复制代码

```
bool WorkerThread::CreateThread()
{
    if (!m_thread)
        m_thread = new thread(&WorkerThread::Process, this);
    return true;
}
```

事件循环

该`Process()` 事件循环如下所示。线程依赖于`std::queue<ThreadMsg*>` 消息队列。`std::queue` 不是线程安全的，因此对队列的所有访问都必须受互斥锁保护。A`std::condition_variable` 用于挂起线程，直到收到新消息已添加到队列的通知。

C++ 缩小▲ 复制代码

```
void WorkerThread::Process()
{
    m_timerExit = false;
    std::thread timerThread(&WorkerThread::TimerThread, this);

    while (1)
    {
        std::shared_ptr<ThreadMsg> msg;
        {
            // Wait for a message to be added to the queue
            std::unique_lock<std::mutex> lk(m_mutex);
            while (m_queue.empty())
                m_cv.wait(lk);

            if (m_queue.empty())
                continue;

            msg = m_queue.front();
            m_queue.pop();
        }

        switch (msg->id)
        {
            case MSG_POST_USER_DATA:
            {
                ASSERT_TRUE(msg->msg != NULL);

                auto userData = std::static_pointer_cast<UserData>(msg->msg);
                cout << userData->msg.c_str() << " " << userData->year << " on " << THREAD_NAME
<< endl;

                break;
            }

            case MSG_TIMER:
                cout << "Timer expired on " << THREAD_NAME << endl;
                break;

            case MSG_EXIT_THREAD:
            {
                m_timerExit = true;
                timerThread.join();
                return;
            }

            default:
                ASSERT();
        }
    }
}
```

`PostMsg()` `ThreadMsg` 在堆上创建一个新的，将消息添加到队列中，然后使用条件变量通知工作线程。

C++ 复制代码

```
void WorkerThread::PostMsg(std::shared_ptr<UserData> data)
{
    ASSERT_TRUE(m_thread);

    // Create a new ThreadMsg
    std::shared_ptr<ThreadMsg> threadMsg(new ThreadMsg(MSG_POST_USER_DATA, data));

    // Add user data msg to queue and notify worker thread
    std::unique_lock<std::mutex> lk(m_mutex);
    m_queue.push(threadMsg);
    m_cv.notify_one();
}
```

循环将继续处理消息，直到`MSG_EXIT_THREAD` 接收到 并且线程退出。

C++ 复制代码

```
void WorkerThread::ExitThread()
{
    if (!m_thread)
        return;

    // Create a new ThreadMsg
    std::shared_ptr<ThreadMsg> threadMsg(new ThreadMsg(MSG_EXIT_THREAD, 0));

    // Put exit thread message into the queue
    {
        lock_guard<mutex> lock(m_mutex);
        m_queue.push(threadMsg);
        m_cv.notify_one();
    }

    m_thread->join();
    m_thread = nullptr;
}
```

事件循环 (Win32)

下面的代码片段将`std::thread` 上面的事件循环与使用 Windows API 的类似 Win32 版本进行了对比。注意`GetMessage()` API 用于代替`std::queue`. 使用 将消息发布到操作系统消息队列`PostThreadMessage()`。最后，`timerSetEvent()` 用于将`WM_USER_TIMER` 消息放入队列。所有这些服务都由操作系统提供。`std::thread WorkerThread` 此处介绍的实现避免了原始操作系统调用，但实现功能与 Win32 版本相同，仅依赖于 C++ 标准库。

C++ 缩小▲ 复制代码

```
unsigned long WorkerThread::Process(void* parameter)
{
    MSG msg;
    BOOL bRet;

    // Start periodic timer
    MMRESULT timerId = timeSetEvent(250, 10, &WorkerThread::TimerExpired,
                       reinterpret_cast<DWORD>(this), TIME_PERIODIC);

    while ((bRet = GetMessage(&msg, NULL, WM_USER_BEGIN, WM_USER_END)) != 0)
    {
        switch (msg.message)
        {
            case WM_DISPATCH_DELEGATE:
            {
                ASSERT_TRUE(msg.wParam != NULL);

                // Convert the ThreadMsg void* data back to a UserData*
                const UserData* userData = static_cast<const UserData*>(msg.wParam);

                cout << userData->msg.c_str() << " " << userData->year << " on " << THREAD_NAME
<< endl;

                // Delete dynamic data passed through message queue
                delete userData;
                break;
            }

            case WM_USER_TIMER:
                cout << "Timer expired on " << THREAD_NAME << endl;
                break;

            case WM_EXIT_THREAD:
                timeKillEvent(timerId);
                return 0;
```

```
            default:
                ASSERT();
        }
    }
    return 0;
}
```

计时器

使用辅助私有线程将低分辨率定期计时器消息插入队列中。计时器线程是在 内部创建的`Process()`。

C++ 复制代码

```
void WorkerThread::Process()
{
    m_timerExit = false;
    std::thread timerThread(&WorkerThread::TimerThread, this);

...
```

计时器线程的唯一职责是`MSG_TIMER` 每 250 毫秒插入一条消息。在这个实现中，没有针对定时器线程将多个定时器消息注入队列的保护。如果工作线程落后并且不能足够快地为消息队列提供服务，则可能会发生这种情况。根据工作线程、处理负载以及插入计时器消息的速度，可以采用额外的逻辑来防止队列泛滥。

C++ 复制代码

```
void WorkerThread::TimerThread()
{
    while (!m_timerExit)
    {
        // Sleep for 250mS then put a MSG_TIMER into the message queue
        std::this_thread::sleep_for(250ms);

        std::shared_ptr<ThreadMsg> threadMsg (new ThreadMsg(MSG_TIMER, 0));

        // Add timer msg to queue and notify worker thread
        std::unique_lock<std::mutex> lk(m_mutex);
        m_queue.push(threadMsg);
        m_cv.notify_one();
    }
}
```

用法

`main()`下面的函数显示了如何使用`WorkerThread` 该类。创建了两个工作线程并向每个线程发送一条消息。短暂延迟后，两个线程都退出。

C++ 缩小▲ 复制代码

```
// Worker thread instances
WorkerThread workerThread1("WorkerThread1");
WorkerThread workerThread2("WorkerThread2");

int main(void)
{
    // Create worker threads
    workerThread1.CreateThread();
    workerThread2.CreateThread();

    // Create message to send to worker thread 1
    std::shared_ptr<UserData> userData1(new UserData());
    userData1->msg = "Hello world";
```

```
    userData1->year = 2017;

    // Post the message to worker thread 1
    workerThread1.PostMsg(userData1);

    // Create message to send to worker thread 2
    std::shared_ptr<UserData> userData2(new UserData());
    userData2->msg = "Goodbye world";
    userData2->year = 2017;

    // Post the message to worker thread 2
    workerThread2.PostMsg(userData2);

    // Give time for messages processing on worker threads
    this_thread::sleep_for(1s);

    workerThread1.ExitThread();
    workerThread2.ExitThread();

    return 0;
}
```

结论

C++ 线程支持库提供了一种独立于平台的方式来编写多线程应用程序代码，而不依赖于特定于操作系统的 API。`WorkerThread` 此处介绍的类是事件循环的基本实现，但所有基础知识都已准备好进行扩展。

历史

- 5 个月，2017年
  - 初始发行
- 7 个月，2017年
  - 更新文章以提供有关事件循环的说明并将实现与 Win32 事件循环进行对比
- 2020 年 9 月 13 日
  - 较小的现代化更新以简化实施。新的源代码和文章更新。

执照

分享

关于作者

大卫·拉弗尼尔

美国

手表该会员

我做专业软件工程师已经超过 20 年了。不编写代码时，我喜欢与家人共度时光，在南加州露营和骑摩托车。

评论和讨论

添加评论或问题 电子邮件提醒 Search Comments

第一 页上一页 下一页

GetMessage 与 PeekMessage
Scott VS 16-Sep-20 22:13

回复：GetMessage 与 PeekMessage
David Lafreniere 18-Sep-20 0:53

消息已关闭
14-Sep-20 13:15

Compile issue
Sandhya Igwe 11-Sep-20 23:29

Re: Compile issue
David Lafreniere 14-Sep-20 0:40

The example worked
coarist 22-Mar-19 2:35

I have a question on this code.
Member 13985826 8-Nov-18 4:38

Re: I have a question on this code.
陈琦 29-Nov-19 7:53

Several issues, not modern C++ at all
bpeikes 17-Jul-18 11:42

Re: Several issues, not modern C++ at all
David Lafreniere 18-Sep-18 0:30

Re: Several issues, not modern C++ at all
TakeyoshiX 18-Jun-19 16:42

Re: Several issues, not modern C++ at all
陈琦 29-Nov-19 7:58

Re: Several issues, not modern C++ at all
Claudio Viotti 11-May-20 6:22

My vote of 1
bpeikes **17-Jul-18 11:33**

I'm missing something!!!
Member 13895321 **3-Jul-18 0:03**

Re: I'm missing something!!!
Member 13895321 3-Jul-18 0:25

My vote of 1
RomanOk **19-Feb-17 20:46**

Re: My vote of 1
David Lafreniere 28-Feb-17 6:04

I have some exception error
Sun-Mi Kang **9-Feb-17 8:58**

Re: I have some exception error
David Lafreniere 9-Feb-17 19:12

Re: I have some exception error
Sun-Mi Kang 10-Feb-17 10:48

Timer
blastar2 **9-Feb-17 8:57**

Re: Timer
David Lafreniere 9-Feb-17 19:11

Re: Timer
陈琦 29-Nov-19 8:14

Outstanding Once Again!
koothkeeper **9-Feb-17 0:28**

刷新 1 2 下一个 ▷

一般 新闻 建议 问题 错误 答案 笑话 赞美 咆哮 管理员

使用Ctrl+Left/Right 切换消息，Ctrl+Up/Down 切换主题，Ctrl+Shift+Left/Right 切换页面。