

# C++ 服务器案例学习笔记

郑华

2018 年 1 月 25 日



# 目录

<b>第一章 开源-muduo-1.1.2</b>	<b>5</b>
1.1 架构图	5
1.2 参考	5
1.3 类设计	5
1.3.1 Thread	5
1.3.2 MutexLock	8
1.3.3 Condition	8
1.3.4 CountdownLatch	8
1.3.5 BlockingQueue 与 BoundedBlockingQueue	9
1.3.6 ThreadPool	11
1.3.7 Singleton	11
<b>第二章 开源-libevent-1.4</b>	<b>13</b>
2.1 参考	13
<b>第三章 开源-lighttpd-1.4.15</b>	<b>15</b>
3.1 架构图	15
<b>第四章 开源-nginx-0.5.38</b>	<b>17</b>
4.1 架构	17
4.2 参考	17
<b>第五章 游戏服务器 - 戴汉水</b>	<b>19</b>
5.1 经验学习	19
5.1.1 #ifdef .. 编译选择	19
5.1.2 string 和 wstring 相互转换	19
5.1.3 GBK 和 UTF8 的相互变换	19
5.1.4 boost 智能指针	19
5.1.5 游戏服务器种类	19
5.1.6 enum 枚举的使用	19
5.1.7 按模块编码，而不是头文件与 cpp 文件	19
5.2 错误记录	20
5.2.1 错误 204 error LNK2001: 无法解析的外部符号	20
<b>第六章 游戏服务器 2 - 戴汉水</b>	<b>21</b>
6.1 经验	21
6.1.1 设置自己的解决方案目录结构	21
6.1.2 学会程序跟踪，与 Debug	21
6.2 错误记录	21
6.2.1 Error MSB3073 代码 9009	21

6.2.2	Error fatal error LNK1104: cannot open file 'atlsd.lib' . . . . .	21
6.2.3	Project : error PRJ0019: 工具从”正在执行预生成事件...” . . . . .	21
6.2.4	subwcrev 不是内部或外部命令，也不是可运行的程序 . . . . .	21
6.2.5	resource 文件目录如果打不开 . . . . .	21
<b>第七章</b>	<b>游戏解决方案的搭建</b>	<b>23</b>
7.1	DLL 动态库 . . . . .	23
7.2	调用 DLL . . . . .	23
7.3	调用 DLL 中的头文件，但其实不是 . . . . .	23
<b>第八章</b>	<b>Others</b>	<b>25</b>
8.1	线程池 . . . . .	25
8.1.1	核心理念 . . . . .	25
8.1.2	组成部分 . . . . .	25
8.1.3	实现 . . . . .	26

# 第一章 开源-muduo-1.1.2

## 1.1 架构图

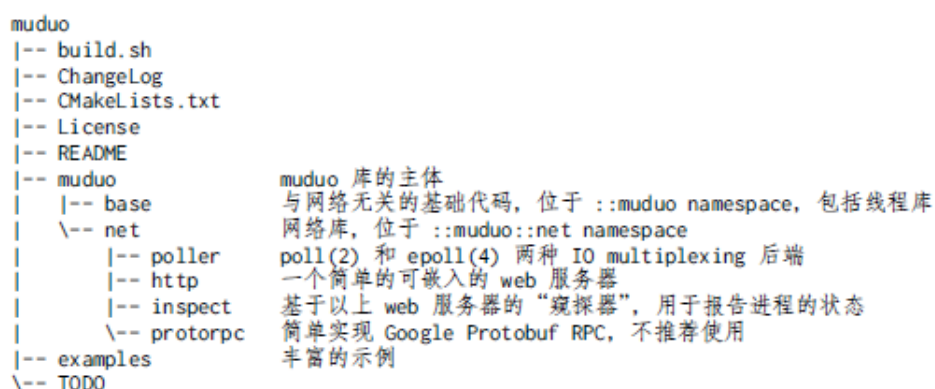


图 1.1: 架构图

## 1.2 参考

代码架构：见 muduo 参考手册，net 文件夹下

博客：<http://www.cppblog.com/Solstice/default.html?page=3>

[https://www.oschina.net/question/28\\_61182](https://www.oschina.net/question/28_61182)

## 1.3 类设计

### 1.3.1 Thread

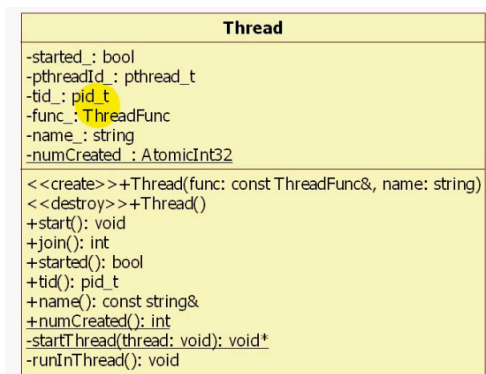


图 1.2: Thread 类图

## Thread 头文件

```
namespace muduo
{
    class Thread : boost::noncopyable
    {
    public:
        typedef boost::function<void ()> ThreadFunc;

        explicit Thread(const ThreadFunc&, const string& name = string());
#ifdef __GXX_EXPERIMENTAL_CXX0X__
        explicit Thread(ThreadFunc&&, const string& name = string());
#endif
        ~Thread();

        void start();
        int join(); // return pthread_join()

        bool started() const { return started_; }

        pid_t tid() const { return *tid_; }
        const string& name() const { return name_; }

        static int numCreated() { return numCreated_.get(); }

    private:
        void setDefaultName();

        bool started_;
        bool joined_;
        pthread_t pthreadId_;
        boost::shared_ptr<pid_t> tid_;
        ThreadFunc func_;
        string name_;

        static AtomicInt32 numCreated_;
    };
}
```

### 要点

- 该线程类使用基于对象的方式实现，即注册回调函数来实现各对象不同的功能。
- 静态方法不能调用非静态，如 this
- 参数类型匹配，例如传递对象指针先得转化为 void\*，进去后然后再转回来。
- pthread\_create 在线程创建以后，就开始运行相关的线程函数
- pthread\_join 如果没有加pthread\_join() 方法，main 线程里面直接就执行起走了，加了之后是等待线程执行了之后才执行的后面的代码。

<http://blog.csdn.net/dinghqalex/article/details/42921931>

回调图如下1.3.1:

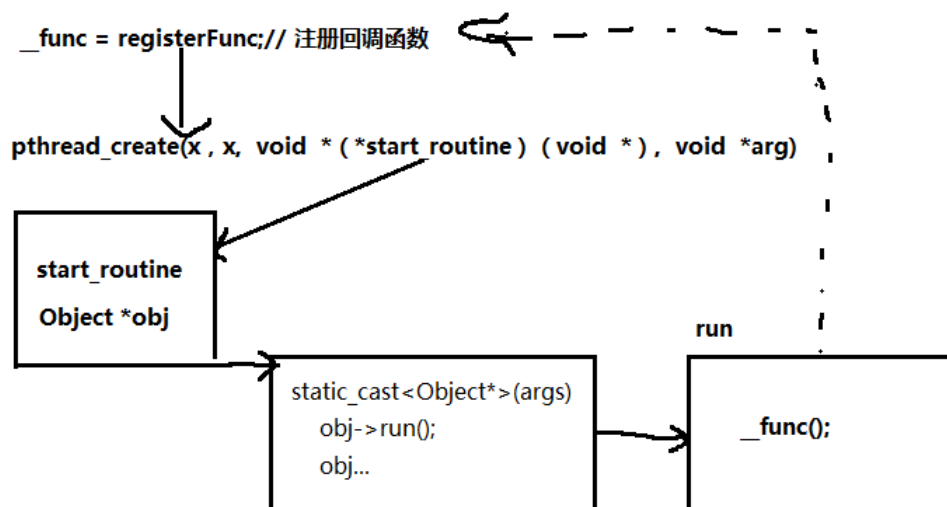


图 1.3: 回调演示

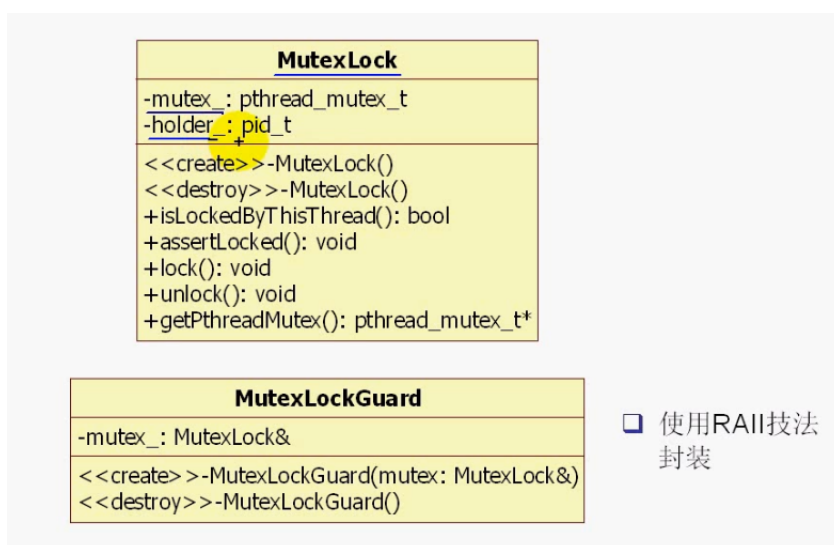


图 1.4: MutexLock 类图

### 1.3.2 MutexLock

#### 要点

- 存在锁竞争
- 

### 1.3.3 Condition

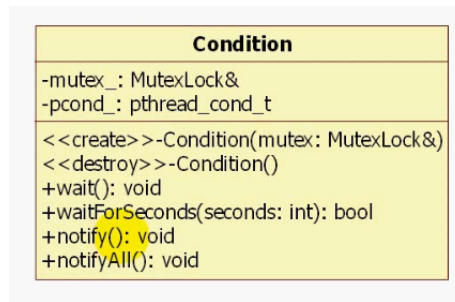


图 1.5: Condition 类图

#### 具体流程

- 锁住 `mutex_lock`
- 等待条件满足

```
while()
{
    mutex_unlock
    等待条件
    mutex_lock
}
```

- 解锁 `mutex_unlock`

#### 要点

- 观察者模式

### 1.3.4 CountdownLatch

可以用于所有子线程等待主线程发起“起跑”

可以用于主线程等待子线程初始化完毕才开始工作。

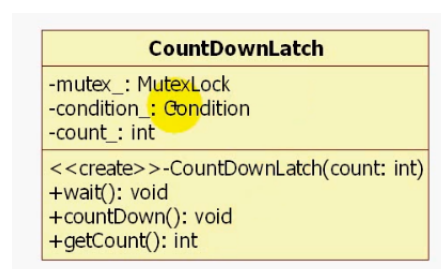


图 1.6: Condition 类图



### 1.3.5 BlockingQueue 与 BoundedBlockingQueue

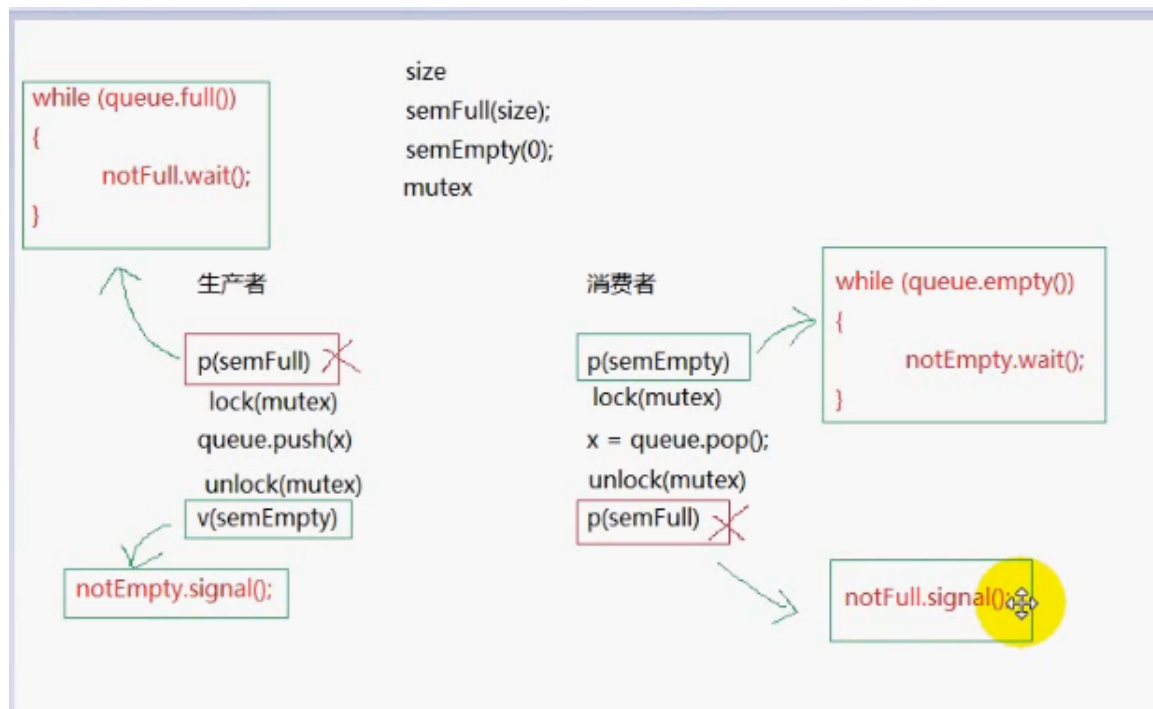


图 1.7: 有界队列

BlockingQueue<T>

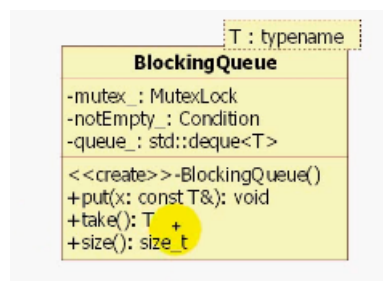


图 1.8: BlockingQueue 类图

BoundedBlockingQueue<T>

实现示例 ->

```

namespace muduo
{
    template<typename T>
    class BlockingQueue : boost::noncopyable
    {
    public:
        BlockingQueue()
        : mutex_(),
          notEmpty_(mutex_),
          queue_()
        {
            // ...
        }
    };
    
```

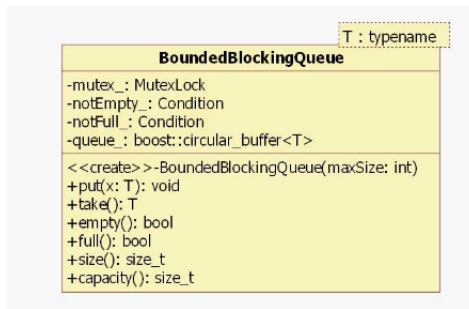


图 1.9: BlockingQueue

```

}

void put(const T& x)
{
    MutexLockGuard lock(mutex_);
    queue_.push_back(x);
    notEmpty_.notify(); // wait morphing saves us
    // http://www.domaigine.com/blog/computing/condvars-signal-with-mutex-locked-or-not/
}

#ifdef __GXX_EXPERIMENTAL_CXX0X__
void put(T&& x)
{
    MutexLockGuard lock(mutex_);
    queue_.push_back(std::move(x));
    notEmpty_.notify();
}
// FIXME: emplace()
#endif

T take()
{
    MutexLockGuard lock(mutex_);
    // always use a while-loop, due to spurious wakeup
    while (queue_.empty())
    {
        notEmpty_.wait();
    }
    assert(!queue_.empty());
#ifdef __GXX_EXPERIMENTAL_CXX0X__
    T front(std::move(queue_.front()));
#else
    T front(queue_.front());
#endif
    queue_.pop_front();
    return front;
}

size_t size() const
{
    MutexLockGuard lock(mutex_);
    return queue_.size();
}

private:
mutable MutexLock mutex_;
  
```

```

        Condition notEmpty_;
        std::deque<T> queue_;
    };
}

```

## 核心要点

- 对于无界队列只需要将 notFull 条件变量去掉即可

## 1.3.6 ThreadPool

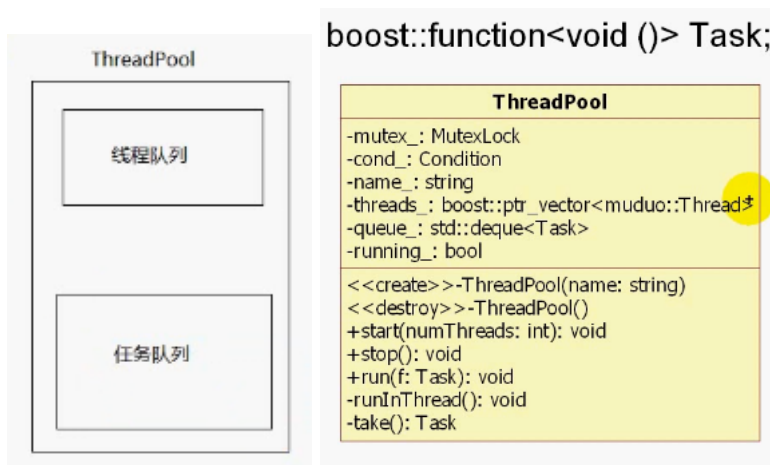


图 1.10: ThreadPool 模型

示例代码 <https://github.com/ctzhenghua/muduo/blob/master/muduo/base/ThreadPool.cc>

## 1.3.7 Singleton

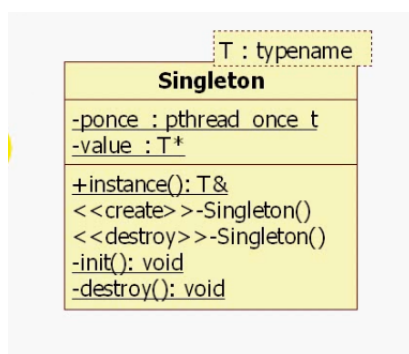


图 1.11: Singleton 模型



## 第二章 开源-libevent-1.4

### 2.1 参考

代码架构：见参考手册，net 文件夹下



## 第三章 开源-lighthttpd-1.4.15

### 3.1 架构图

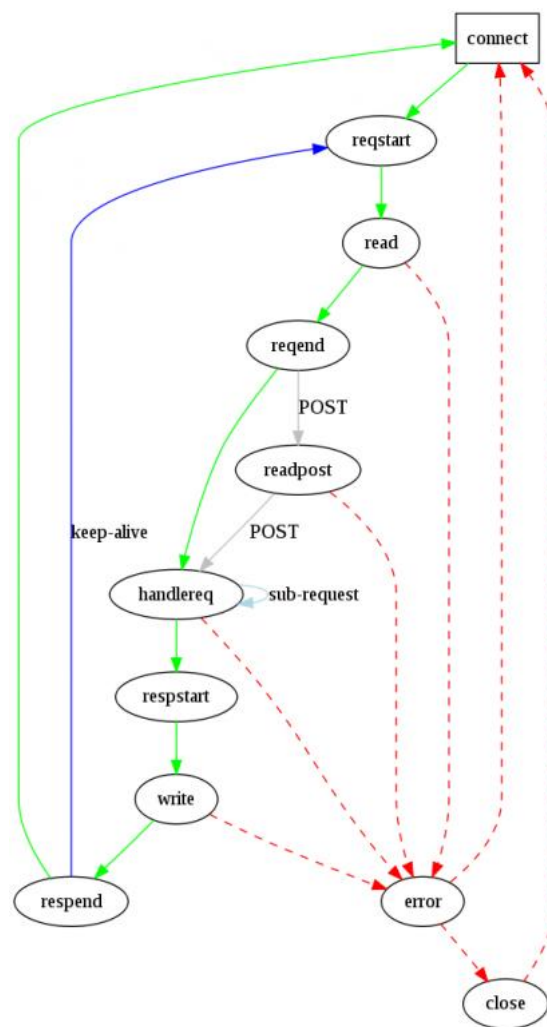


图 3.1: 架构图





## 第四章 开源-nginx-0.5.38

### 4.1 架构

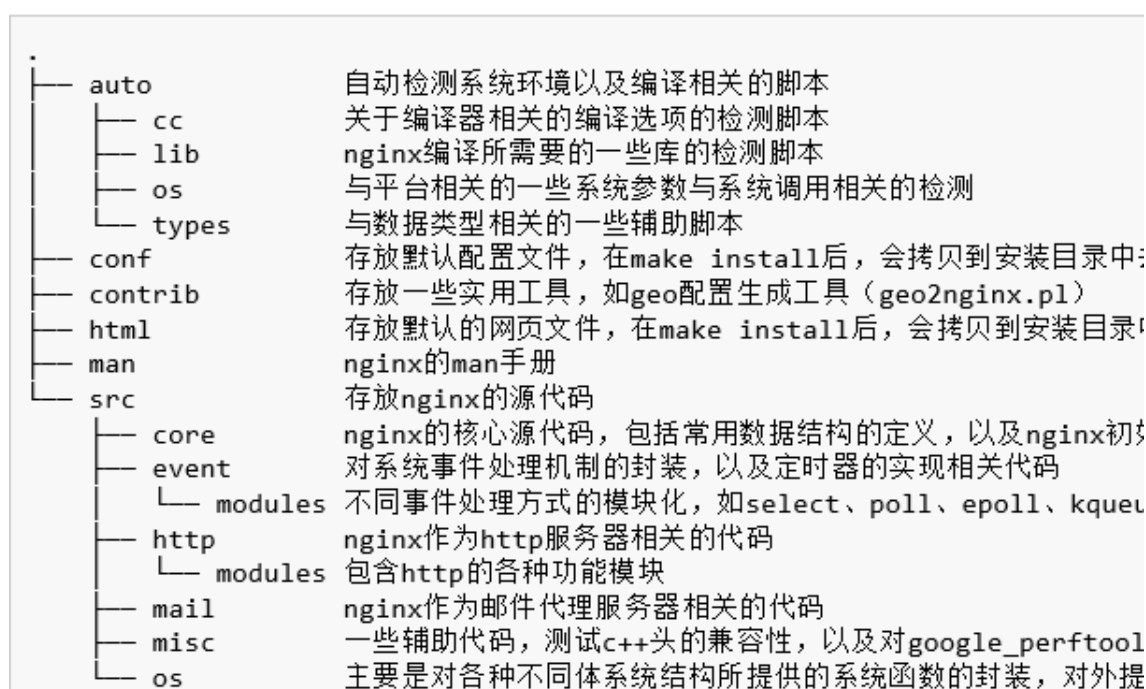


图 4.1: 架构图

### 4.2 参考

代码架构: [http://tengine.taobao.org/book/chapter\\_09.html](http://tengine.taobao.org/book/chapter_09.html)



## 第五章 游戏服务器 - 戴汉水

### 5.1 经验学习

#### 5.1.1 `#ifdef ..` 编译选择

#### 5.1.2 `string` 和 `wstring` 相互转换

#### 5.1.3 GBK 和 UTF8 的相互变换

#### 5.1.4 `boost` 智能指针

#### 5.1.5 游戏服务器种类

数据库服务器，游戏服务器，登陆服务器，battle 服务器

#### 5.1.6 `enum` 枚举的使用

#### 5.1.7 按模块编码，而不是头文件与 `cpp` 文件

有点像 java 包的概念，像包一样的分解程序，分解成 lib, 然后调用，继承

## 5.2 错误记录

### 5.2.1 错误 204 error LNK2001: 无法解析的外部符号

```
"__declspec(dllimport)  
class std::basic_ostream<char,struct std::char_traits<char> >  
& __cdecl std::endl(class std::basic_ostream<char,struct std::char_traits<char> > &)"
```

**解决办法** VS20xx:

项目、属性、链接器、常规、附加库目录: 填写附加依赖库所在目录分号间隔多项

项目、属性、链接器、输入、附加依赖项: 填写附加依赖库的名字.lib 空格或分号间隔多项

## 第六章 游戏服务器 2 - 戴汉水

### 6.1 经验

#### 6.1.1 设置自己的解决方案目录结构

: [http://www.360doc.com/content/13/0422/10/8251840\\_280065154.shtml](http://www.360doc.com/content/13/0422/10/8251840_280065154.shtml)

#### 6.1.2 学会程序跟踪，与 Debug

### 6.2 错误记录

#### 6.2.1 Error MSB3073 代码 9009

:

#### 6.2.2 Error fatal error LNK1104: cannot open file 'atlsd.lib'

:<http://blog.csdn.net/hsluoyc/article/details/46312293>

貌似是 2008 以前版本用的库有个 atlsd.lib，由于 atlsd.lib 在高版本 VS 中也不存在，因此只好用其 Release 版本代替，经测试可以使用。没有安装高版本 VS 的同学可以留言索取 atls.lib。

#### 6.2.3 Project : error PRJ0019: 工具从”正在执行预生成事件...”

配置属性-生成事件-预编译事件或生成后事件-命令行，删除命令行内容即可

#### 6.2.4 subwcrev 不是内部或外部命令，也不是可运行的程序

: svn 命令：该版本 visual studio 未安装 SVN 版本控制器，导致的错误

#### 6.2.5 resource 文件目录如果打不开

请检查是不是不存在.rc 文件，或.rc 文件多了后缀如.rc.in



## 第七章 游戏解决方案的搭建

7.1 DLL 动态库

7.2 调用 DLL

7.3 调用 DLL 中的头文件，但其实不是





## 第八章 Others

### 8.1 线程池

多线程技术主要解决处理器单元内多个线程执行的问题，它可以显著减少处理器单元的闲置时间，增加处理器单元的吞吐能力。

线程池的好处就在于线程复用，一个任务处理完成后，当前线程可以直接处理下一个任务，而不是销毁后再创建，非常适用于连续产生大量并发任务的场合。

<http://wenku.baidu.com/view/f0bd6127ccbff121dd36831a.html>

#### 8.1.1 核心思想

假设一个服务器完成一项任务所需时间为： $T_1$  创建线程时间， $T_2$  在线程中执行任务的时间， $T_3$  销毁线程时间。

如果： $T_1 + T_3$  远大于  $T_2$ ，则可以采用线程池，以提高服务器性能。

线程池技术正是关注如何缩短或调整  $T_1, T_3$  时间的技术，从而提高服务器程序性能的，它把  $T_1, T_3$  分别安排在服务器程序的启动和结束的时间段或者一些空闲的时间段，这样在服务器程序处理客户请求时，不会有  $T_1, T_3$  的开销了。

除此之外，线程池能够减少创建的线程个数。通常线程池所允许的并发线程是有上界的，如果同时需要并发的线程数超过上界，那么一部分线程将会等待。而传统方案中，如果同时请求数目为 2000，那么最坏情况下，系统可能需要产生 2000 个线程。尽管这不是一个很大的数目，但是也有部分机器可能达不到这种要求。

因此线程池的出现正是着眼于减少线程池本身带来的开销。线程池采用预创建的技术，在应用程序启动之后，将立即创建一定数量的线程 ( $N_1$ )，放入空闲队列中。这些线程都是处于阻塞 (Suspended) 状态，不消耗 CPU，但占用较小的内存空间。当任务到来后，缓冲池选择一个空闲线程，把任务传入此线程中运行。当  $N_1$  个线程都在处理任务后，缓冲池自动创建一定数量的新线程，用于处理更多的任务。在任务执行完毕后线程也不退出，而是继续保持在池中等待下一次的任务。当系统比较空闲时，大部分线程都一直处于暂停状态，线程池自动销毁一部分线程，回收系统资源。

基于这种预创建技术，线程池将线程创建和销毁本身所带来的开销分摊到了各个具体的任务上，执行次数越多，每个任务所分担到的线程本身开销则越小，不过我们另外可能还需要考虑进去线程之间同步所带来的开销。

#### 8.1.2 组成部分

1. 线程池管理器 **ThreadPool** 用于创建并管理线程池，包括创建线程池，销毁线程池，添加新任务
2. 工作线程 **PoolWorker** 线程池中线程，在没有任务时处于等待状态，可以循环的执行任务
3. 任务接口 **Task** 每个任务必须实现的接口，以供工作线程调度任务的执行，它主要规定了任务的入口，任务执行完后的收尾工作，任务的执行状态等
4. 任务队列 **taskQueue** 用于存放没有处理的任务。提供一种缓冲机制。

### 8.1.3 实现

**C++** <http://wenku.baidu.com/view/a4cc1093daef5ef7ba0d3c3a.html>

**Boost** [http://www.oschina.net/code/snippet\\_170948\\_18231](http://www.oschina.net/code/snippet_170948_18231)