易栈服务端

项目环境的搭建

配置系统环境

默认的 Ubuntu 是使用官方的源

```
ada@ubuntu:~$ sudo apt update
[sudo] password for ada:
Get:1 http://security.ubuntu.com/ubuntu bionic-security InRelease [88.7 kB]
Hit:2 http://us.archive.ubuntu.com/ubuntu bionic InRelease
Get:3 http://us.archive.ubuntu.com/ubuntu bionic-updates InRelease [88.7 kB]
Get:4 http://security.ubuntu.com/ubuntu bionic-security/main amd64 DEP-11 Metadata [49.0 kB]
Get:5 http://us.archive.ubuntu.com/ubuntu bionic-backports InRelease [74.6 kB]
Get:6 http://security.ubuntu.com/ubuntu bionic-security/universe amd64 DEP-11 Metadata [59.7 kB]
Get:7 http://security.ubuntu.com/ubuntu bionic-security/multiverse amd64 DEP-11 Metadata [2,464 B]
Get:8 http://us.archive.ubuntu.com/ubuntu bionic-updates/main amd64 DEP-11 Metadata [295 kB]
Get:9 http://us.archive.ubuntu.com/ubuntu bionic-updates/universe amd64 DEP-11 Metadata [288 kB]
Get:10 http://us.archive.ubuntu.com/ubuntu bionic-updates/universe DEP-11 64x64 Icons [485 kB]
Get:11 http://us.archive.ubuntu.com/ubuntu bionic-updates/multiverse amd64 DEP-11 Metadata [2,468 B]
Get:12 http://us.archive.ubuntu.com/ubuntu bionic-backports/universe amd64 DEP-11 Metadata [9,284 B]
```

更新比较慢，非常卡，而且容易失败

修改更新源到清华源

https://mirrors.tuna.tsinghua.edu.cn/

远程连接到 Ubuntu 系统

输入 sudo cp /etc/apt/sources.list /etc/apt/sources.list.bak

进行源的备份

输入 sudo vim /etc/apt/sources.list

准备编辑源，如下图

```
ada@ubuntu:~$ sudo cp /etc/apt/sources.list /etc/apt/sources.list.bak
ada@ubuntu:~$ sudo vim /etc/apt/sources.list
```

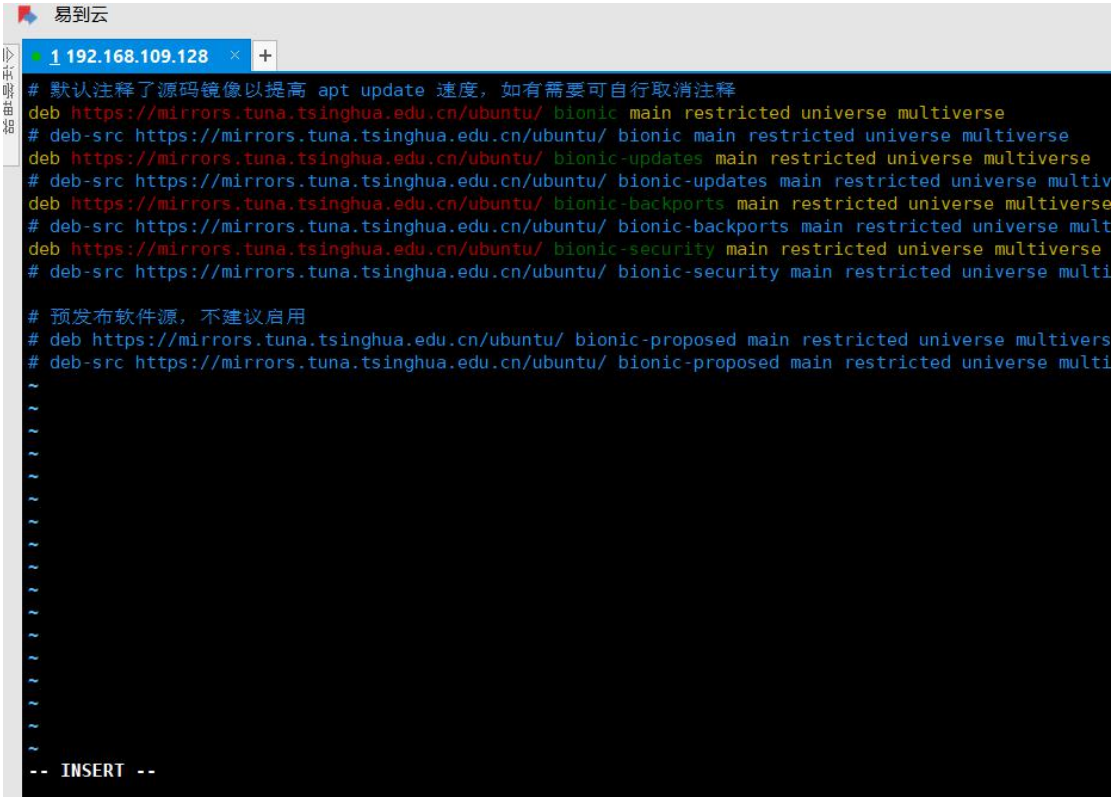
下面内容全部删除（输入 dd 删除一行，按住不放，不停的删除一行）

1 192.168.109.128

```
#deb cdrom:[Ubuntu 18.04.3 LTS _Bionic Beaver_ - Release amd64 (20190805)]/ bionic main restricted

# See http://help.ubuntu.com/community/UpgradeNotes for how to upgrade to
# newer versions of the distribution.
deb http://us.archive.ubuntu.com/ubuntu/ bionic main restricted
# deb-src http://us.archive.ubuntu.com/ubuntu/ bionic main restricted

## Major bug fix updates produced after the final release of the
## distribution.
deb http://us.archive.ubuntu.com/ubuntu/ bionic-updates main restricted
# deb-src http://us.archive.ubuntu.com/ubuntu/ bionic-updates main restricted

## N.B. software from this repository is ENTIRELY UNSUPPORTED by the Ubuntu
## team. Also, please note that software in universe WILL NOT receive any
## review or updates from the Ubuntu security team.
deb http://us.archive.ubuntu.com/ubuntu/ bionic universe
# deb-src http://us.archive.ubuntu.com/ubuntu/ bionic universe
deb http://us.archive.ubuntu.com/ubuntu/ bionic-updates universe
# deb-src http://us.archive.ubuntu.com/ubuntu/ bionic-updates universe

## N.B. software from this repository is ENTIRELY UNSUPPORTED by the Ubuntu
## team, and may not be under a free licence. Please satisfy yourself as to
## your rights to use the software. Also, please note that software in
## multiverse WILL NOT receive any review or updates from the Ubuntu
## security team.
deb http://us.archive.ubuntu.com/ubuntu/ bionic multiverse
# deb-src http://us.archive.ubuntu.com/ubuntu/ bionic multiverse
deb http://us.archive.ubuntu.com/ubuntu/ bionic-updates multiverse
# deb-src http://us.archive.ubuntu.com/ubuntu/ bionic-updates multiverse
```

然后输入 a 进入插入模式

把之前的内容复制后，可以直接粘贴过来



然后按一下 ESC

再输入:wq 回车，即可保存退出

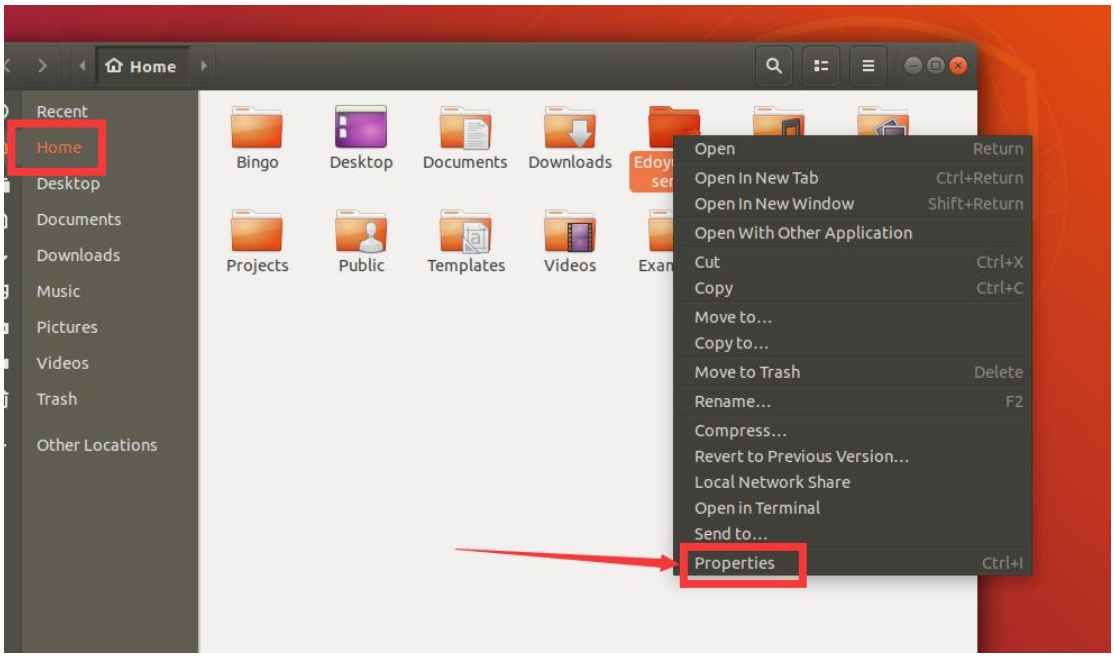
再次更新的时候

```
ada@ubuntu:~$ sudo apt-get update
Get:1 https://mirrors.tuna.tsinghua.edu.cn/ubuntu bionic InRelease [242 kB]
Get:2 https://mirrors.tuna.tsinghua.edu.cn/ubuntu bionic-updates InRelease [88.7 kB]
Get:3 https://mirrors.tuna.tsinghua.edu.cn/ubuntu bionic-backports InRelease [74.6 kB]
Get:4 https://mirrors.tuna.tsinghua.edu.cn/ubuntu bionic-security InRelease [88.7 kB]
Get:5 https://mirrors.tuna.tsinghua.edu.cn/ubuntu bionic/main amd64 Packages [1,019 kB]
Get:6 https://mirrors.tuna.tsinghua.edu.cn/ubuntu bionic/main i386 Packages [1,007 kB]
Get:7 https://mirrors.tuna.tsinghua.edu.cn/ubuntu bionic/main Translation-en [516 kB]
Get:8 https://mirrors.tuna.tsinghua.edu.cn/ubuntu bionic/main amd64 DEP-11 Metadata [477 kB]
Get:9 https://mirrors.tuna.tsinghua.edu.cn/ubuntu bionic/main DEP-11 48x48 Icons [118 kB]
```

可以看到已经是清华的源了

局域网共享文件夹

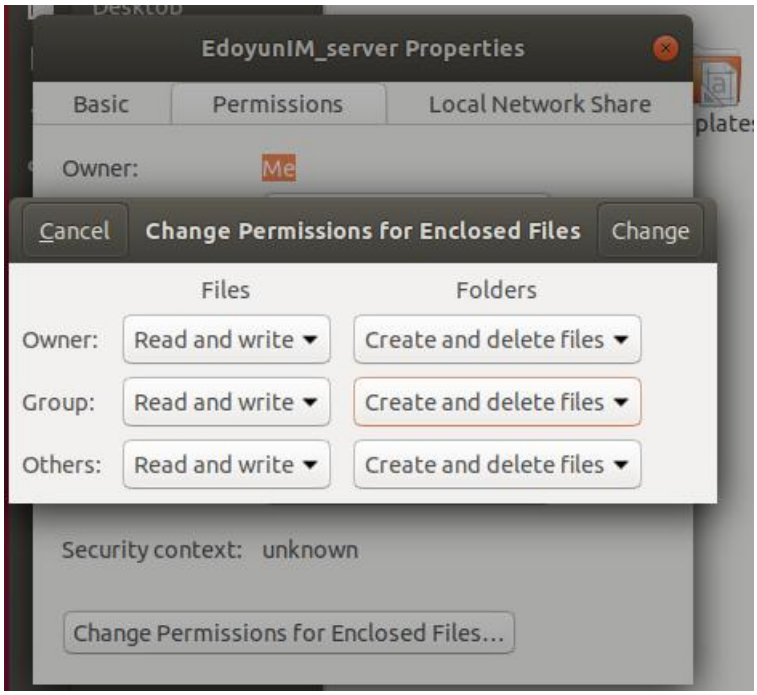
在 home 下新建一个文件夹 EdoyunIM_server，右键单击选择属性（Properties）



可以看到下面的内容

分别设置 Permissions（权限）和 Local Network Share


EdoyunIM_server Properties
Basic
Permissions
Local Network Share
Owner:
Me
Access:
Create and delete files
Group:
ada
Access:
Create and delete files
Others
Access:
Create and delete files
Security context: unknown
Change Permissions for Enclosed Files…

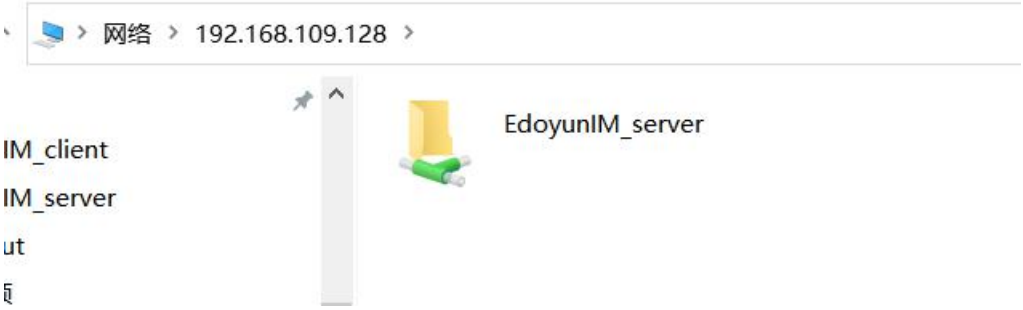

EdoyunIM_server Properties
Basic
Permissions
Local Network Share
Owner:
Me
Cancel
Change Permissions for Enclosed Files
Change
Files
Folders
Owner:
Read and write
Create and delete files
Group:
Read and write
Create and delete files
Others:
Read and write
Create and delete files
Security context: unknown
Change Permissions for Enclosed Files…

一定要开启客户访问，否则就需要创建 ada 组的新账号，使用密码访问

可能会需要安装一些服务，选择安装即可

在 Windows 下打开试一试

然后切换到这个目录



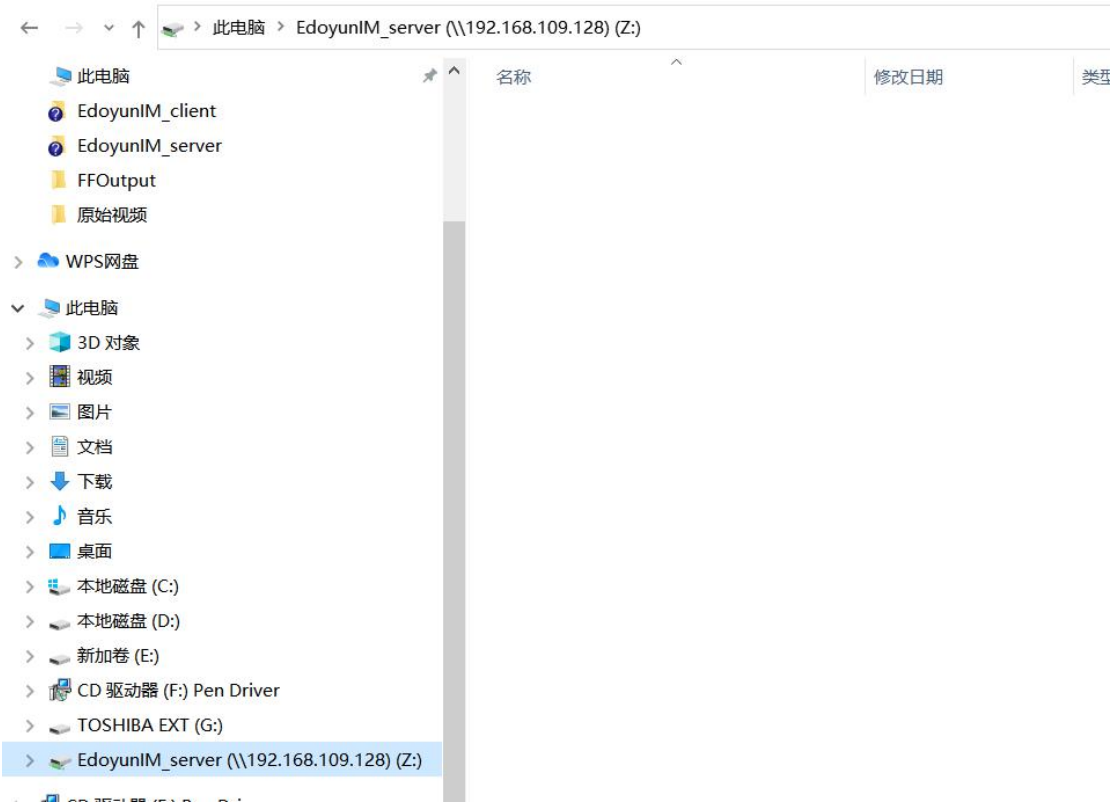
右键单击文件夹

找到 映射网络驱动器

选择 Z 或者其他你喜欢的驱动器

然后在我的电脑即可看到这个驱动器

打开后，即可看到一个空的文件夹

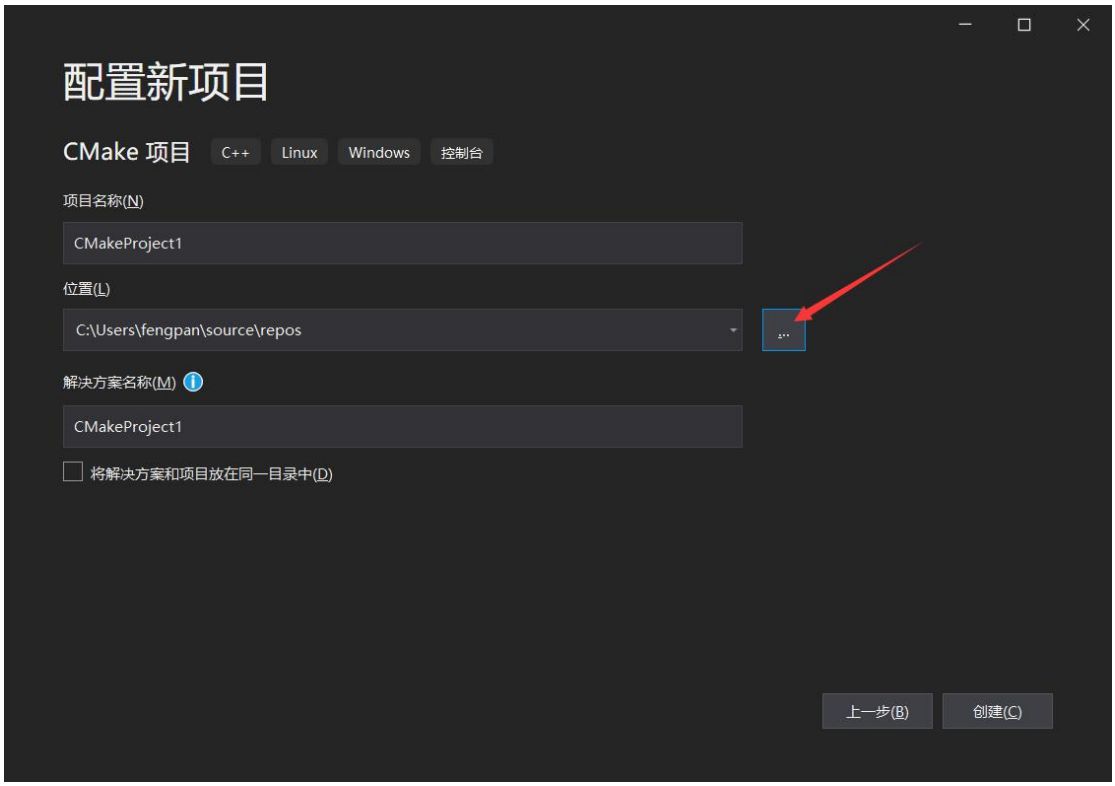


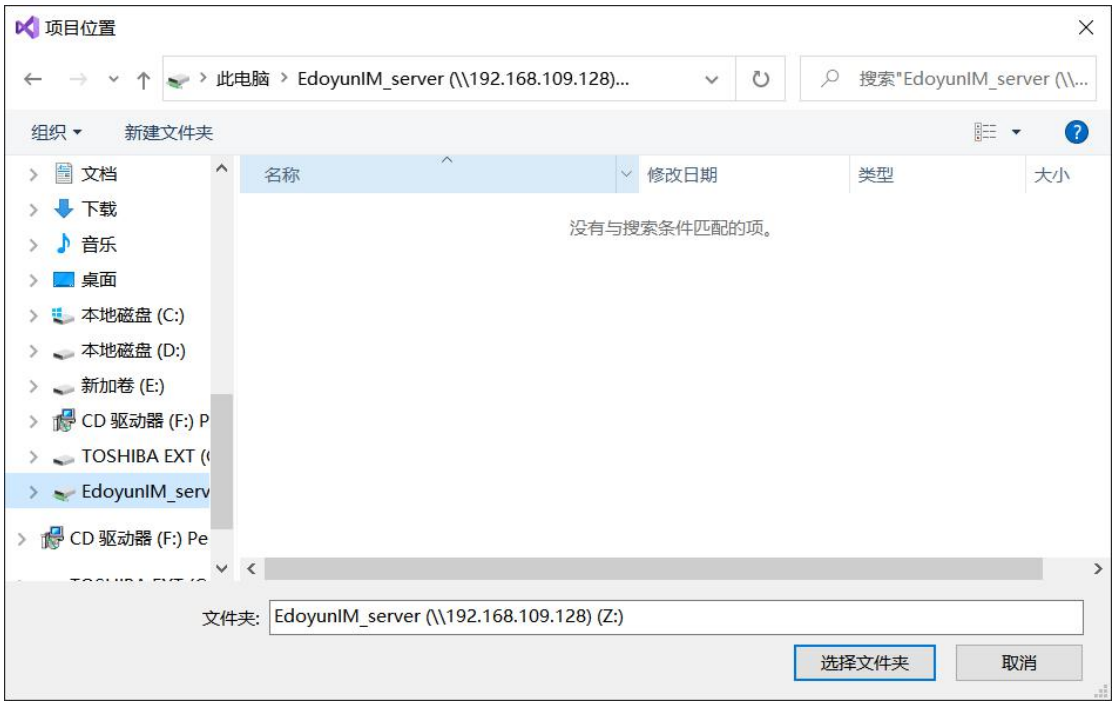
后续的项目，我们就在这里创建

在 Windows 编辑，在 Linux 里面编译

建立 cmake 项目

选择项目位置到网络共享文件夹





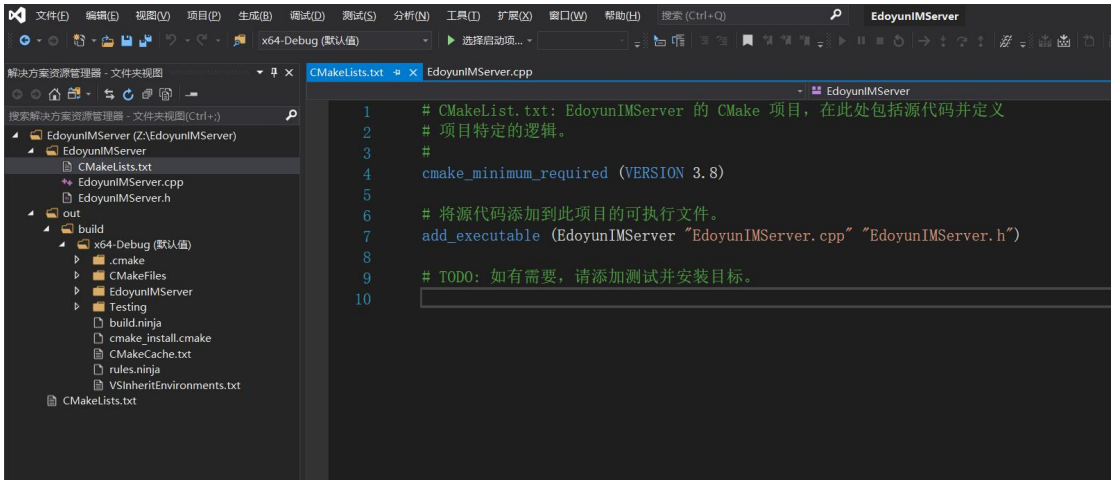
修改项目名为 EdoyunIMServer

如果提出警告，说权限不足，无法写入或者修改一些文件

则可以在命令行输入如下命令：

sudo chmod -R 777 /home/ada/EdoyunIMServer/

```
ada@ubuntu:~/EdoyunIM_server$ sudo chmod -R 777 EdoyunIMServer/
[sudo] password for ada:
ada@ubuntu:~/EdoyunIM_server$
```



在 out/build/建立一个 Ubuntu 文件夹

```
ada@ubuntu:~/EdoyunIM_server/EdoyunIMServer$ ls -l
total 12
-rwxrwxrwx 1 nobody nogroup  234 Jan 21 11:38 CMakeLists.txt
drwxrwxrwx 2 nobody nogroup 4096 Jan 21 11:38 EdoyunIMServer
drwxrwxrwx 3 nobody nogroup 4096 Jan 21 11:38 out
ada@ubuntu:~/EdoyunIM_server/EdoyunIMServer$ cd out
ada@ubuntu:~/EdoyunIM_server/EdoyunIMServer/out$ ls -l
total 4
drwxrwxrwx 3 nobody nogroup 4096 Jan 21 11:38 build
ada@ubuntu:~/EdoyunIM_server/EdoyunIMServer/out$ cd build/
ada@ubuntu:~/EdoyunIM_server/EdoyunIMServer/out/build$ ls -l
total 4
drwxrwxrwx 6 nobody nogroup 4096 Jan 21 11:42 'x64-Debug (默认值)'
ada@ubuntu:~/EdoyunIM_server/EdoyunIMServer/out/build$ mkdir Ubuntu
ada@ubuntu:~/EdoyunIM_server/EdoyunIMServer/out/build$ chmod 777 Ubuntu/
ada@ubuntu:~/EdoyunIM_server/EdoyunIMServer/out/build$ ls -l
total 8
drwxrwxrwx 2 ada    ada     4096 Jan 21 11:47  Ubuntu
drwxrwxrwx 6 nobody nogroup 4096 Jan 21 11:42 'x64-Debug (默认值)'
ada@ubuntu:~/EdoyunIM_server/EdoyunIMServer/out/build$ cd Ubuntu/
ada@ubuntu:~/EdoyunIM_server/EdoyunIMServer/out/build/Ubuntu$
```

然后执行 cmake

```
ada@ubuntu:~/EdoyunIM_server/EdoyunIMServer/out/build/Ubuntu$ cmake ~/EdoyunIM_server/EdoyunIMServer/
-- The C compiler identification is GNU 7.5.0
-- The CXX compiler identification is GNU 7.5.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /home/ada/EdoyunIM_server/EdoyunIMServer/out/build/Ubuntu
ada@ubuntu:~/EdoyunIM_server/EdoyunIMServer/out/build/Ubuntu$
```

然后执行 make

```
ada@ubuntu:~/EdoyunIM_server/EdoyunIMServer/out/build/Ubuntu$ make
Scanning dependencies of target EdoyunIMServer
[ 50%] Building CXX object EdoyunIMServer/CMakeFiles/EdoyunIMServer.dir/EdoyunIMServer.cpp.o
[100%] Linking CXX executable EdoyunIMServer
[100%] Built target EdoyunIMServer
ada@ubuntu:~/EdoyunIM_server/EdoyunIMServer/out/build/Ubuntu$ ls -l
total 36
-rw-rw-r-- 1 ada ada 12714 Jan 21 11:48 CMakeCache.txt
drwxrwxr-x 4 ada ada  4096 Jan 21 11:49 CMakeFiles
-rw-rw-r-- 1 ada ada  1746 Jan 21 11:48 cmake_install.cmake
drwxrwxr-x 3 ada ada  4096 Jan 21 11:49 EdoyunIMServer
-rw-rw-r-- 1 ada ada  4343 Jan 21 11:48 Makefile
ada@ubuntu:~/EdoyunIM_server/EdoyunIMServer/out/build/Ubuntu$ ls -l EdoyunIMServer/
total 28
drwxrwxr-x 3 ada ada 4096 Jan 21 11:48 CMakeFiles
-rw-rw-r-- 1 ada ada 1145 Jan 21 11:48 cmake_install.cmake
-rwxrwxr-x 1 ada ada 8936 Jan 21 11:49 EdoyunIMServer
-rw-rw-r-- 1 ada ada 6230 Jan 21 11:48 Makefile
ada@ubuntu:~/EdoyunIM_server/EdoyunIMServer/out/build/Ubuntu$
```

可以在 EdoyunIMServer 目录下面看到可执行程序

```
ada@ubuntu:~/EdoyunIM_server/EdoyunIMServer/out/build/Ubuntu$ EdoyunIMServer/EdoyunIMServer
Hello CMake.
ada@ubuntu:~/EdoyunIM_server/EdoyunIMServer/out/build/Ubuntu$
```

执行一下，即可看到项目内容
到此，项目基本的环境配置就完成了

CMake 入门

```
# CMakeList.txt: EdoyumIMServer 的 CMake 项目，在此处包括源代码并定义
# 项目特定的逻辑。
# //
cmake_minimum_required (VERSION 3.8)

project(EdoyunServer)

#这个宏，会影响编译参数
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++0x -g -Wall -O0 -Wno-unused-variable
-pthread")

#设置库目录
link_directories(# 针对 64 位的
    ${PROJECT_SOURCE_DIR}/lib
    /usr/lib64/mysql/
)

find_library(MYSQL_LIB libmysqlclient.so /usr/lib64/mysql/)
if (NOT MYSQL_LIB)#NOT 只能用大写
    message(FATAL_ERROR "not found mysqlclient.so at lib64")
endif(NOT MYSQL_LIB)#必须进行匹配，否则会报错

set(chat_srcs
```

```
"EdoyumIMServer.cpp"
"EdoyumIMServer.h")

# 将源代码添加到此项目的可执行文件。
add_executable (imchatserver ${chat_srcs})
#只是确认库存在或者设置库目录，是没有用的，需要链接进来才能起效果
target_link_libraries(imchatserver mysqlclient)

# 将源代码添加到此项目的可执行文件。
add_executable (imfileserver ${chat_srcs})
#只是确认库存在或者设置库目录，是没有用的，需要链接进来才能起效果
target_link_libraries(imfileserver)
# TODO: 如有需要，请添加测试并安装目标。
```

一个服务器的开始

后台运行

如何开启一个后台运行程序

- 参数处理

getopt 函数

- 信号处理

signal 函数

- 子进程

fork 函数

- 后台运行

setsid 使得前台无法影响后台

重置 stdin stdout stderr 到/dev/null 使得后台无法影响前台

- 整体流程
  1. 处理信号
  2. 分析参数
  3. 守护模式
  4. 设置日志
  5. 主程序
  6. 结束退出

Muduo（木铎）库介绍

木铎的含义：文事奋木铎，武事奋金铎

源码下载地址：
https://github.com/chenshuo/muduo
下载木铎库
git clone https://github.com/chenshuo/muduo.git
如果 Ubuntu 提示没有 git 命令
则使用
sudo apt install git
命令来安装
如果网速不好也可以通过
https://github.com/chenshuo/muduo/archive/master.zip
直接下载源码
或者将 https://github.com/chenshuo/muduo.git 导入到码云上再下载

文档结构

```
.
├── base
│   ├── AsyncLogging.h*
```

```
|   ├── Atomic.h
|   ├── BlockingQueue.h
|   ├── BoundedBlockingQueue.h
|   ├── BUILD.bazel
|   ├── Condition.h
|   ├── copyable.h
|   ├── CountDownLatch.h
|   ├── CurrentThread.h
|   ├── Date.h
|   ├── Exception.h
|   ├── FileUtil.h
|   ├── GzipFile.h
|   ├── LogFile.h*
|   ├── Logging.h*
|   ├── LogStream.h*
|   ├── Mutex.h
|   ├── noncopyable.h
|   ├── ProcessInfo.h
|   ├── Singleton.h
|   ├── StringPiece.h
|   ├── tests
|   ├── Thread.h
|   ├── ThreadLocal.h
|   ├── ThreadLocalSingleton.h
|   ├── ThreadPool.h
|   ├── Timestamp.h
|   ├── TimeZone.h
|   ├── Types.h
|   └── WeakCallback.h
└── net
    ├── Acceptor.h
    ├── boilerplate.h
    ├── Buffer.h
    ├── BUILD.bazel
    ├── Callbacks.h
    ├── Channel.h
    ├── CMakeLists.txt
    ├── Connector.h
    ├── Endian.h
    ├── EventLoop.h
    ├── EventLoopThread.h
    ├── EventLoopThreadPool.h
    ├── http
    |   ├── BUILD.bazel
```

```
│   ├── CMakeLists.txt
│   ├── HttpContext.h
│   ├── HttpRequest.h
│   ├── HttpResponse.h
│   ├── HttpServer.h
│   └── tests
├── InetAddress.h
├── inspect
│   ├── Inspector.h
│   ├── PerformanceInspector.h
│   ├── ProcessInspector.h
│   ├── SystemInspector.h
│   └── tests
├── poller
│   ├── EPollPoller.h
│   └── PollPoller.h
├── Poller.h
├── protobuf
│   ├── BufferStream.h
│   └── ProtobufCodecLite.h
├── protorpc
│   ├── google-inl.h
│   ├── README
│   ├── RpcChannel.h
│   ├── RpcCodec.h
│   ├── rpc.proto
│   ├── RpcServer.h
│   └── rpcservice.proto
├── Socket.h
├── SocketsOps.h
├── TcpClient.h
├── TcpConnection.h
├── TcpServer.h
├── tests
├── Timer.h
├── TimerId.h
├── TimerQueue.h
└── ZlibStream.h
```

muduo 库的编译：

安装 boost 库

sudo apt install g++ cmake make libboost-dev

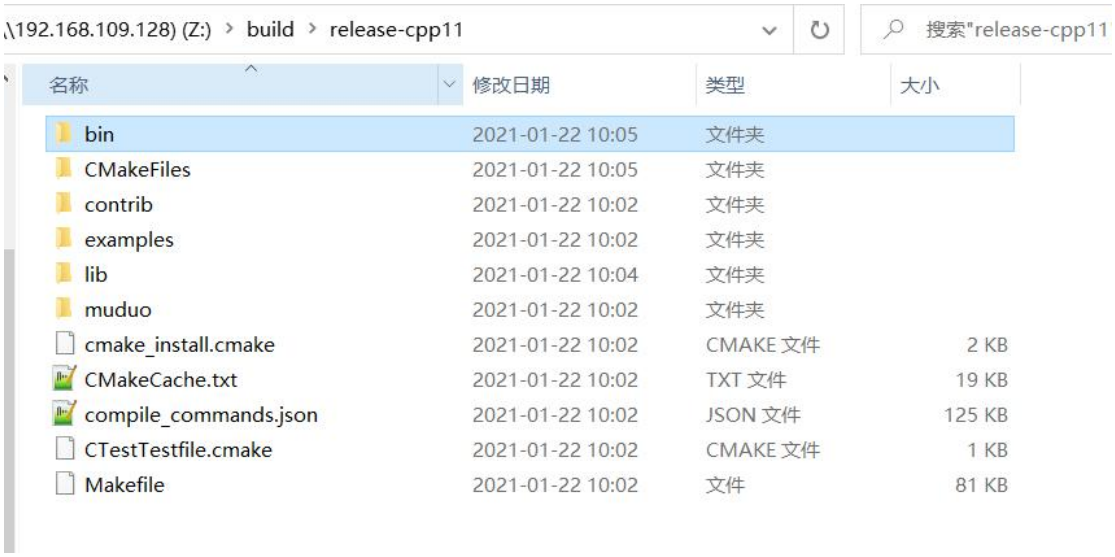
```
$ sudo apt install g++ cmake make libboost-dev
```

然后运行./build.sh

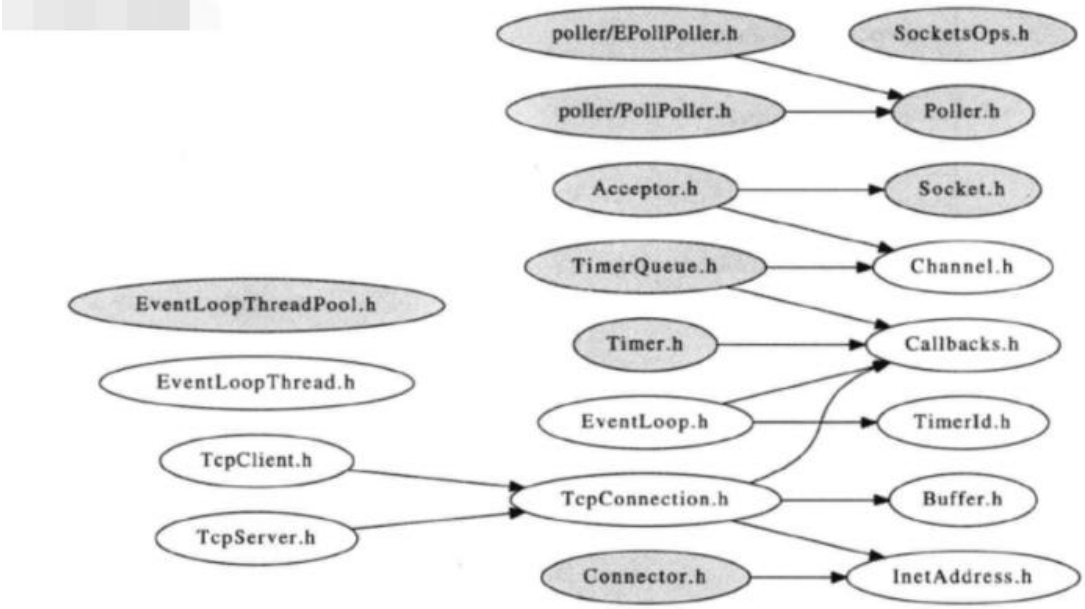
```
[ 96%] Built target twisted_finger06
Scanning dependencies of target wordcount_receiver
[ 96%] Building CXX object examples/wordcount/CMakeFiles/wordcou
[ 96%] Linking CXX executable ../../bin/wordcount_receiver
[ 96%] Built target wordcount_receiver
Scanning dependencies of target wordcount_hasher
[ 97%] Building CXX object examples/wordcount/CMakeFiles/wordcou
[ 97%] Linking CXX executable ../../bin/wordcount_hasher
[ 97%] Built target wordcount_hasher
Scanning dependencies of target zeromq_remote_lat
[ 97%] Building CXX object examples/zeromq/CMakeFiles/zeromq_rem
[ 98%] Linking CXX executable ../../bin/zeromq_remote_lat
[ 98%] Built target zeromq_remote_lat
Scanning dependencies of target zeromq_local_lat
[100%] Building CXX object examples/zeromq/CMakeFiles/zeromq_loc
[100%] Linking CXX executable ../../bin/zeromq_local_lat
[100%] Built target zeromq_local_lat
```

编译完成后，在 build 目录下

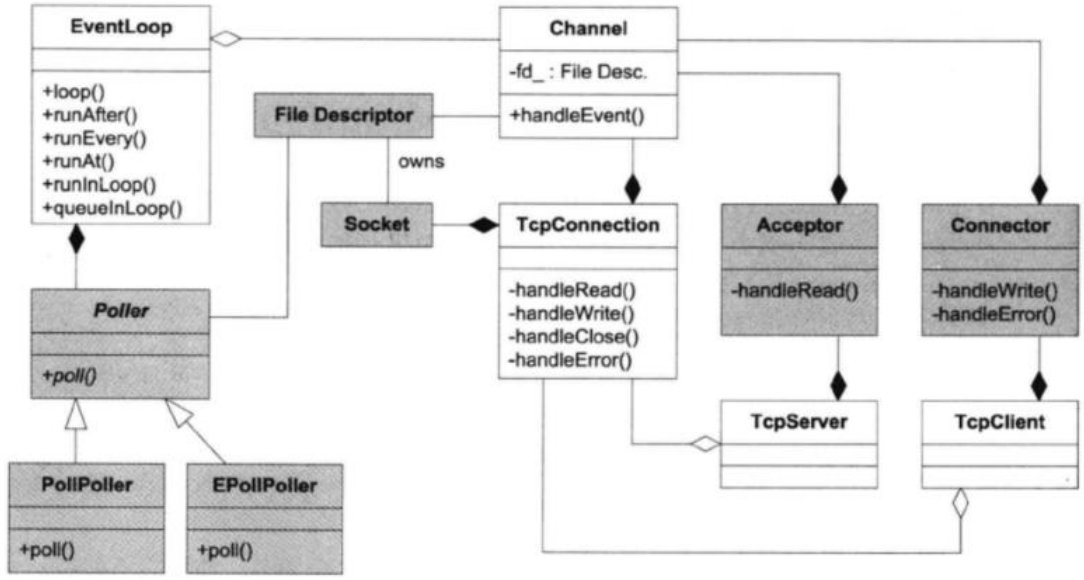
可以找到生成的文件



头文件的包含结构



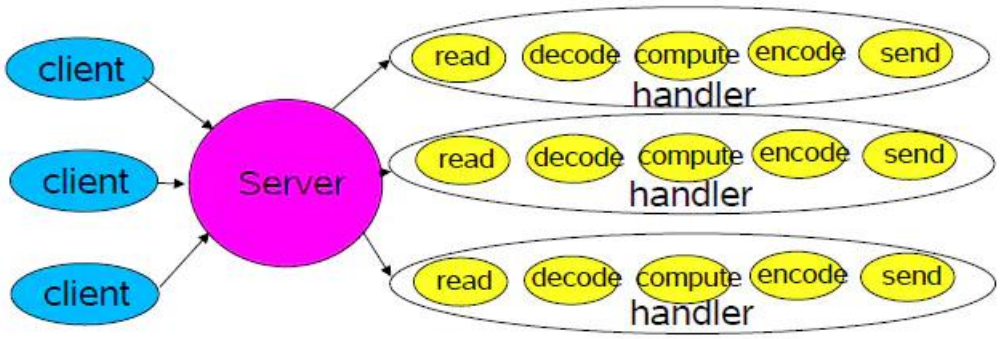
简化的类图



整个模块采用 reactor 模式

Reactor 模式介绍

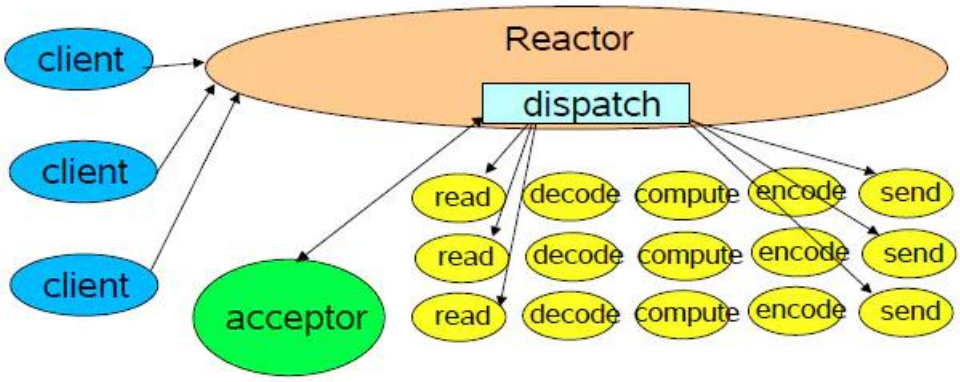
起源
一般我们做服务器，常常是基于线程架构的
也就是 thread-based architecture
这种模式可以用下图来描述：



这种方式适合初中级程序员
优点在于结构清晰，逻辑明了，可读性强
但是缺点也是非常明显的：
每个客户端的请求，都需要一个线程来处理
如果有一万个用户，就需要一万个线程来处理
有十万个用户，就需要开十万个线程
暂且不说十万个，单单一万个线程，仅仅是切换这些线程
基本 CPU 就废了
处了这个问题，还有另外一个问题，就是无法控制算力
比如一百个线程，可能某一刻，一百个线程都在闲着，CPU 闲到发慌
也可能下一刻所有的线程都需要忙起来，CPU 瞬间原地爆炸
而且这种情况完全不受服务器控制
也难以预测，完全由用户的行为来主导

基于上面这种方式的问题
我们有了**事件驱动架构**（event-driven architecture）
这种架构把要使用 CPU 的内容定义为一个事件
比如网络编程中的客户端接入、数据读取、数据发送、连接关闭
当有客户端接入的事件发生的时候
我们就安排线程来处理接入
处理完了，就将该事件移出线程，等待其他事件的发生
这种方式有很多好处
比如我们可以控制 CPU 的利用率
还是上面那个例子，一百个用户连上来
如果某一刻一百个人都在做请求
这个时候我们可以指定几个线程来处理这些请求
没有处理到的用户会在一个事件队列中等候处理
这样不会出现一百个用户同时被处理，但是每个都无法很好的处理

基于事件驱动架构的模式就是 Reactor 模式
这种模式的典型情况如下图



用户来了，reactor（反应堆）会将事件分发给接收、读取、解码、计算、编码、发送各个处理模块
这些模块没有单独的执行权限
全部靠 reactor 来分派线程，进而执行内容
木铎库就是采用这种方式来处理的
这种方式，除非没有任何事件发生
否则线程就一直都会干活
但是再多事件发生，也只会有指定数量的线程在干活
如果硬件性能差，则把数量减少一些
如果硬件性能好，则把线程数量增加一些
整个服务器的性能峰值就可以控制在手上了

在这种反应堆模式中，几个概念是需要强调的

- Handle（句柄）

具体的事件源，可以是文件描述符、网络套接字等等

- Synchronous Event Demultiplexer 同步事件分离器

分离器一般是系统的接口
比如 select、poll 或者 epoll 函数
这些东西将程序的状态由事件触发状态切换到事件处理状态
比如 select 会阻塞，直到某个 select 关注的 handle 产生事件

- Event Handler 事件处理器

这个元素里面一般包含一个回调函数
当 handle 上产生事件的时候
这个回调函数则可能被执行

- Concrete Event Handler 具体的事件处理器

注意，这个一般是事件处理器的子类
会实现具体的回调，完成业务逻辑

- Initiation Dispatcher 初始分发器

提供注册、删除与转发 event handler 的方法
当同步事件分离器（Synchronous Event Demultiplexer）发现某个 handle 上有事件的时候
就会通知初始分发器（本概念）来调用事件处理器来处理事件

✧ 工作流程
■ 用户需要在初始分发器中注册具体的事件处理器
说明在什么事件发生的时候，调用本具体的事件处理器
注意，注册的时候，也要事件处理器绑定好对应的句柄

■ 注册完成后，初始分发器会开启事件循环，然后使得同步事件分离器来等待事件的发生

■ 当某个句柄上产生的事件为就绪的时候
同步事件分离器就会通知初始化分发器

■ 初始化分发器会调用事件处理器的回调：
通过事件来定位对应的句柄和句柄回调方法

■ 具体的事件处理器会调用器内部关联的回调方法来处理具体的事件

使用木铎库创建一个简单的服务器

```
#include "EdoyumIMServer.h"
#include <signal.h>
#include <fcntl.h>
#include "base/Logging.h"
#include "net/EventLoop.h"
#include "net/EventLoopThread.h"
#include "net/EventLoopThreadPool.h"
#include "net/TcpServer.h"

using namespace std;

void show_help(const char* cmd)
{
    cout << "found error argument!\r\n";
    cout << "Usage:" << std::endl;
    cout << cmd << " [-d]" << std::endl;
    cout << "\t-d run in daemon mode.\r\n";
}

void signal_exit(int signum)
{
    cout << "signal " << signum << " found, exit ...\r\n";
```

```
    //TODO:退出的清除
    switch (signum)
    {
    case SIGINT:
    case SIGKILL:
    case SIGTERM:
    case SIGILL:
    case SIGSEGV:
    case SIGTRAP:
    case SIGABRT:
        //TODO:
        break;
    default:
        //TODO:
        break;
    }
    exit(signum);
}

void daemon()
{
    signal(SIGCHLD, SIG_IGN);
    int pid = fork();
    if (pid < 0) {
        cout << "fork call error,code is " << pid << " error code is " << errno << std::endl;
        exit(-1);
    }
    if (pid > 0) {//主进程在此结束
        exit(0);
    }
    //这里的代码只可能是子进程了
    //这里可以避免父进程所在的会话结束时，把子进程带走
    //不让前台影响后台
    setsid();

    //不让后台影响前台
    int fd = open("/dev/null", O_RDWR, 0);
    cout << "invoke success!" << std::endl;
    cout << "STDIN_FILENO is " << STDIN_FILENO << std::endl;
    cout << "STDOUT_FILENO is " << STDOUT_FILENO << std::endl;
    cout << "STDERR_FILENO is " << STDERR_FILENO << std::endl;
    cout << "fd is " << fd << std::endl;
    if (fd != -1) {
        dup2(fd, STDIN_FILENO);
```

```
        dup2(fd, STDOUT_FILENO);
        dup2(fd, STDERR_FILENO);
    }
    if (fd > STDERR_FILENO)
        close(fd);
}
void onConnection(const muduo::net::TcpConnectionPtr& conn)
{
    cout << conn->name() << std::endl;
}

void onMessage(const muduo::net::TcpConnectionPtr& conn,
    muduo::net::Buffer* buf,
    muduo::Timestamp time)
{
    conn->send(buf);
    conn->shutdown();
}

int main(int argc, char* argv[], char* env[])
{
    signal(SIGCHLD, SIG_DFL);
    signal(SIGPIPE, SIG_IGN);//网络当中，管道操作
    signal(SIGINT, signal_exit);//中断错误
    signal(SIGKILL, signal_exit);
    signal(SIGTERM, signal_exit);//ctrl + c
    signal(SIGILL, signal_exit);//非法指令错误
    signal(SIGSEGV, signal_exit);//段错误
    signal(SIGTRAP, signal_exit);//ctrl + break
    signal(SIGABRT, signal_exit);//abort 函数调用触发

    cout << "imchatserver is invoking ..." << endl;
    int ch = 0;
    //sever -a    server a
    bool is_daemon = false;
    while ((ch = getopt(argc, argv, "d")) != -1) {
        cout << "ch = " << ch << std::endl;
        cout << "current " << optind << " value:" << argv[optind - 1] << std::endl;
        switch (ch)
        {
        case 'd':
            is_daemon = true;
            break;
        default:
```

```
            show_help(argv[0]);
            return -1;
        }
    }

    if (is_daemon) {
        daemon();
    }

    muduo::net::EventLoop loop;
    muduo::net::InetAddress addr(9527);
    muduo::net::TcpServer server(&loop, addr, "echo server");
    server.setConnectionCallback(onConnection);
    server.setMessageCallback(onMessage);
    server.start();
    loop.loop();

    return 0;
}
```

用木铎库实现一个单例服务器

预备知识：
几个关键字=default =delete explicit implicit

=default
=delete

explicit
implicit

decltype

PTHREAD_ONCE_INIT

SFINAE Substitution failure is not an error 替换失败不是错误

Linux 下数据库的设计与开发

MySQL 数据的基本接口与封装

内容见课程
附件：Ascii 码表

Dec	Hx	Oct	Char	Dec	Hx	Oct	Chr	Dec	Hx	Oct	Chr	Dec	Hx	Oct	Chr
0	0	000	NUL (null)	32	20	040	Space	64	40	100	@	96	60	140	`
1	1	001	SOH (start of heading)	33	21	041	!	65	41	101	A	97	61	141	a
2	2	002	STX (start of text)	34	22	042	"	66	42	102	B	98	62	142	b
3	3	003	ETX (end of text)	35	23	043	#	67	43	103	C	99	63	143	c
4	4	004	EOT (end of transmission)	36	24	044	$	68	44	104	D	100	64	144	d
5	5	005	ENQ (enquiry)	37	25	045	%	69	45	105	E	101	65	145	e
6	6	006	ACK (acknowledge)	38	26	046	&	70	46	106	F	102	66	146	f
7	7	007	BEL (bell)	39	27	047	'	71	47	107	G	103	67	147	g
8	8	010	BS (backspace)	40	28	050	(	72	48	110	H	104	68	150	h
9	9	011	TAB (horizontal tab)	41	29	051	)	73	49	111	I	105	69	151	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	74	4A	112	J	106	6A	152	j
11	B	013	VT (vertical tab)	43	2B	053	+	75	4B	113	K	107	6B	153	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	76	4C	114	L	108	6C	154	l
13	D	015	CR (carriage return)	45	2D	055	-	77	4D	115	M	109	6D	155	m
14	E	016	SO (shift out)	46	2E	056	.	78	4E	116	N	110	6E	156	n
15	F	017	SI (shift in)	47	2F	057	/	79	4F	117	O	111	6F	157	o
16	10	020	DLE (data link escape)	48	30	060	0	80	50	120	P	112	70	160	p
17	11	021	DC1 (device control 1)	49	31	061	1	81	51	121	Q	113	71	161	q
18	12	022	DC2 (device control 2)	50	32	062	2	82	52	122	R	114	72	162	r
19	13	023	DC3 (device control 3)	51	33	063	3	83	53	123	S	115	73	163	s
20	14	024	DC4 (device control 4)	52	34	064	4	84	54	124	T	116	74	164	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	85	55	125	U	117	75	165	u
22	16	026	SYN (synchronous idle)	54	36	066	6	86	56	126	V	118	76	166	v
23	17	027	ETB (end of trans. block)	55	37	067	7	87	57	127	W	119	77	167	w
24	18	030	CAN (cancel)	56	38	070	8	88	58	130	X	120	78	170	x
25	19	031	EM (end of medium)	57	39	071	9	89	59	131	Y	121	79	171	y
26	1A	032	SUB (substitute)	58	3A	072	:	90	5A	132	Z	122	7A	172	z
27	1B	033	ESC (escape)	59	3B	073	;	91	5B	133	[	123	7B	173	{
28	1C	034	FS (file separator)	60	3C	074	<	92	5C	134	\	124	7C	174	|
29	1D	035	GS (group separator)	61	3D	075	=	93	5D	135	]	125	7D	175	}
30	1E	036	RS (record separator)	62	3E	076	>	94	5E	136	^	126	7E	176	~
31	1F	037	US (unit separator)	63	3F	077	?	95	5F	137	_	127	7F	177	DEL

业务的开发

内容见视频课程