易播Linux后台服务器

开发环境搭建

预备程序

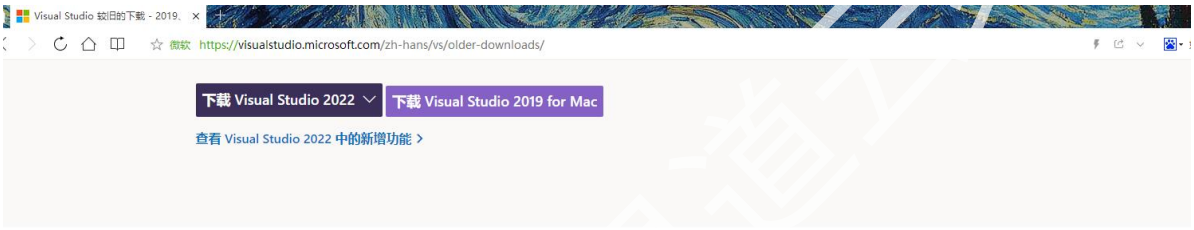
以下程序联系助教获取

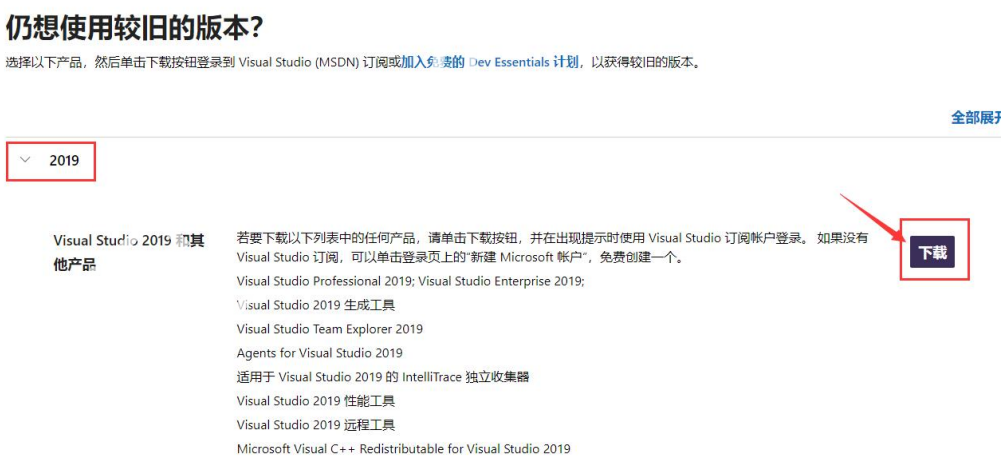
- 虚拟机VMware和Ubuntu虚拟机
- Visual Studio 2019
- SimpleRemote（ssh工具）
- FileZilla（FTP文件传输工具）

Visual Studio 2019的下载

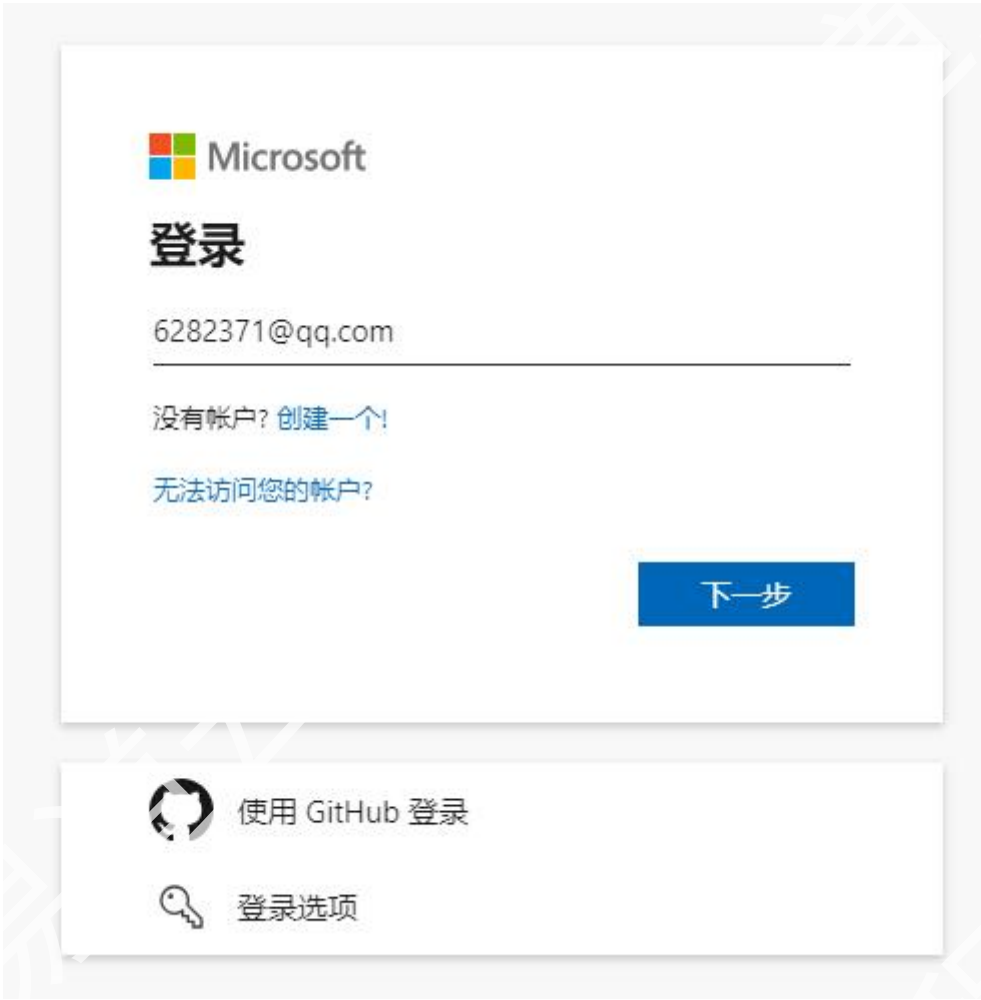
首先打开https://visualstudio.microsoft.com/zh-hans/vs/older-downloads/

按照下图所示，展开2019栏目，点击下载



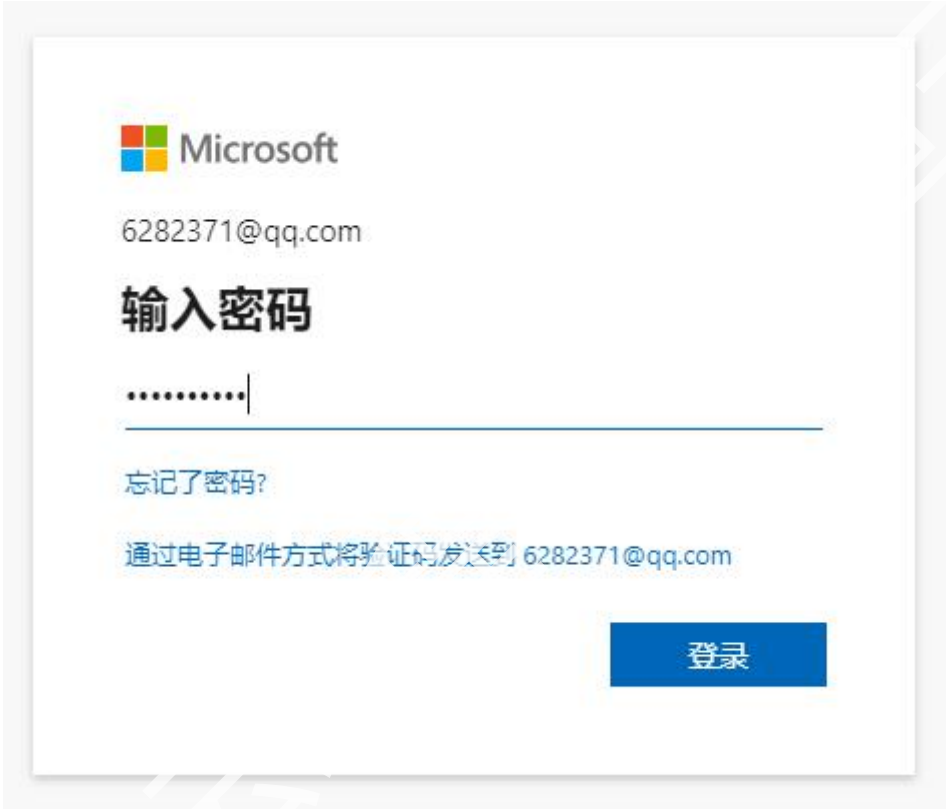


注意，如果没有在微软注册账号，此时会弹出下图所示的对话框



在上面输入账号（如果没有可以点击创建一个，使用qq邮箱创建一个）

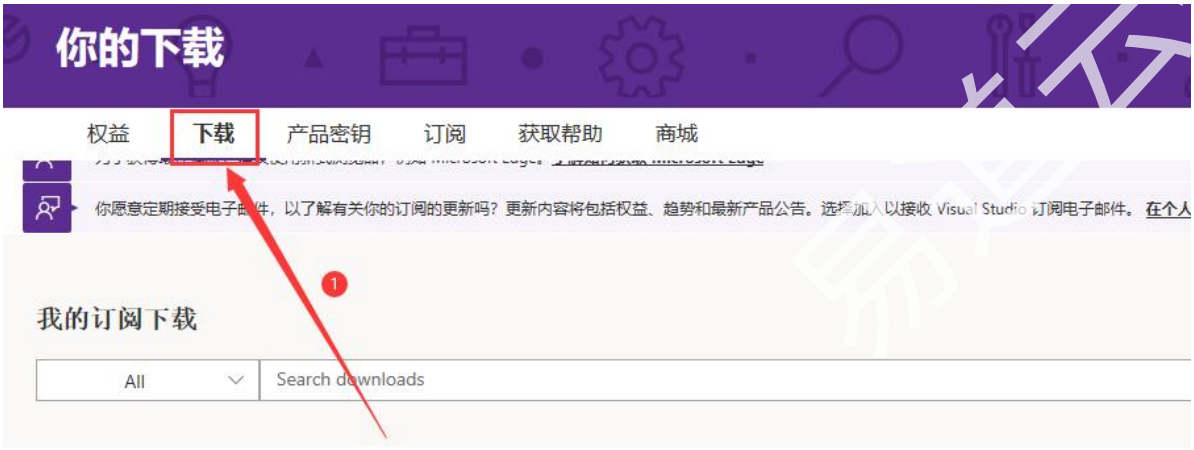
输入账号之后，点击下一步，进入密码对话框



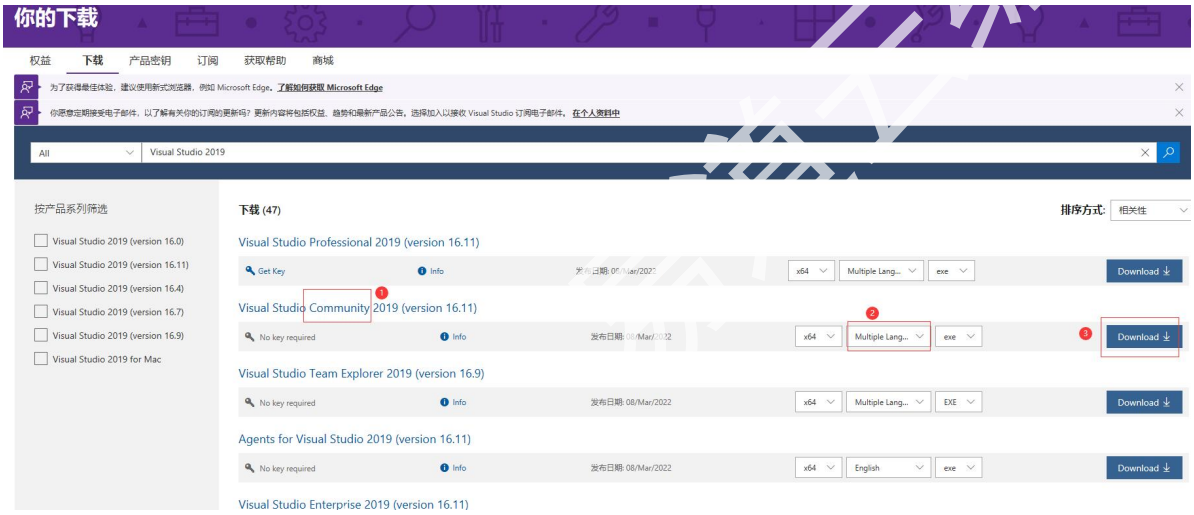
然后输入密码，点击登录

成功登录之后，会提示你是否保持登录状态，勾选保持登录状态

接着会进入如下页面



按照顺序点击页面中红色方框标记的位置，即可进入最终的下载页面



注意1这里，一定要选择Community（社区）版本。

这个是官方免费正版的，和专业版（Professional）以及团队版（Team Explorer）差异并没有太大。

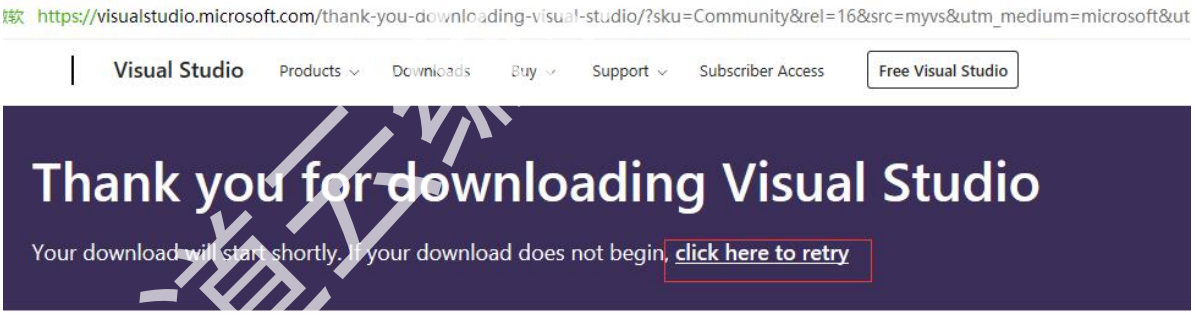
初学者强烈建议使用该版本！！！

第二个点就是语言，一定要使用多语言版本，否则界面可能不是中文的！

参考图中2标识的位置，进行选择

最后就可以点击下载了！

如果下载没有开始，并且长时间停留在这个页面，那么可以点击下图中的连接，再次激发下载



可以看到下载界面

下载完成后，点击程序开始安装

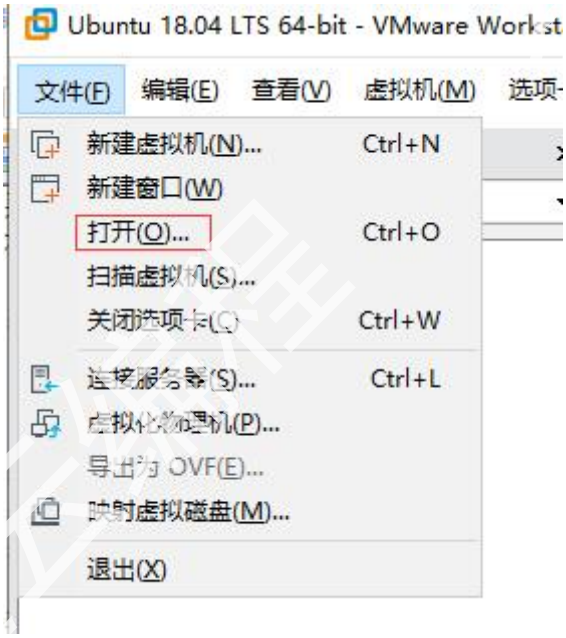
注意，不要随意修改安装路径

如果默认的C盘空间不足，**可以修改盘符，但是不要修改路径！！！**

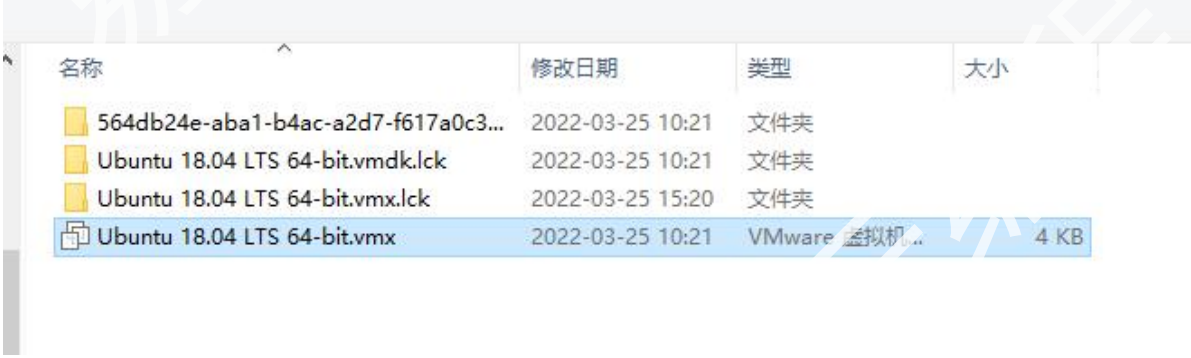
2019的下载就介绍到这里，详细的安装过程，可以参考VS的安装视频。

虚拟机环境介绍

启动VMware Workstation之后，使用菜单里面**文件→打开**来打开虚拟机



在弹出的窗口里面选择vmx文件

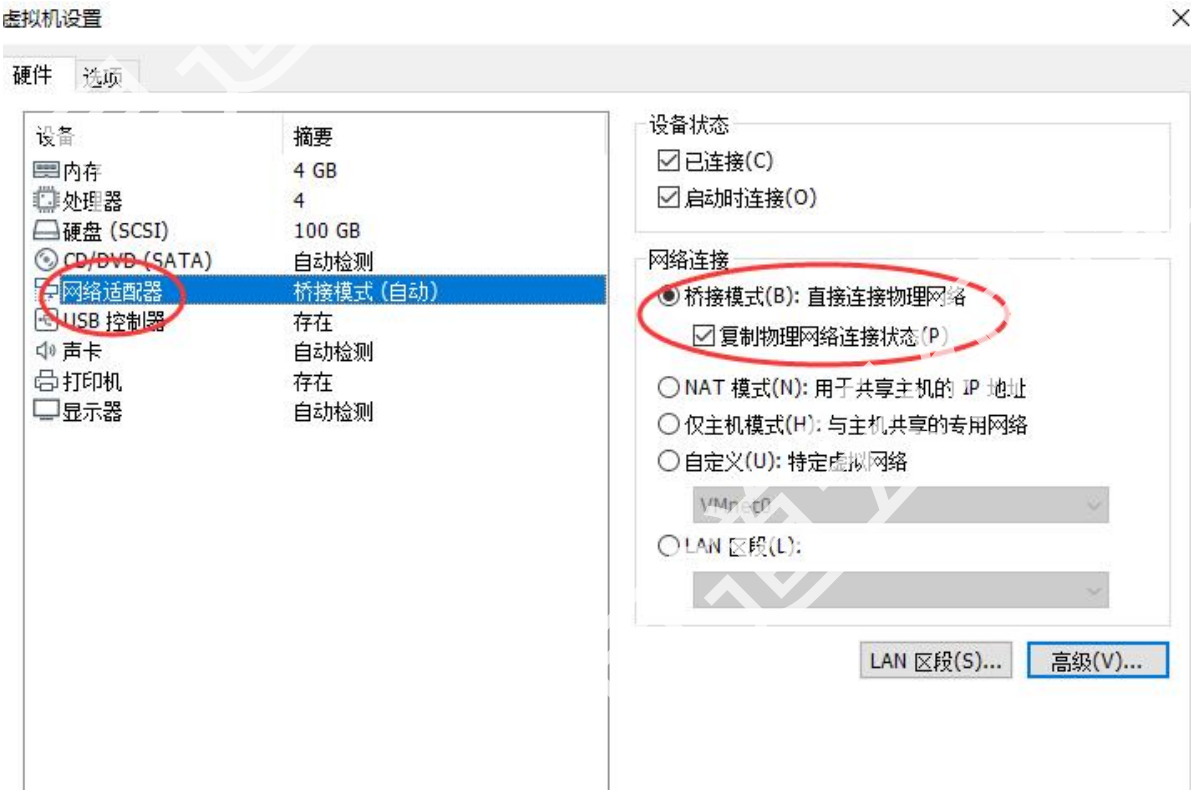


注意，VM会问你虚拟机是哪里来的。一个是复制来的，一个是移动来的。

选择复制来的。

然后按照下图所示打开虚拟机设置

在**硬件→网络适配器→桥接模式**设置网络，结果如下图：



另外依据自己的机器调整**内存大小**和**处理器数量**

然后启动虚拟机，来到登录界面：

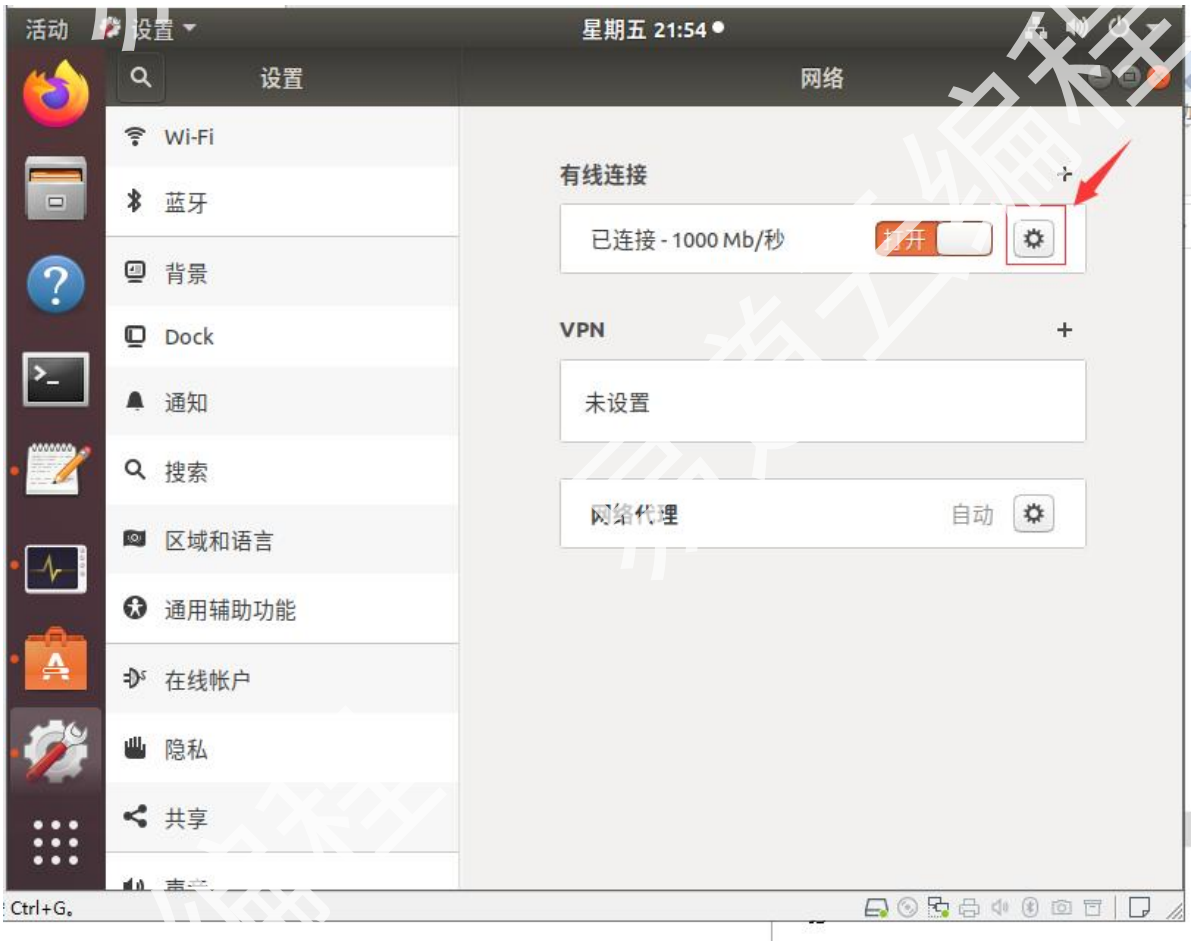
点击ydy，输入密码ydy619619进行登录

登录后，大概是这个样子：

点击左上角的网络按钮

点击有线连接-→有线设置

打开网络设置，点击下图所示按钮



然后在网络设置页面查看ip

记下这个ip，后面会用到！！！

如果这个IP和你自己主机的IP地址不在一个网段

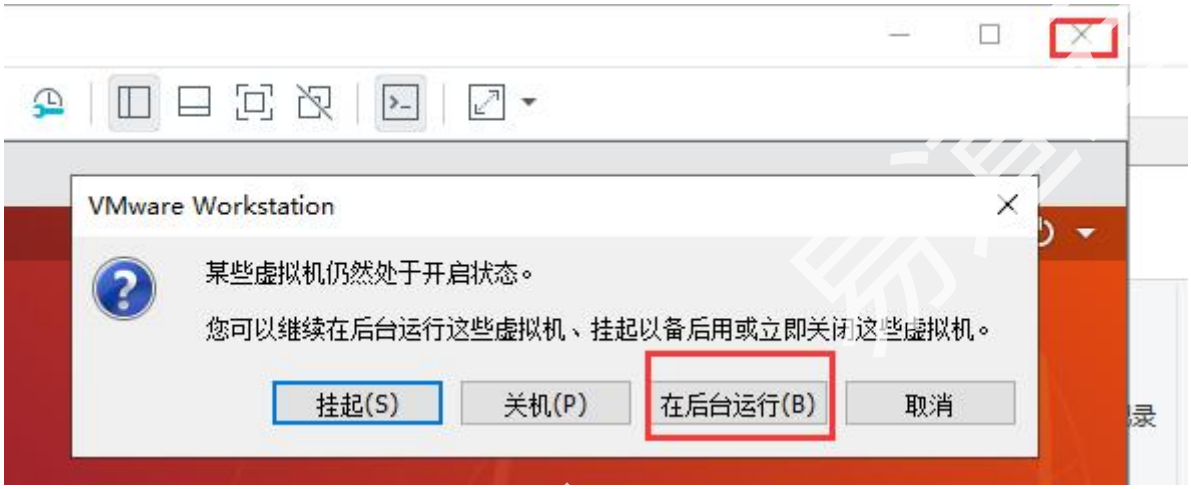
可以IPv4页面进行修改，如下图

至此虚拟机的IP已经拿到（或者设置完成）

请务必让虚拟机使用固定IP，否则后面会因为IP变化而导致各种设置失效

如果IP和局域网其他机器冲突，那么请调整IP地址，直到没有冲突产生

虚拟机成功启动之后，可以点击虚拟机右上角的关闭按钮，然后选择在后台运行



如上图所示。

这样可以节约一定的系统资源。

SimpleRemote的使用

在任意目录建立一个SimpleRemote的文件夹

然后将SimpleRemote.exe复制到该文件夹，即完成SimpleRemote的安装

双击exe打开软件，右键单击，打开新建菜单，如下图：

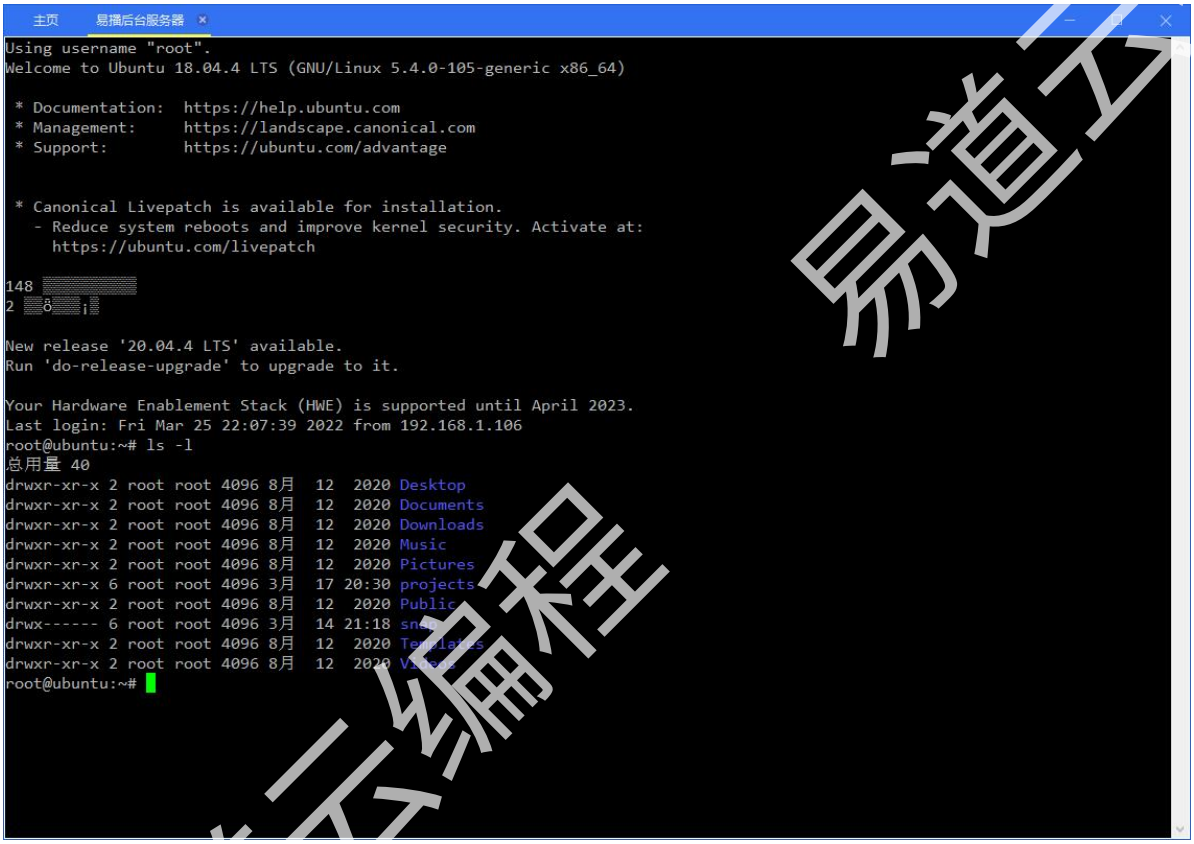
选择**SSH连接**

将前面我们记录下来的ip、账号和密码，按照下图进行填写：

记住，字符集那里要填写为utf-8

否则后面中文显示可能会出现乱码！

然后双击标签栏，或者输入回车，即可进入ssh界面：



输入`ls -l`命令

如果能够看到**中文显示的月字**，则表示一切正常，配置正确

如果需要关机，输入

```
shutdown -P 0
```

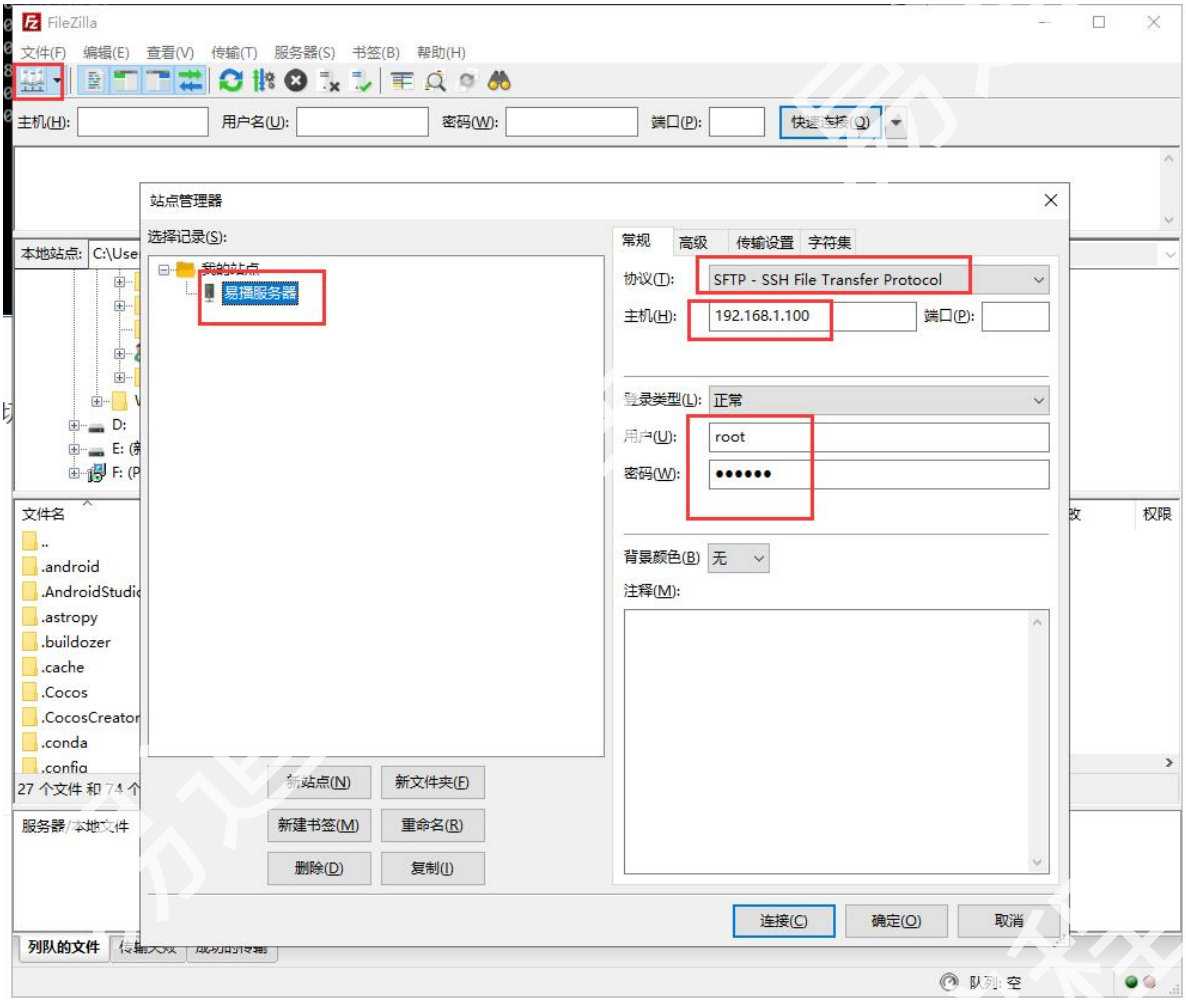
则虚拟机会在一分钟内（依据主机性能来决定，可能会超过一分钟）自动关闭。

后面我们就可以开始愉快的学习了！

FileZilla的使用

通过FileZilla_3.58.0_win64-setup.exe安装文件，按照默认配置，安装好FileZilla之后，就可以启动FileZilla了

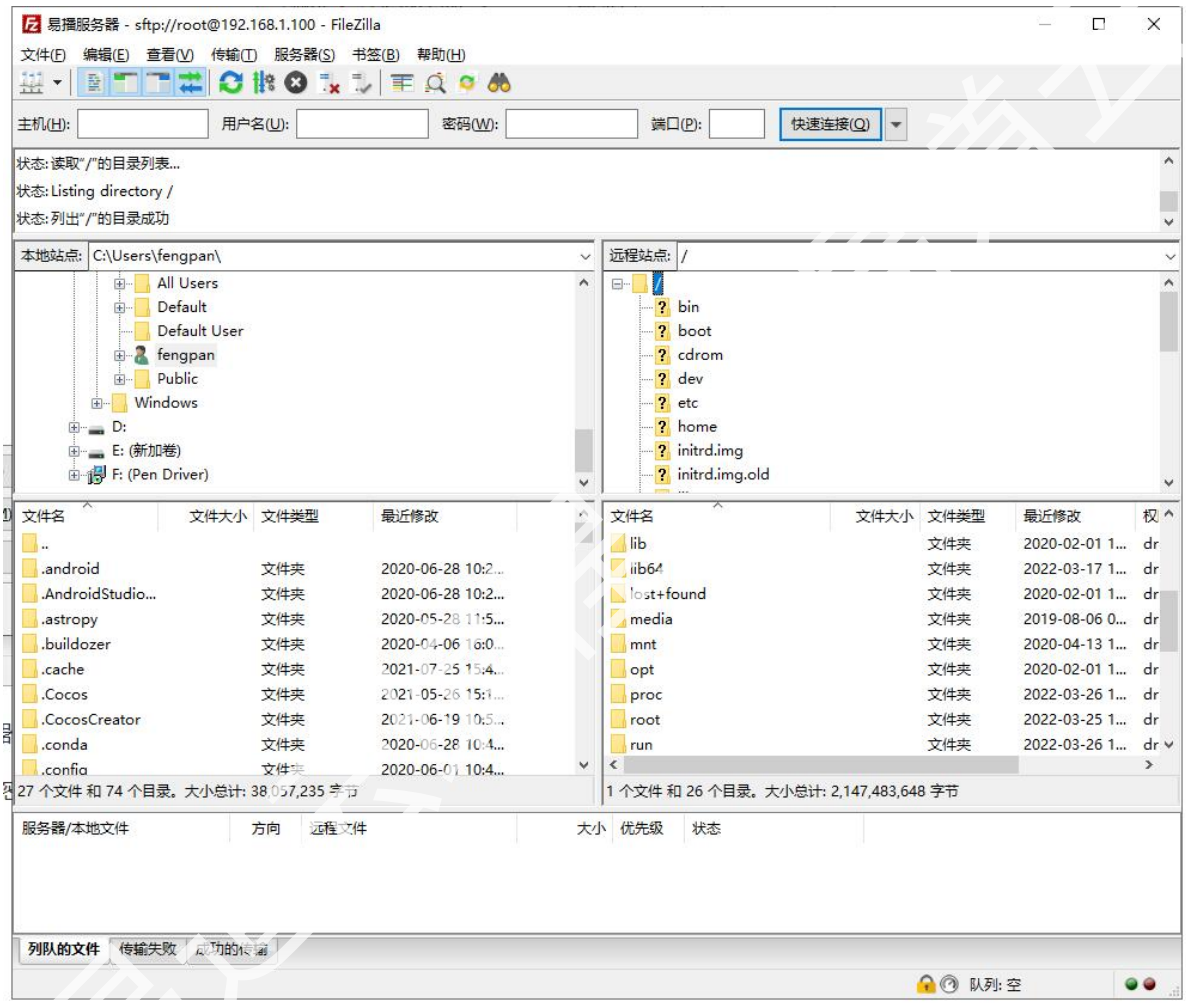
启动之后，在左上角，点击下图红色方框的按钮，即可打开连接配置（如下图）



然后在我的站点，重命名站点为易播服务器。

右边选择SFTP，依次填入主机IP、用户、密码（`root`、`123456`）

然后点击连接即可进入工作模式：



左边是本机地址，右边是远程的服务器地址。

可以将本地文件上传到服务器，也可以将远程的服务器文件下载到本地。

这个工具提供一个文件上传和下载的功能，非常方便。

项目的开发

前面我们已经搭建好了开发环境，安装好了开发软件。

下面我们就从零开始，一点点的把项目实现。

项目的创建

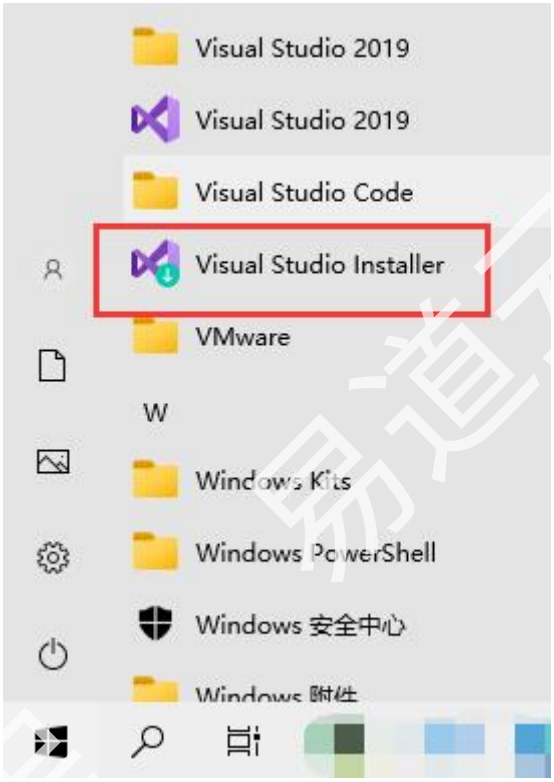
打开Visual Studio 2019

选择右下角的**创建新项目**

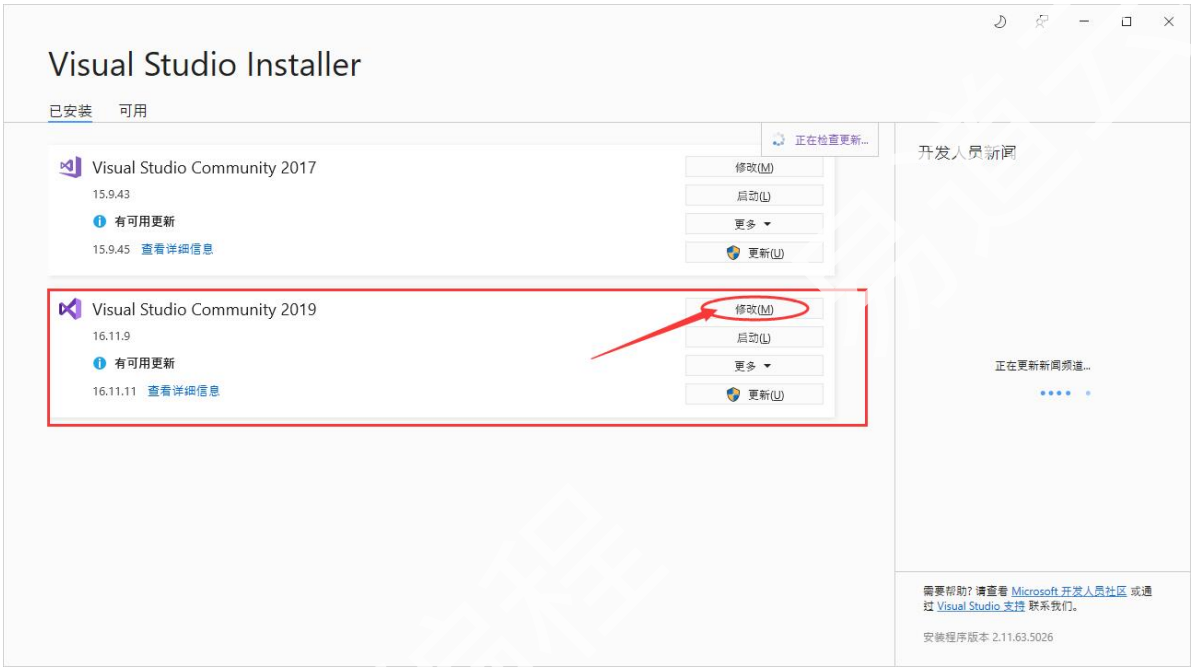
然后选择C++、Linux、控制台里面的**控制台应用程序**

然后点击下一步

如果你没有这个，请在开始菜单里面找到visual studio installer



打开后，找到2019的社区版，点击修改：

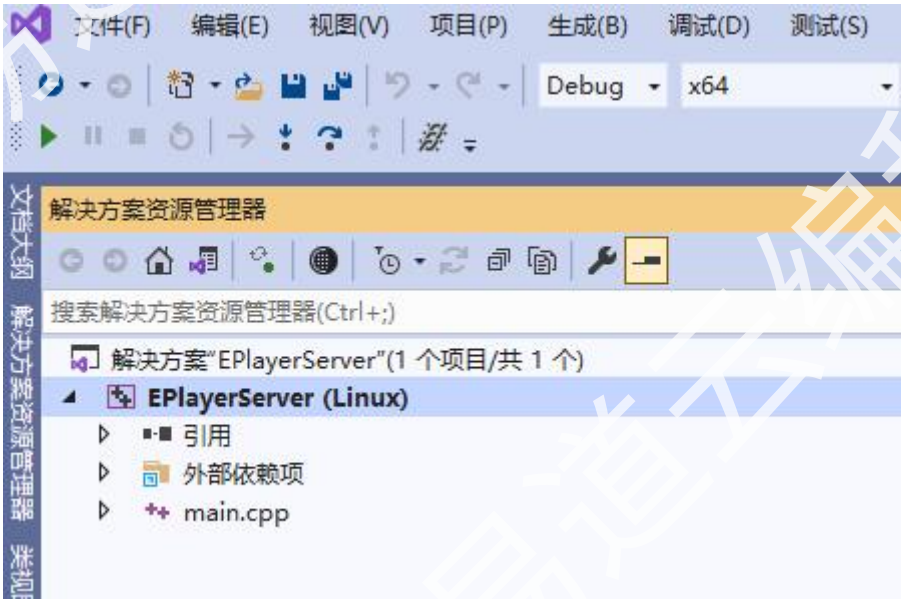


然后看看**使用C++的Linux开发**是否勾选：

如果没有勾选，则勾选上，再重复前面的操作即可。

然后按照下图输入项目名称（**EPlayerServer**）和路径：

然后点击创建按钮，即可进入项目目录

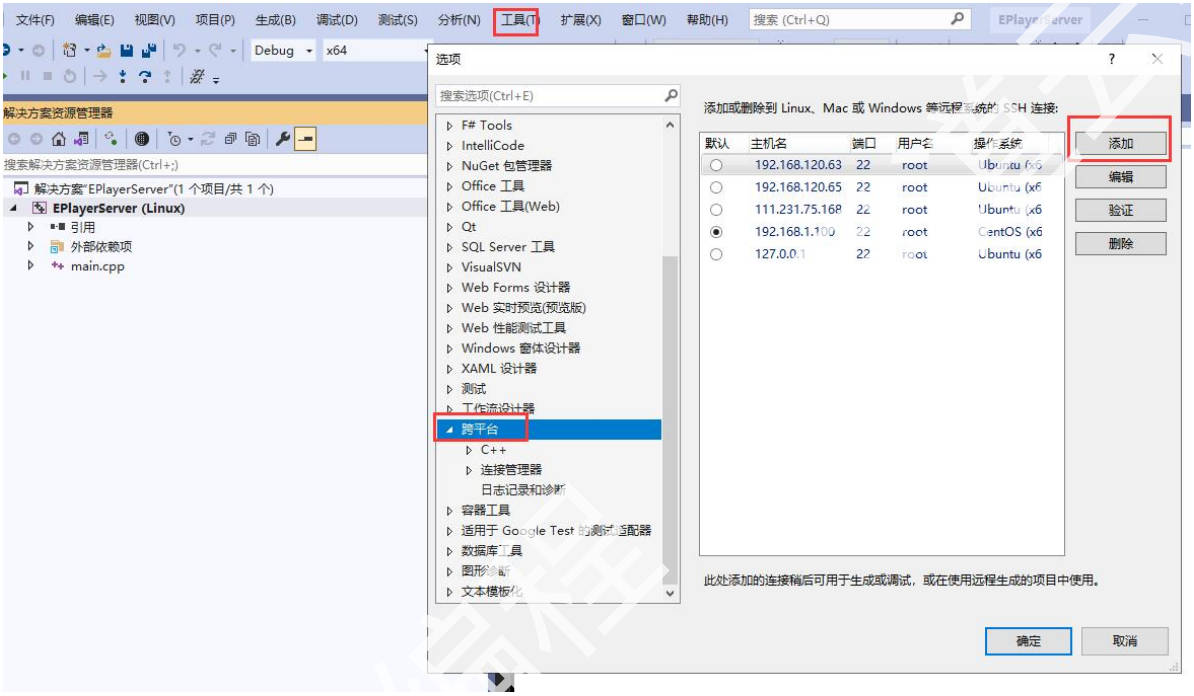


到目前为止，我们的项目算是建立了。

但是开发环境还需要一些设置。

首先需要确保虚拟机已经打开，并且虚拟系统Ubuntu已经启动。

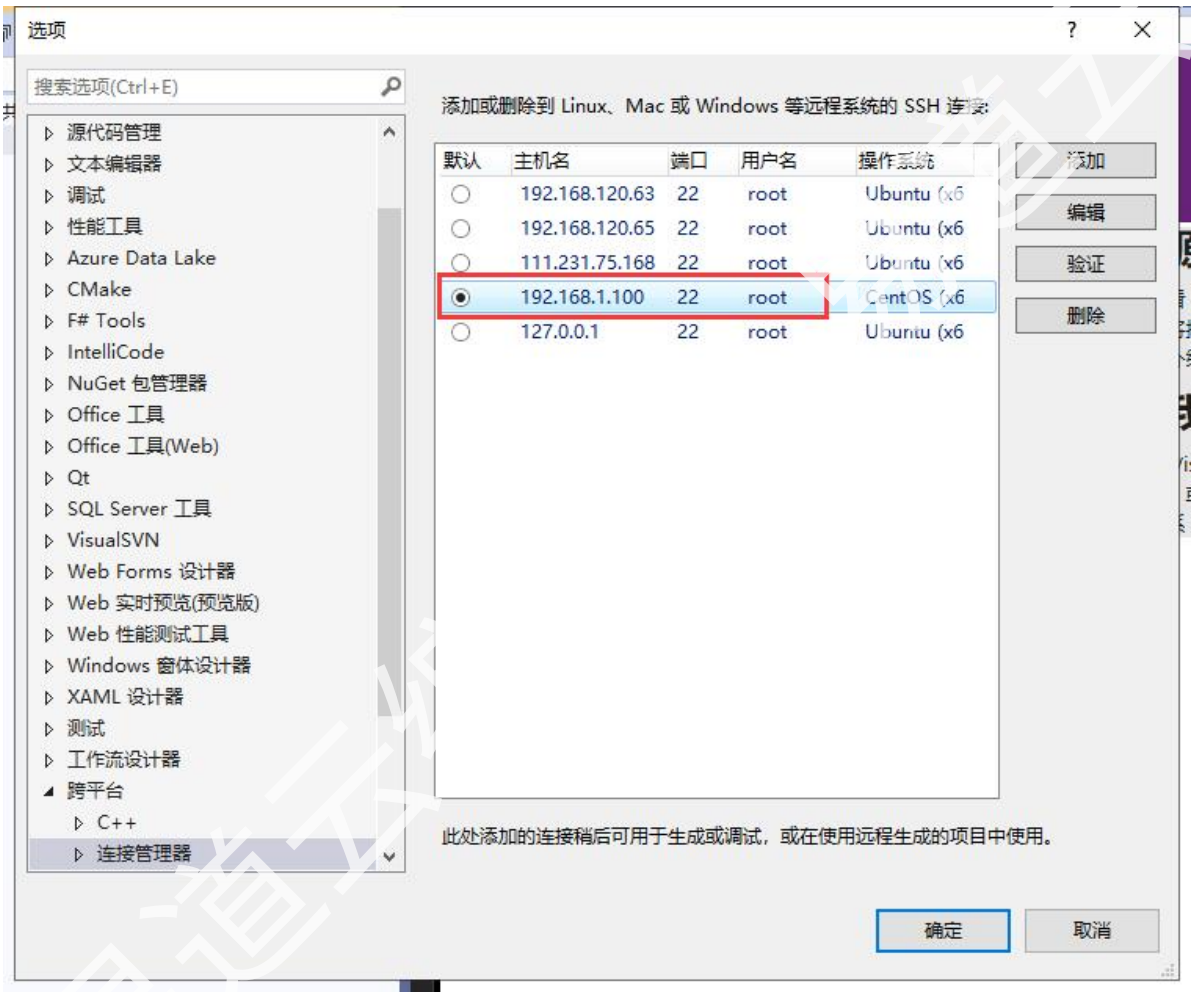
然后我们需要在**菜单→工具→选项→跨平台→连接管理器**



点击**添加**按钮，进入远程连接界面：

输入虚拟机的IP地址、用户名和密码，然后点击连接。

成功后，我们会看到如下内容：

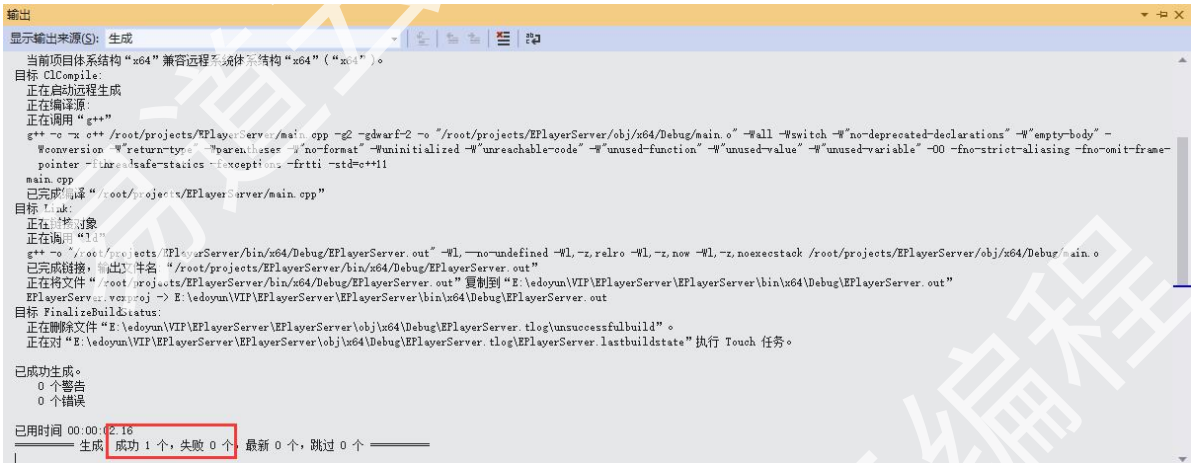


则表示成功。（注意，操作系统有可能识别不正确，但是对开发没有影响）

勾选我们新加的账户（默认那一排，让我们新加的账户处于上图状态即可）

点击确认按钮，回到项目页面

点击**生成**下面的**生成解决方案**



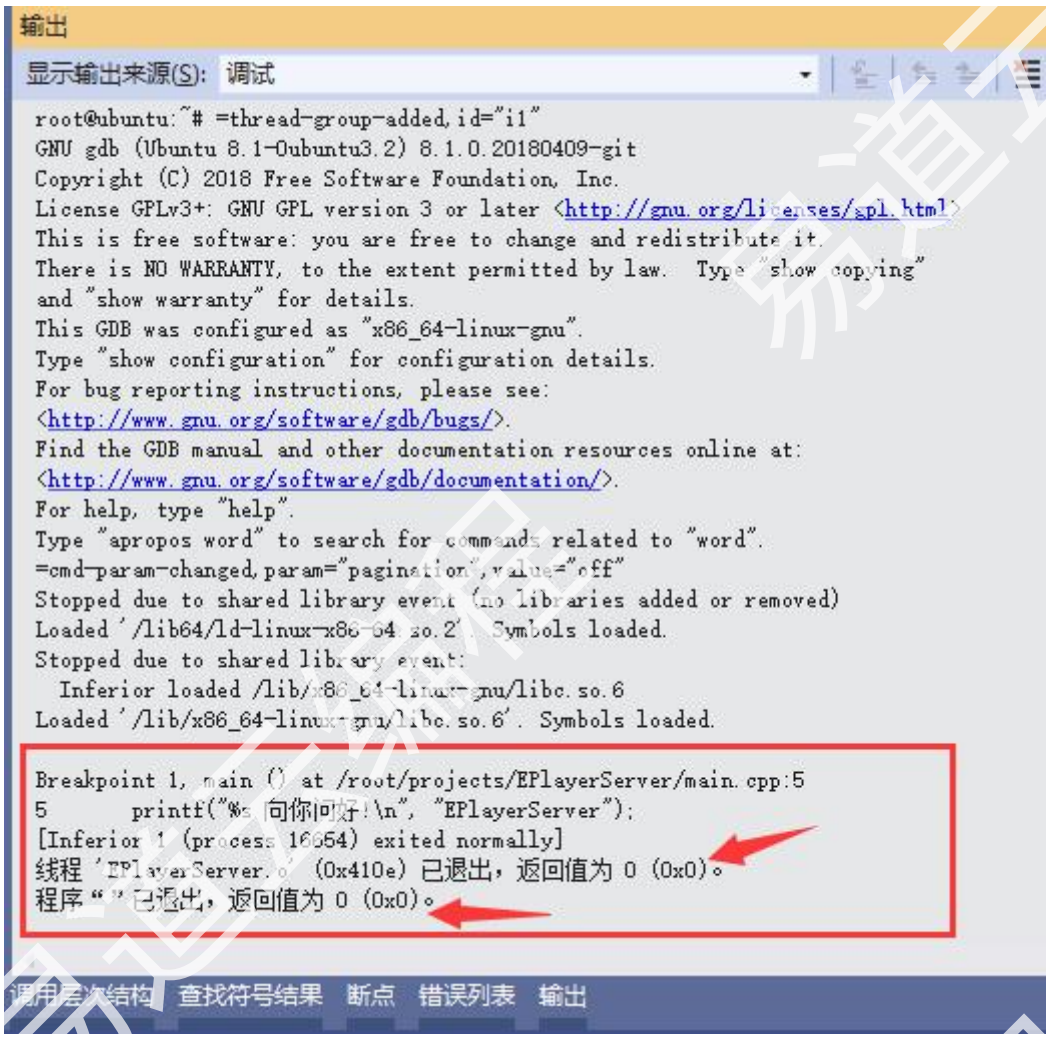
看到生成成功1个，则表示一切ok。

如果有错误，看看前面的操作是否存在问题，再运行一次。

然后点击下图所示的按钮：

尝试运行程序。

看到下面的结果，则表示运行一切正常。



返回值为0，程序返回值也为0，说明项目配置一切ok，我们就可以开始正式的代码开发了。

项目会在虚拟机系统里面的：

`/root/projects/EPlayerServer/bin/x64/Debug`

该路径也可以写作`~/projects/EPlayerServer/bin/x64/Debug`

下面。

我们登录SimpleRemote之后

可以通过`cd /root/projects/EPlayerServer/bin/x64/Debug`

命令进入该路径。

使用`ls -l`来查看目录下面的文件

使用`./EPlayerServer.out`来运行程序

过程如下图：

我们可以在控制台看到中英文显示！

至此，项目建立完成，并且环境确认无误。我们就可以开始后面的项目开发啦！

进程和进程的创建

线程默认是进程内竞争，而进程是操作系统资源分配最小的调度单位。

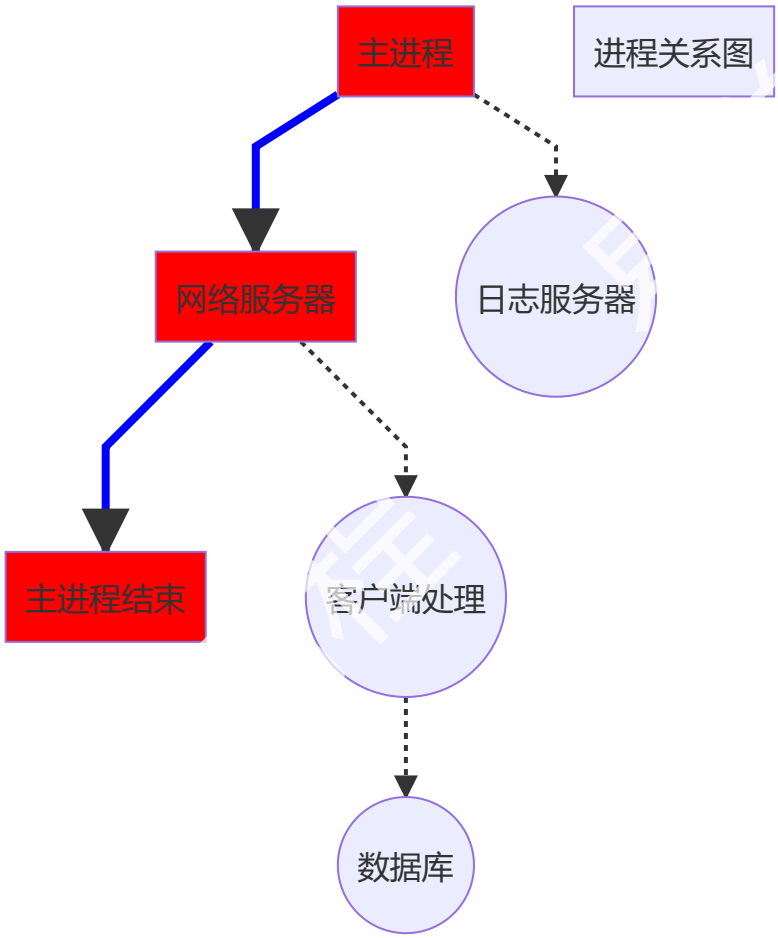
这也就意味着，如果要充分利用系统资源，最好的形式是多线程多进程模式。

所以我们最好将一个整体功能，分散到多个进程之中，从而实现资源利用率的最大化。

否则就只能多个线程在一个进程内进行竞争，没法充分利用系统的资源。

毕竟多个进程竞争资源，比一个进程竞争资源，要有利得多。

下图是我们这个服务器项目要实现的进程结构图



图中方框部分都是主进程模块，圆框则是子进程。

主进程只负责网络服务器部分，接入客户端，其他一概不管。

日志则由日志服务器进程来处理。

接入客户端之后，发送给客户端处理进程。

如果处理过程需要数据库，则和数据库进程进行交互。

这样，将一个进程完成的事情，分成了四个进程进行。

而且每个进程中可以依据自己的需求，开启多个线程来完成。

在Linux中，开启进程一般通过exec系列函数或者fork函数来完成。

即使是exec函数，也会要使用到fork函数。

所以开启进程，fork函数是无法绕开的。

而fork函数会对线程造成影响，所以我们一定要先定好进程结构，然后再开启线程。

首先，由于线程无法被复制，所以在子进程中，一些线程会消失（没有被复制过来）
其次，如果程序逻辑依赖多线程模式的时候，fork可能在子进程中破坏掉这种模式，进而使得程序出现无法预料的问题。
所以一定要先准备好进程结构，再去使用线程！！！

由于数据库我们最后会使用MySQL，而MySQL进程是由第三方提供并随服务器启动而启动的服务程序。

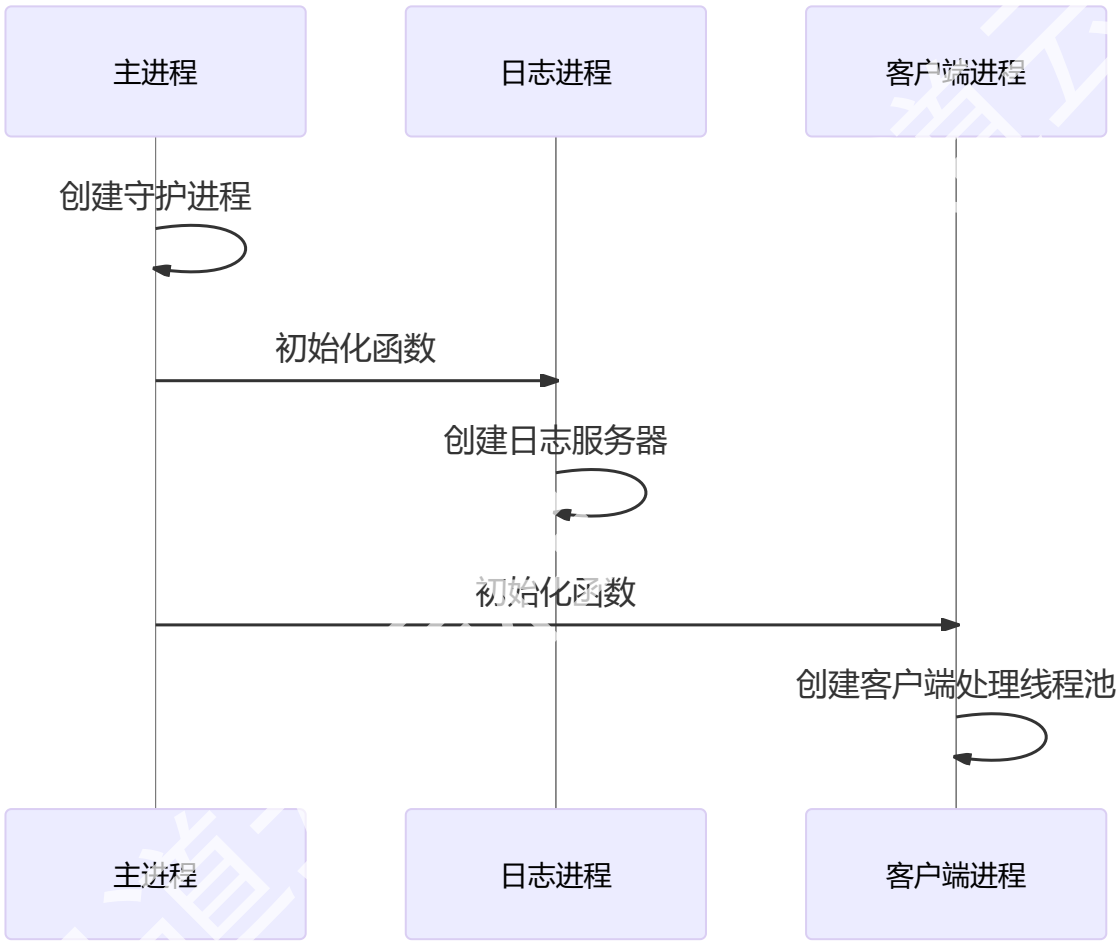
所以我们最终要生成的进程是日志进程和客户端通信处理进程。

这意味着我们需要在一开始，就分离出两个子进程，分别处理日志和客户端

由于日志进程在后台服务器程序中的重要作用。

所以日志子进程应该优先创建，然后再创建客户端处理子进程。

所以整个进程的创建顺序，会按照进程关系图中所示顺序，进行创建。



进程模块的实现方案

创建进程的流程和结构，我们现在已经知道了。

但是如何实现，还有几个问题，需要我们一个一个去解决。

首先，每个子进程的逻辑并不一样，所需要的参数可能相互冲突。

那么如何满足这些需求呢？

其次，客户端处理进程，需要处理客户端。

我们这是一个网络程序，主进程接收到客户端之后，如何通知子进程去处理呢？

客户端这个时候是一个文件描述符，怎么告诉子进程去处理呢？

所以我们需要两个功能：

- 灵活的进程入口函数

- 进程间传递文件描述符

第二个功能我们稍后再说，我们先讲讲第一个功能

这个功能可以有三种做法：

1. **使用无属性的指针参数和固定参数的进程入口函数来实现**
2. **使用面向对象的参数和统一的进程入口函数来实现**
3. **使用模板函数来实现**

这三种方式都可以实现，但是方便程度和安全性不一样。

第一种方式**技术上最简单**，但是类型在转换的时候，可能出现问题。

而且可以传入的参数数量是固定的，以后其他项目很难复用此代码。

第二种方式比第一种好了不少。**参数不是固定的，可移植性强了很多。**

但是这种方式需要专门写一个参数封装和解析的代码。

这种解析代码的复用性会比较差。

因为每个进程的任务不一样，参数也不一样，参数的含义也可能大相径庭。

第三种方式难度最大，但是**使用起来最方便，可以移植性最强。**

参数可以随时修改，函数也可以是类的成员函数。

此外参数无需解析，直接原样转发到目标函数。

实现起来也不需要太多代码，stl里面准备好了很多工具，可以直接使用。

就是模板编程不太好理解。

我们这里将采取第三种方式来实现。

进程入口函数的实现

fork函数介绍

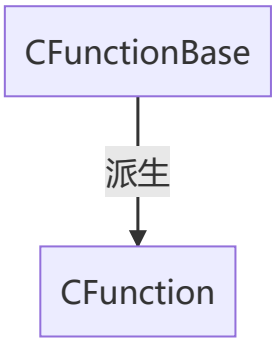
```
#include <unistd.h>
pid_t fork(void);
```

返回值：

主进程中，会返回子进程的pid。

子进程中，返回值为0。

如果失败，返回-1。




```
#include <cstdio>

#include <unistd.h>
#include <functional>

class CFunctionBase
{
public:
    virtual ~CFunctionBase() {}
    virtual int operator()() = 0;
```

```
};

template<typename _FUNCTION_, typename... _ARGS_>
class CFunction :public CFunctionBase
{
public:
    CFunction(_FUNCTION_ func, _ARGS_... args) {
    }
    virtual ~CFunction() {}
    virtual int operator()() {
        return m_binder();
    }
    std::_Bindres_helper<int, _FUNCTION_, _ARGS_...>::type m_binder;
};

class CProcess
{
public:
    CProcess() {
        m_func = NULL;
    }
    ~CProcess() {
        if (m_func != NULL) {
            delete m_func;
            m_func = NULL;
        }
    }

    template<typename _FUNCTION_, typename... _ARGS_>
    int SetEntryFunction(_FUNCTION_ func, _ARGS_... args)
    {
        m_func = new CFunction(func, args...);
        return 0;
    }
```

```
45
46      int CreateSubProcess() {
47          if (m_func == NULL)return -1;
48          pid_t pid = fork();
49          if (pid == -1)return -2;
50          if (pid == 0) {
51              //子进程
52              return (*m_func)();
53          }
54          //主进程
55          m_pid = pid;
56          return 0;
57      }
58
59  private:
60      CFunctionBase* m_func;
61      pid_t m_pid;
62  };
63
64
65  int CreateLogServer(CProcess* proc)
66  {
67      return 0;
68  }
69
70  int CreateClientServer(CProcess* proc)
71  {
72      return 0;
73  }
74
75  int main()
76  {
77      CProcess proclog,proccliets;
78      proclog.SetEntryFunction(CreateLogServer,
    &proclog);
79      int ret = proclog.CreateSubProcess();
```

```
80
    procclients.SetEntryFunction(CreateClientServer,
    &procclients);
81      ret = procclients.CreateSubProcess();
82      return 0;
83  }
```

进程间文件描述符的实现

```
1  #include <unistd.h>
2  #include <sys/types.h>
3  #include <functional>
4  #include <memory.h>
5  #include <sys/socket.h>
6
7
8  class CFunctionBase
9  {
10 public:
11     virtual ~CFunctionBase() {}
12     virtual int operator()() = 0;
13 };
14
15 template<typename _FUNCTION_, typename...
   _ARGS_>
16 class CFunction :public CFunctionBase
17 {
18 public:
19     CFunction(_FUNCTION_ func, _ARGS_... args)
20         :m_binder(std::forward<_FUNCTION_>
   (func), std::forward<_ARGS_>(args)...)
21     {}
22     virtual ~CFunction() {}
23     virtual int operator()() {
24         return m_binder();
25     }
```

```
    typename std::_Bindres_helper<int, _FUNCTION_, _ARGS_...>::type m_binder;
};

class CProcess
{
public:
    CProcess() {
        m_func = NULL;
        memset(pipes, 0, sizeof(pipes));
    }
    ~CProcess() {
        if (m_func != NULL) {
            delete m_func;
            m_func = NULL;
        }
    }

    template<typename _FUNCTION_, typename... _ARGS_>
    int SetEntryFunction(_FUNCTION_ func, _ARGS_... args)
    {
        m_func = new CFunction<_FUNCTION_, _ARGS_...>(func, args...);
        return 0;
    }

    int CreateSubProcess() {
        if (m_func == NULL)return -1;
        int ret = socketpair(AF_LOCAL, SOCK_STREAM, 0, pipes);
        if (ret == -1)return -2;
        pid_t pid = fork();
        if (pid == -1)return -3;
        if (pid == 0) {
            //子进程
```

```
58              close(pipes[1]);//关闭掉写
59              pipes[1] = 0;
60              return (*m_func)();
61          }
62          //主进程
63          close(pipes[0]);
64          pipes[0] = 0;
65          m_pid = pid;
66          return 0;
67      }
68
69      int SendFD(int fd) {//主进程完成
70          struct msghdr msg;
71          iovec iov[2];
72          iov[0].iov_base = (char*)"edoyun";
73          iov[0].iov_len = 7;
74          iov[1].iov_base = (char*)"jueding";
75          iov[1].iov_len = 8;
76          msg.msg_iov = iov;
77          msg.msg_iovlen = 2;
78
79          // 下面的数据，才是我们需要传递的。
80          cmsghdr* cmsg = (cmsghdr*)calloc(1,
CMSG_LEN(sizeof(int)));
81          if (cmsg == NULL)return -1;
82          cmsg->cmsg_len = CMSG_LEN(sizeof(int));
83          cmsg->cmsg_level = SOL_SOCKET;
84          cmsg->cmsg_type = SCM_RIGHTS;
85          *(int*)CMSG_DATA(cmsg) = fd;
86          msg.msg_control = cmsg;
87          msg.msg_controllen = cmsg->cmsg_len;
88
89          ssize_t ret = sendmsg(pipes[1], &msg,
0);
90          free(cmsg);
91          if (ret == -1) {
92              return -2;
```

```
93          }
94          return 0;
95      }
96
97      int RecvFD(int& fd)
98      {
99          msghdr msg;
100         iovec iov[2];
101         char buf[][10] = { "","" };
102         iov[0].iov_base = buf[0];
103         iov[0].iov_len = sizeof(buf[0]);
104         iov[1].iov_base = buf[1];
105         iov[1].iov_len = sizeof(buf[1]);
106         msg.msg_iov = iov;
107         msg.msg_iovlen = 2;
108
109         cmsghdr* cmsg = (cmsghdr*)calloc(1,
CMSG_LEN(sizeof(int)));
110         if (cmsg == NULL)return -1;
111         cmsg->cmsg_len = CMSG_LEN(sizeof(int));
112         cmsg->cmsg_level = SOL_SOCKET;
113         cmsg->cmsg_type = SCM_RIGHTS;
114         msg.msg_control = cmsg;
115         msg.msg_controllen =
CMSG_LEN(sizeof(int));
116         ssize_t ret = recvmsg(pipes[0], &msg,
0);
117         if (ret == -1) {
118             free(cmsg);
119             return -2;
120         }
121         fd = *(int*)CMSG_DATA(cmsg);
122         return 0;
123     }
124
125
126 private:
```

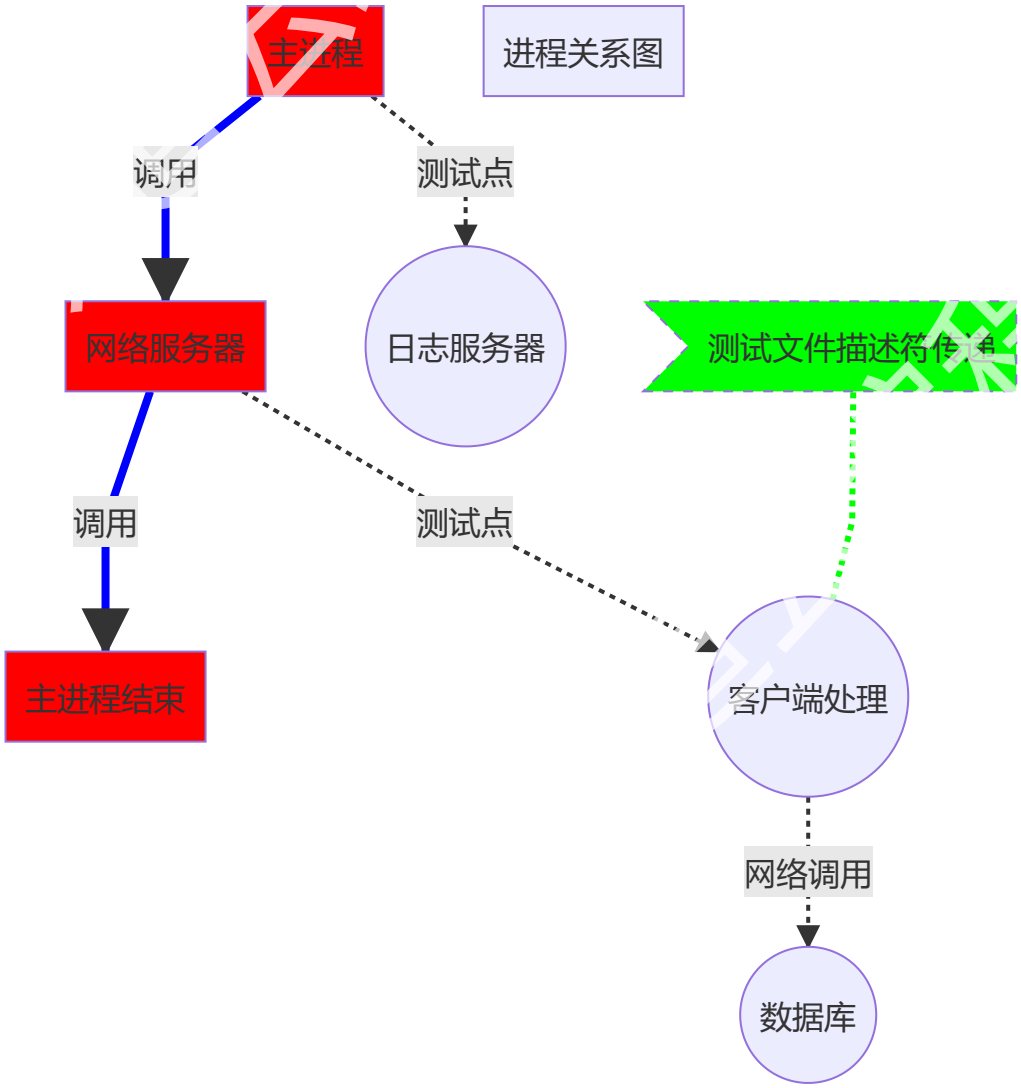
```
127     CFunctionBase* m_func;
128     pid_t m_pid;
129     int pipes[2];
130 };
```

进程代码测试

测试点的设置：

- 进程分离
- 文件描述符传递

关键点如下图



测试代码如下：

```cpp
#include <cstdio>

#include <unistd.h>
#include <sys/types.h>
#include <functional>
#include <memory.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <fcntl.h>


class CFunctionBase
{
public:
    virtual ~CFunctionBase() {}
    virtual int operator()() = 0;
};

template<typename _FUNCTION_, typename...
_ARGS_>
class CFunction :public CFunctionBase
{
public:
    CFunction(_FUNCTION_ func, _ARGS_... args)
        :m_binder(std::forward<_FUNCTION_>
(func), std::forward<_ARGS_>(args)...)
    {}
    virtual ~CFunction() {}
    virtual int operator()() {
        return m_binder();
    }
    typename std::_Bindres_helper<int,
_FUNCTION_, _ARGS_...>::type m_binder;
};

class CProcess
{
```

```
35 public:
36     CProcess() {
37         m_func = NULL;
38         memset(pipes, 0, sizeof(pipes));
39     }
40     ~CProcess() {
41         if (m_func != NULL) {
42             delete m_func;
43             m_func = NULL;
44         }
45     }
46 
47     template<typename _FUNCTION_, typename...
_ARGS_>
48     int SetEntryFunction(_FUNCTION_ func,
_ARGS_... args)
49     {
50         m_func = new CFunction<_FUNCTION_,
_ARGS_...>(func, args...);
51         return 0;
52     }
53 
54     int CreateSubProcess() {
55         if (m_func == NULL)return -1;
56         int ret = socketpair(AF_LOCAL,
SOCK_STREAM, 0, pipes);
57         if (ret == -1)return -2;
58         pid_t pid = fork();
59         if (pid == -1)return -3;
60         if (pid == 0) {
61             //子进程
62             close(pipes[1]);//关闭掉写
63             pipes[1] = 0;
64             ret = (*m_func)();
65             exit(0);
66         }
67         //主进程
```

```
68          close(pipes[0]);
69          pipes[0] = 0;
70          m_pid = pid;
71          return 0;
72      }
73
74      int SendFD(int fd) {//主进程完成
75          struct msghdr msg;
76          iovec iov[2];
77          char buf[2][10] = { "edoyun","jueding"
};
78          iov[0].iov_base = buf[0];
79          iov[0].iov_len = sizeof(buf[0]);
80          iov[1].iov_base = buf[1];
81          iov[1].iov_len = sizeof(buf[1]);
82          msg.msg_iov = iov;
83          msg.msg_iovlen = 2;
84
85          // 下面的数据，才是我们需要传递的。
86          cmsghdr* cmsg = (cmsghdr*)calloc(1,
CMSG_LEN(sizeof(int)));
87          if (cmsg == NULL)return -1;
88          cmsg->cmsg_len = CMSG_LEN(sizeof(int));
89          cmsg->cmsg_level = SOL_SOCKET;
90          cmsg->cmsg_type = SCM_RIGHTS;
91          *(int*)CMSG_DATA(cmsg) = fd;
92          msg.msg_control = cmsg;
93          msg.msg_controllen = cmsg->cmsg_len;
94
95          ssize_t ret = sendmsg(pipes[1], &msg,
0);
96          free(cmsg);
97          if (ret == -1) {
98              return -2;
99          }
100         return 0;
101     }
```

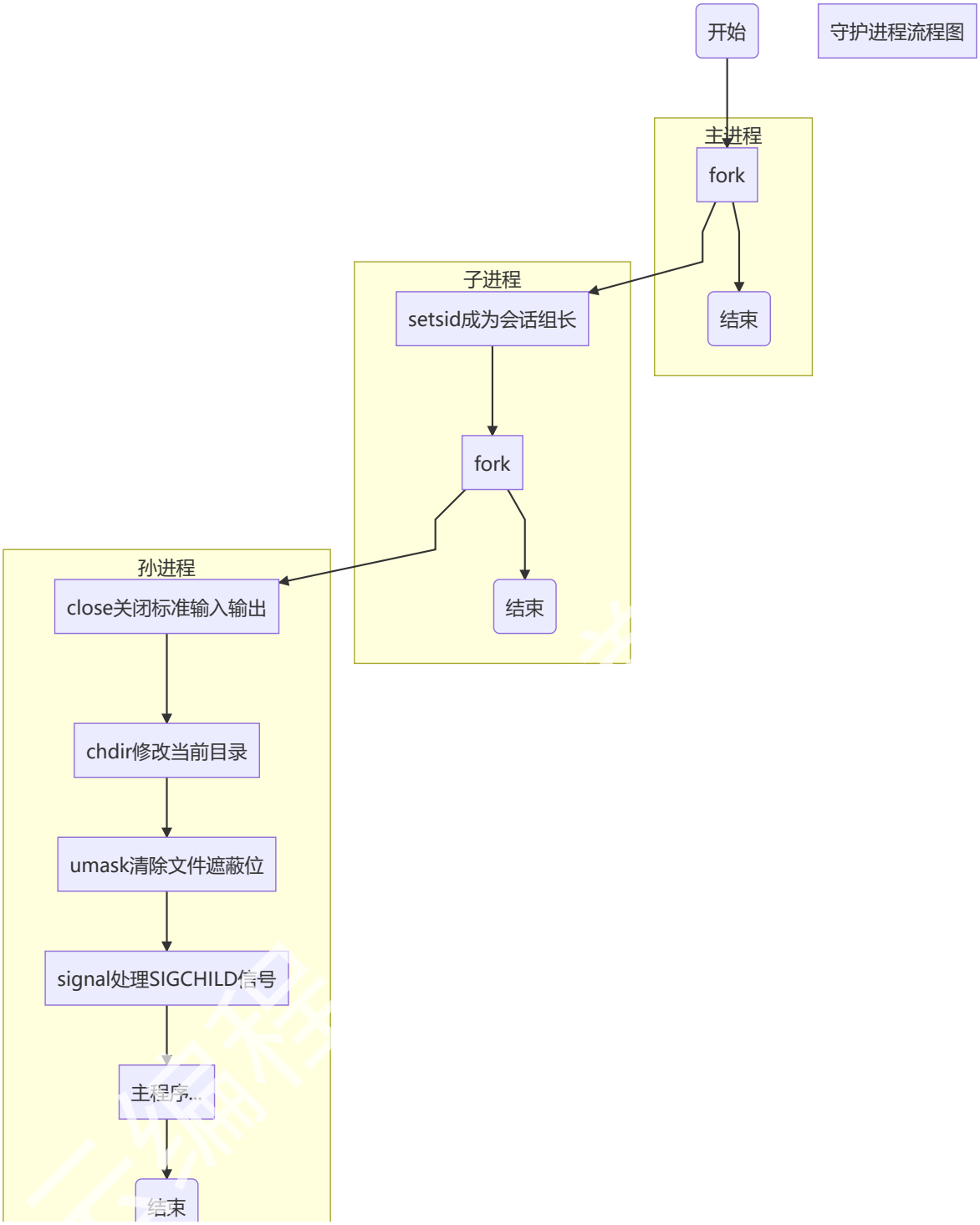
```
102
103     int RecvFD(int& fd)
104     {
105         msghdr msg;
106         iovec iov[2];
107         char buf[][10] = { "","" };
108         iov[0].iov_base = buf[0];
109         iov[0].iov_len = sizeof(buf[0]);
110         iov[1].iov_base = buf[1];
111         iov[1].iov_len = sizeof(buf[1]);
112         msg.msg_iov = iov;
113         msg.msg_iovlen = 2;
114
115         cmsghdr* cmsg = (cmsghdr*)calloc(1,
    CMSG_LEN(sizeof(int)));
116         if (cmsg == NULL)return -1;
117         cmsg->cmsg_len = CMSG_LEN(sizeof(int));
118         cmsg->cmsg_level = SOL_SOCKET;
119         cmsg->cmsg_type = SCM_RIGHTS;
120         msg.msg_control = cmsg;
121         msg.msg_controllen =
    CMSG_LEN(sizeof(int));
122         ssize_t ret = recvmsg(pipes[0], &msg,
    0);
123         if (ret == -1) {
124             free(cmsg);
125             return -2;
126         }
127         fd = *(int*)CMSG_DATA(cmsg);
128         free(cmsg);
129         return 0;
130     }
131
132
133 private:
134     CFunctionBase* m_func;
135     pid_t m_pid;
```

```
136     int pipes[2];
137 };
138
139
140 int CreateLogServer(CProcess* proc)
141 {
142     printf("%s(%d):<%s> pid=%d\n", __FILE__,
    __LINE__, __FUNCTION__, getpid());
143     return 0;
144 }
145
146 int CreateClientServer(CProcess* proc)
147 {
148     printf("%s(%d):<%s> pid=%d\n", __FILE__,
    __LINE__, __FUNCTION__, getpid());
149     int fd = -1;
150     int ret = proc->RecvFD(fd);
151     printf("%s(%d):<%s> ret=%d\n", __FILE__,
    __LINE__, __FUNCTION__, ret);
152     printf("%s(%d):<%s> fd=%d\n", __FILE__,
    __LINE__, __FUNCTION__, fd);
153     sleep(1);
154     char buf[10] = "";
155     lseek(fd, 0, SEEK_SET);
156     read(fd, buf, sizeof(buf));
157     printf("%s(%d):<%s> buf=%s\n", __FILE__,
    __LINE__, __FUNCTION__, buf);
158     close(fd);
159     return 0;
160 }
161
162 int main()
163 {
164     CProcess proclog, procclients;
165     printf("%s(%d):<%s> pid=%d\n", __FILE__,
    __LINE__, __FUNCTION__, getpid());
```

```
166     proclog.SetEntryFunction(CreateLogServer,
    &proclog);
167     int ret = proclog.CreateSubProcess();
168     if (ret != 0) {
169         printf("%s(%d):<%s> pid=%d\n", __FILE__,
    __LINE__, __FUNCTION__, getpid());
170         return -1;
171     }
172     printf("%s(%d):<%s> pid=%d\n", __FILE__,
    __LINE__, __FUNCTION__, getpid());
173 
    procclients.SetEntryFunction(CreateClientServer,
    &procclients);
174     ret = procclients.CreateSubProcess();
175     if (ret != 0) {
176         printf("%s(%d):<%s> pid=%d\n", __FILE__,
    __LINE__, __FUNCTION__, getpid());
177         return -2;
178     }
179     printf("%s(%d):<%s> pid=%d\n", __FILE__,
    __LINE__, __FUNCTION__, getpid());
180     usleep(100 * 000);
181     int fd = open("./test.txt", O_RDWR | O_CREAT
    | O_APPEND);
182     printf("%s(%d):<%s> fd=%d\n", __FILE__,
    __LINE__, __FUNCTION__, fd);
183     if (fd == -1)return -3;
184     ret = procclients.SendFD(fd);
185     printf("%s(%d):<%s> ret=%d\n", __FILE__,
    __LINE__, __FUNCTION__, ret);
186     if (ret != 0)printf("errno:%d msg:%s\n",
    errno, strerror(errno));
187     write(fd, "edoyun", 6);
188     close(fd);
189     return 0;
190 }
```

守护进程的实现

守护进程的流程



守护进程实现代码如下：

```
static int SwitchDeamon() {
        pid_t ret = fork();
        if (ret == -1)return -1;
        if (ret > 0)exit(0);//主进程到此为止
        //子进程内容如下
        ret = setsid();
        if (ret == -1)return -2;//失败，则返回
        ret = fork();
        if (ret == -1)return -3;
        if (ret > 0)exit(0);//子进程到此为止
        //孙进程的内容如下，进入守护状态
        for (int i = 0; i < 3; i++) close(i);
        umask(0);
        signal(SIGCHLD, SIG_IGN);
        return 0;
    }
```

日志模块的设计

现在我们一开始就是多进程模式了，所以直接就可以上进程间通信。

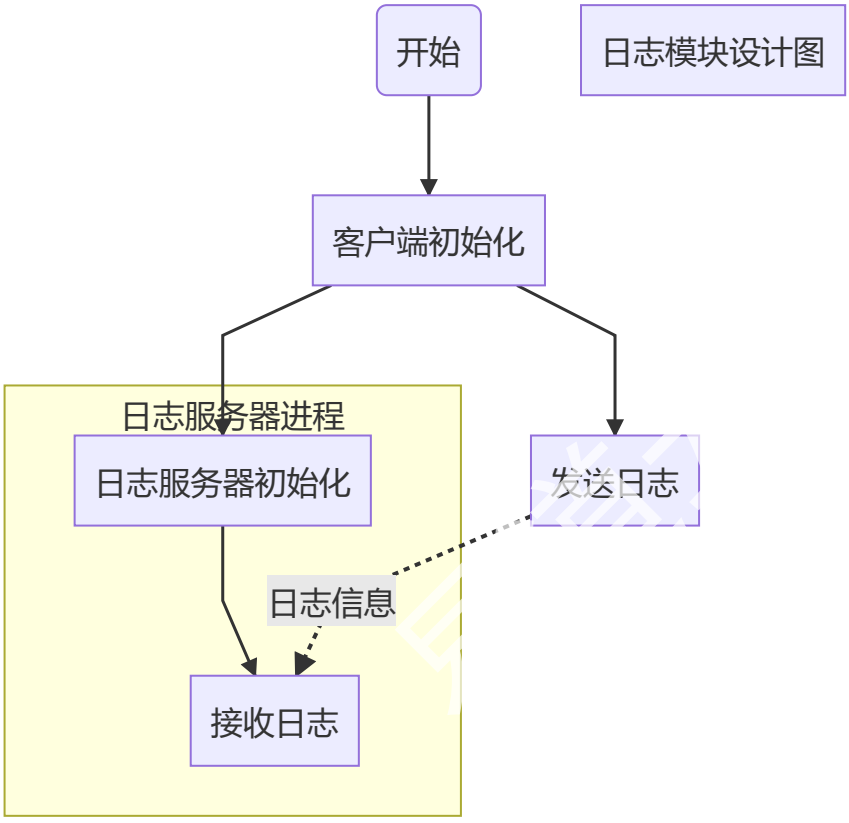
进程间通信，最方便最快速的就是**本地套接字**通信了。

- 文件通信磁盘速度慢
- 管道在多线程环境下不太方便（可能会出现内容插入），而且是**单向**的。
- 信号量**信息量太少**
- 内存共享需要反复加锁同步，否则可能出现问题
- 消息函数（sendmsg、recvmsg）需要创建时确定
- 网络套接字通信，需要额外的IP和端口

所以本地套接字是最佳选择

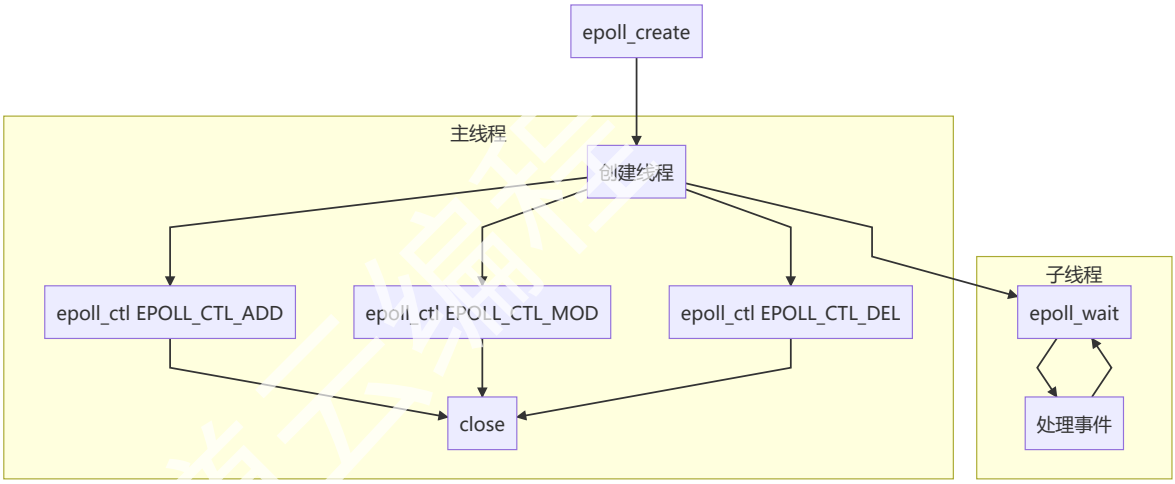
- 无需IP和端口，不影响服务器对外的资源
- 信息无需加锁，可以多线程并发写
- 数据传输量巨大，传输速率高（纯内存读写）

日志模块的设计图



Epoll的封装

epoll简单模型



接口：

```cpp
#pragma once
#include <unistd.h>
#include <sys/epoll.h>
#include <vector>
#include <errno.h>
#include <sys/signal.h>
#include <memory.h>

#define EVENT_SIZE  128
class EpollData
{
public:
    EpollData() { m_data.u64 = 0; }
    EpollData(void* ptr) { m_data.ptr = ptr; }
```

```cpp
    explicit EpollData(int fd) { m_data.fd = fd; }
    explicit EpollData(uint32_t u32) { m_data.u32 = u32; }
    explicit EpollData(uint64_t u64) { m_data.u64 = u64; }
    EpollData(const EpollData& data) { m_data.u64 = data.m_data.u64; }
public:
    EpollData& operator=(const EpollData& data)
    {
        if (this != &data)
            m_data.u64 = data.m_data.u64;
        return *this;
    }
    EpollData& operator=(void* data) {
        m_data.ptr = data;
        return *this;
    }
    EpollData& operator=(int data) {
        m_data.fd = data;
        return *this;
    }
    EpollData& operator=(uint32_t data) {
        m_data.u32 = data;
        return *this;
    }
    EpollData& operator=(uint64_t data) {
        m_data.u64 = data;
        return *this;
    }
    operator epoll_data_t() { return m_data; }
    operator epoll_data_t()const { return m_data; }
    operator epoll_data_t* () { return &m_data; }
```

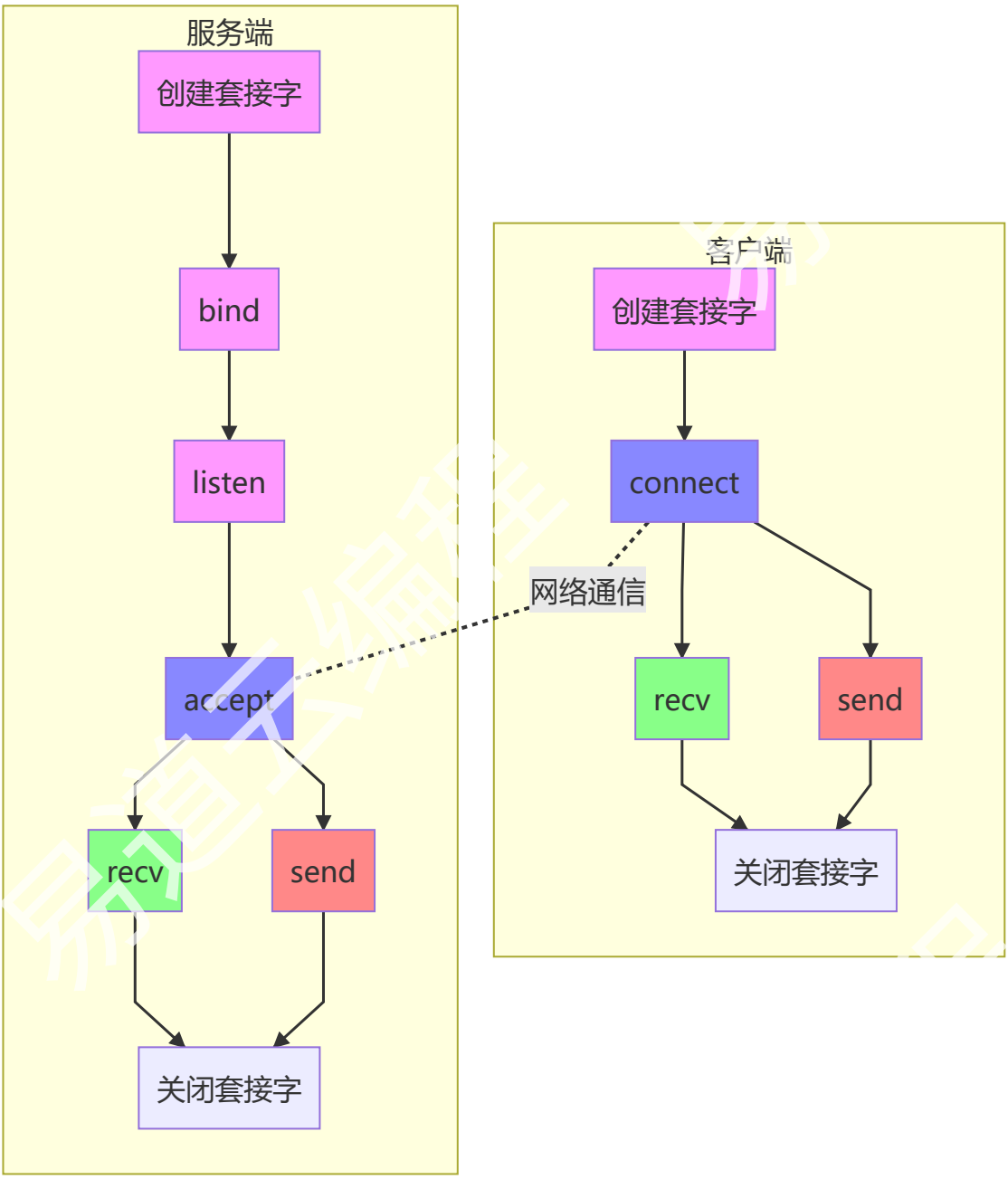
```
44      operator const epoll_data_t* ()const { return &m_data; }
45  private:
46      epoll_data_t m_data;
47  };
48
49  using EPEvents = std::vector<epoll_event>;
50
51  class CEpoll
52  {
53  public:
54      CEpoll() {
55          m_epoll = -1;
56      }
57      ~CEpoll() {
58          close();
59      }
60  public:
61      CEpoll(const CEpoll&) = delete;
62      CEpoll& operator=(const CEpoll&) = delete;
63  public:
64      operator int()const { return m_epoll; }
65  public:
66      int Create(unsigned count) {
67          if (m_epoll != -1)return -1;
68          m_epoll = epoll_create(count);
69          if (m_epoll == -1)return -2;
70          return 0;
71      }
72      //小于0表示错误 等于0表示没有事情发生 大于0表示成功拿到事件
73      ssize_t WaitEvents(EPEvents& events, int timeout = 10) {
74          if (m_epoll == -1)return -1;
75          EPEvents evs(EVENT_SIZE);
76          int ret = epoll_wait(m_epoll, evs.data(), (int)evs.size(), timeout);
```

```
77          if (ret == -1) {
78              if ((errno == EINTR) || (errno ==
EAGAIN)) {
79                  return 0;
80              }
81              return -2;
82          }
83          if (ret > (int)events.size()) {
84              events.resize(ret);
85          }
86          memcpy(events.data(), evs.data(),
sizeof(epoll_event) * ret);
87          return ret;
88      }
89      int Add(int fd, const EpollData& data =
EpollData((void*)0), uint32_t events = EPOLLIN)
90      {
91          if (m_epoll == -1)return -1;
92          epoll_event ev = { events,data };
93          int ret = epoll_ctl(m_epoll,
EPOLL_CTL_ADD, fd, &ev);
94          if (ret == -1)return -2;
95          return 0;
96      }
97      int Modify(int fd, uint32_t events, const
EpollData& data = EpollData((void*)0))
98      {
99          if (m_epoll == -1)return -1;
100         epoll_event ev = { events,data };
101         int ret = epoll_ctl(m_epoll,
EPOLL_CTL_MOD, fd, &ev);
102         if (ret == -1)return -2;
103         return 0;
104     }
105     int Del(int fd)
106     {
107         if (m_epoll == -1)return -1;
```

```
108         int ret = epoll_ctl(m_epoll,
    EPOLL_CTL_DEL, fd, NULL);
109         if (ret == -1)return -2;
110         return 0;
111     }
112     void Close() {
113         if (m_epoll != -1) {
114             int fd = m_epoll;
115             m_epoll = -1;
116             close(fd);
117         }
118     }
119
120 private:
121     int m_epoll;
122 };
```

进程间通信的实现

本地套接字的封装




```
#pragma once
#include <unistd.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string>
#include <fcntl.h>

class Buffer :public std::string
```

```
{
public:
    Buffer() :std::string() {}
    Buffer(size_t size) :std::string() { resize(size); }
    operator char* () { return (char*)c_str(); }
    operator char* () const { return (char*)c_str(); }
    operator const char* () const { return c_str(); }
};

enum SockAttr {
    SOCK_ISSERVER = 1,//是否服务器 1表示是 0表示客户端
    SOCK_ISNONBLOCK = 2,//是否阻塞 1表示非阻塞 0表示阻塞
    SOCK_ISUDP = 4,//是否为UDP 1表示udp 0表示tcp
};

class CSockParam {
public:
    CSockParam() {
        bzero(&addr_in, sizeof(addr_in));
        bzero(&addr_un, sizeof(addr_un));
        port = -1;
        attr = 0;//默认是客户端、阻塞、tcp
    }
    CSockParam(const Buffer& ip, short port, int attr) {
        this->ip = ip;
        this->port = port;
        this->attr = attr;
        addr_in.sin_family = AF_INET;
        addr_in.sin_port = port;
        addr_in.sin_addr.s_addr = inet_addr(ip);
    }
```

```
42      CSockParam(const Buffer& path, int attr) {
43          ip = path;
44          addr_un.sun_family = AF_UNIX;
45          strcpy(addr_un.sun_path, path);
46          this->attr = attr;
47      }
48      ~CSockParam() {}
49      CSockParam(const CSockParam& param) {
50          ip = param.ip;
51          port = param.port;
52          attr = param.attr;
53          memcpy(&addr_in, &param.addr_in,
    sizeof(addr_in));
54          memcpy(&addr_un, &param.addr_un,
    sizeof(addr_un));
55      }
56  public:
57      CSockParam& operator=(const CSockParam&
    param) {
58          if (this != &param) {
59              ip = param.ip;
60              port = param.port;
61              attr = param.attr;
62              memcpy(&addr_in, &param.addr_in,
    sizeof(addr_in));
63              memcpy(&addr_un, &param.addr_un,
    sizeof(addr_un));
64          }
65          return *this;
66      }
67      sockaddr* addrin() { return
    (sockaddr*)&addr_in; }
68      sockaddr* addrun() { return
    (sockaddr*)&addr_un; }
69  public:
70      //地址
71      sockaddr_in addr_in;
```

```
72      sockaddr_un addr_un;
73      //ip
74      Buffer ip;
75      //端口
76      short port;
77      //参考SockAttr
78      int attr;
79  };
80
81  class CSocketBase
82  {
83  public:
84      CSocketBase() {
85          m_socket = -1;
86          m_status = 0;//初始化未完成
87      }
88      //传递析构操作
89      virtual ~CSocketBase() {
90          Close();
91      }
92  public:
93      //初始化 服务器 套接字创建、bind、listen  客户端
    套接字创建
94      virtual int Init(const CSockParam& param) =
    0;
95      //连接 服务器 accept 客户端 connect  对于udp这里
    可以忽略
96      virtual int Link(CSocketBase** pClient =
    NULL) = 0;
97      //发送数据
98      virtual int Send(const Buffer& data) = 0;
99      //接收数据
100     virtual int Recv(Buffer& data) = 0;
101     //关闭连接
102     virtual int Close() {
103         m_status = 3;
104         if (m_socket != -1) {
```

```
105             int fd = m_socket;
106             m_socket = -1;
107             close(fd);
108         }
109     };
110 protected:
111     //套接字描述符，默认是-1
112     int m_socket;
113     //状态 0初始化未完成 1初始化完成 2连接完成 3已经关闭
114     int m_status;
115     //初始化参数
116     CSockParam m_param;
117 };
118 
119 class CLocalSocket
120     :public CSocketBase
121 {
122 public:
123     CLocalSocket() :CSocketBase() {}
124     CLocalSocket(int sock) :CSocketBase() {
125         m_socket = sock;
126     }
127     //传递析构操作
128     virtual ~CLocalSocket() {
129         Close();
130     }
131 public:
132     //初始化 服务器 套接字创建、bind、listen  客户端套接字创建
133     virtual int Init(const CSockParam& param) {
134         if (m_status != 0)return -1;
135         m_param = param;
136         int type = (m_param.attr & SOCK_ISUDP) ? SOCK_DGRAM : SOCK_STREAM;
137         if (m_socket == -1)
```

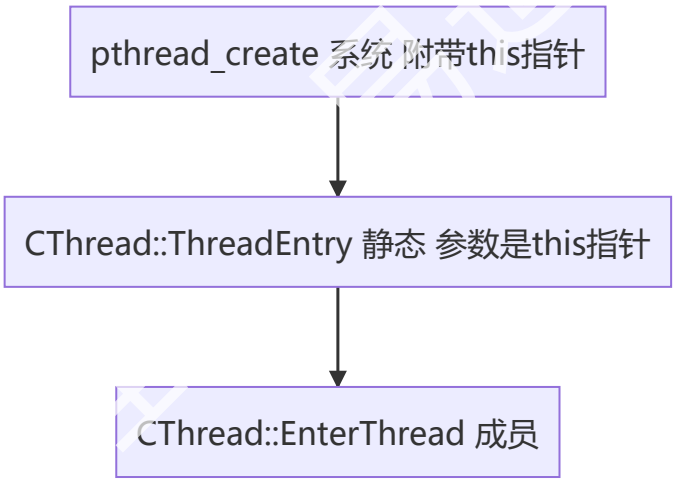
```
138             m_socket = socket(PF_LOCAL, type,
    0);
139         if (m_socket == -1)return -2;
140         int ret = 0;
141         if (m_param.attr & SOCK_ISSERVER) {
142             ret = bind(m_socket,
    m_param.addrun(), sizeof(sockaddr_un));
143             if (ret == -1) return -3;
144             ret = listen(m_socket, 32);
145             if (ret == -1)return -4;
146         }
147         if (m_param.attr & SOCK_ISNONBLOCK) {
148             int option = fcntl(m_socket,
    F_GETFL);
149             if (option == -1)return -5;
150             option |= O_NONBLOCK;
151             ret = fcntl(m_socket, F_SETFL,
    option);
152             if (ret == -1)return -6;
153         }
154         m_status = 1;
155         return 0;
156     }
157     //连接 服务器 accept 客户端 connect  对于udp这里
    可以忽略
158     virtual int Link(CSocketBase** pClient =
    NULL) {
159         if (m_status <= 0 || (m_socket ==
    -1))return -1;
160         int ret = 0;
161         if (m_param.attr & SOCK_ISSERVER) {
162             if (pClient == NULL)return -2;
163             CSockParam param;
164             socklen_t len = sizeof(sockaddr_un);
165             int fd = accept(m_socket,
    param.addrun(), &len);
166             if (fd == -1)return -3;
```

```
167             *pClient = new CLocalSocket(fd);
168             if (*pClient == NULL)return -4;
169             ret = (*pClient)->Init(param);
170             if (ret != 0) {
171                 delete (*pClient);
172                 *pClient = NULL;
173                 return -5;
174             }
175         }
176         else {
177             ret = connect(m_socket,
    m_param.addrun(), sizeof(sockaddr_un));
178             if (ret != 0)return -6;
179         }
180         m_status = 2;
181         return 0;
182     }
183     //发送数据
184     virtual int Send(const Buffer& data) {
185         if (m_status < 2 || (m_socket ==
    -1))return -1;
186         ssize_t index = 0;
187         while (index < (ssize_t)data.size()) {
188             ssize_t len = write(m_socket,
    (char*)data + index, data.size() - index);
189             if (len == 0)return -2;
190             if (len < 0)return -3;
191             index += len;
192         }
193         return 0;
194     }
195     //接收数据 大于零，表示接收成功 小于 表示失败 等于0
    表示没有收到数据，但没有错误
196     virtual int Recv(Buffer& data) {
197         if (m_status < 2 || (m_socket ==
    -1))return -1;
```

```
198         ssize_t len = read(m_socket, data,
    data.size());
199         if (len > 0) {
200             data.resize(len);
201             return (int)len;//收到数据
202         }
203         if (len < 0) {
204             if (errno == EINTR || (errno ==
    EAGAIN)) {
205                 data.clear();
206                 return 0;//没有数据收到
207             }
208             return -2;//发送错误
209         }
210         return -3;//套接字被关闭
211     }
212     //关闭连接
213     virtual int Close() {
214         return CSocketBase::Close();
215     }
216 };
```

线程的封装

静态函数到非静态函数的转换



Thread.h

```
#pragma once
#include <unistd.h>
#include <pthread.h>
#include <fcntl.h>
#include <signal.h>
#include <map>
#include "Function.h"


class CThread
{
public:
    CThread() {
        m_function = NULL;
        m_thread = 0;
        m_bpaused = false;
    }

    template<typename _FUNCTION_, typename...
_ARGS_>
    CThread(_FUNCTION_ func, _ARGS_... args)
        :m_function(new CFunction<_FUNCTION_,
_ARGS_...>(func, args...))
    {
        m_thread = 0;
        m_bpaused = false;
    }

    ~CThread() {}
public:
    CThread(const CThread&) = delete;
    CThread operator=(const CThread&) = delete;
public:
    template<typename _FUNCTION_, typename...
_ARGS_>
```

```
33      int SetThreadFunc(_FUNCTION_ func, _ARGS_... args)
34      {
35          m_function = new CFunction<_FUNCTION_, _ARGS_...>(func, args...);
36          if (m_function == NULL)return -1;
37          return 0;
38      }
39      int Start() {
40          pthread_attr_t attr;
41          int ret = 0;
42          ret = pthread_attr_init(&attr);
43          if (ret != 0)return -1;
44          ret = pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);
45          if (ret != 0)return -2;
46          ret = pthread_attr_setscope(&attr, PTHREAD_SCOPE_PROCESS);
47          if (ret != 0)return -3;
48          ret = pthread_create(&m_thread, &attr, &CThread::ThreadEntry, this);
49          if (ret != 0)return -4;
50          m_mapThread[m_thread] = this;
51          ret = pthread_attr_destroy(&attr);
52          if (ret != 0)return -5;
53          return 0;
54      }
55      int Pause() {
56          if (m_thread != 0)return -1;
57          if (m_bpaused) {
58              m_bpaused = false;
59              return 0;
60          }
61          m_bpaused = true;
62          int ret = pthread_kill(m_thread, SIGUSR1);
63          if (ret != 0) {
```

```
            m_bpaused = false;
            return -2;
        }
        return 0;
    }
    int Stop() {
        if (m_thread != 0) {
            pthread_t thread = m_thread;
            m_thread = 0;
            timespec ts;
            ts.tv_sec = 0;
            ts.tv_nsec = 100 * 1000000;//100ms
            int ret =
pthread_timedjoin_np(thread, NULL, &ts);
            if (ret == ETIMEDOUT) {
                pthread_detach(thread);
                pthread_kill(thread, SIGUSR2);
            }
        }
        return 0;
    }
    bool isValid()const { return m_thread == 0;
}
private:
    //__stdcall
    static void* ThreadEntry(void* arg) {
        CThread* thiz = (CThread*)arg;
        struct sigaction act = { 0 };
        sigemptyset(&act.sa_mask);
        act.sa_flags = SA_SIGINFO;
        act.sa_sigaction = &CThread::Sigaction;
        sigaction(SIGUSR1, &act, NULL);
        sigaction(SIGUSR2, &act, NULL);

        thiz->EnterThread();

        if (thiz->m_thread)thiz->m_thread = 0;
```

```
99          pthread_t thread = pthread_self();//不是冗余，有可能被stop函数把m_thread给清零了
100         auto it = m_mapThread.find(thread);
101         if (it != m_mapThread.end())
102             m_mapThread[thread] = NULL;
103         pthread_detach(thread);
104         pthread_exit(NULL);
105     }
106
107     static void Sigaction(int signo, siginfo_t* info, void* context)
108     {
109         if (signo == SIGUSR1) {
110             pthread_t thread = pthread_self();
111             auto it = m_mapThread.find(thread);
112             if (it != m_mapThread.end()) {
113                 if (it->second) {
114                     while (it->second->m_bpaused) {
115                         if (it->second->m_thread == 0) {
116                             pthread_exit(NULL);
117                         }
118                         usleep(1000);//1ms
119                     }
120                 }
121             }
122         }
123         else if (signo == SIGUSR2) {//线程退出
124             pthread_exit(NULL);
125         }
126     }
127
128     void EnterThread() {//__thiscall
129         if (m_function != NULL) {
130             int ret = (*m_function)();
131             if (ret != 0) {
```

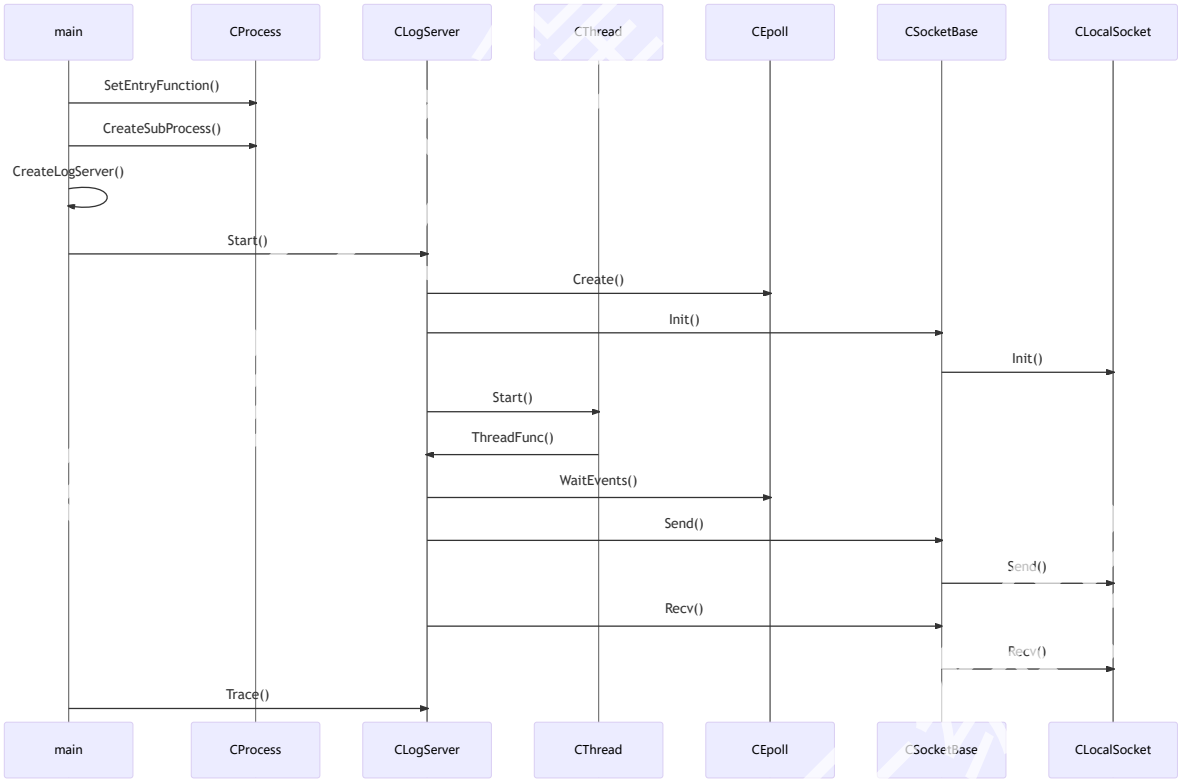
```
                printf("%s(%d):[%s]ret = %d\n", __FILE__, __LINE__, __FUNCTION__);
            }
        }
    }
private:
    CFunctionBase* m_function;
    pthread_t m_thread;
    bool m_bpaused;//true 表示暂停 false表示运行中
    static std::map<pthread_t, CThread*> m_mapThread;
};
```

Thread.cpp

```
#include "Thread.h"

std::map<pthread_t, CThread*> CThread::m_mapThread;
```

日志模块的实现

日志工作时序图




```
#pragma once
#include "Thread.h"
#include "Epoll.h"
```

```cpp
#include "Socket.h"
#include <list>
#include <sys/timeb.h>
#include <stdarg.h>
#include <sstream>
#include <sys/stat.h>

enum LogLevel {
    LOG_INFO,
    LOG_DEBUG,
    LOG_WARNING,
    LOG_ERROR,
    LOG_FATAL
};

class LogInfo {
public:
    LogInfo(
        const char* file, int line, const char* func,
        pid_t pid, pthread_t tid, int level,
        const char* fmt, ...);
    LogInfo(
        const char* file, int line, const char* func,
        pid_t pid, pthread_t tid, int level);

    LogInfo(const char* file, int line, const char* func,
        pid_t pid, pthread_t tid, int level,
        void* pData, size_t nSize);

    ~LogInfo();
    operator Buffer()const {
        return m_buf;
    }
    template<typename T>
```

```
38      LogInfo& operator<<(const T& data) {
39          std::stringstream stream;
40          stream << data;
41          m_buf += stream.str();
42          return *this;
43      }
44  private:
45      bool bAuto;//默认是false 流式日志，则为true
46      Buffer m_buf;
47  };
48
49  class CLoggerServer
50  {
51  public:
52      CLoggerServer() :
53          m_thread(&CLoggerServer::ThreadFunc,
    this)
54      {
55          m_server = NULL;
56          m_path = "./log/" + GetTimeStr() +
    ".log";
57          printf("%s(%d):[%s]path=%s\n", __FILE__,
    __LINE__, __FUNCTION__, (char*)m_path);
58      }
59      ~CLoggerServer() {
60          Close();
61      }
62  public:
63      CLoggerServer(const CLoggerServer&) =
    delete;
64      CLoggerServer& operator=(const
    CLoggerServer&) = delete;
65  public:
66      //日志服务器的启动
67      int Start() {
68          if (m_server != NULL)return -1;
69          if (access("log", W_OK | R_OK) != 0) {
```

```
70              mkdir("log", S_IRUSR | S_IWUSR |
S_IRGRP | S_IWGRP | S_IROTH);
71          }
72          m_file = fopen(m_path, "w+");
73          if (m_file == NULL)return -2;
74          int ret = m_epoll.Create(1);
75          if (ret != 0)return -3;
76          m_server = new CLocalSocket();
77          if (m_server == NULL) {
78              Close();
79              return -4;
80          }
81          ret = m_server-
>Init(CSockParam("./log/server.sock",
(int)SOCK_ISSERVER));
82          if (ret != 0) {
83              Close();
84              return -5;
85          }
86          ret = m_thread.Start();
87          if (ret != 0) {
88              Close();
89              return -6;
90          }
91          return 0;
92      }
93      int ThreadFunc() {
94          EPEvents events;
95          std::map<int, CSocketBase*> mapClients;
96          while (m_thread.isValid() && (m_epoll !=
-1) && (m_server != NULL)) {
97              ssize_t ret =
m_epoll.WaitEvents(events, 1);
98              if (ret < 0)break;
99              if (ret > 0) {
100                 ssize_t i = 0;
101                 for (; i < ret; i++) {
```

```
102                     if (events[i].events &
EPOLLERR) {
103                         break;
104                     }
105                     else if (events[i].events &
EPOLLIN) {
106                         if (events[i].data.ptr
== m_server) {
107                             CSocketBase* pClient
= NULL;
108                             int r = m_server-
>Link(&pClient);
109                             if (r < 0) continue;
110                             r =
m_epoll.Add(*pClient, EpollData((void*)pClient),
EPOLLIN | EPOLLERR);
111                             if (r < 0) {
112                                 delete pClient;
113                                 continue;
114                             }
115                             auto it =
mapClients.find(*pClient);
116                             if (it->second !=
NULL) {
117                                 delete it-
>second;
118                             }
119                             mapClients[*pClient]
= pClient;
120                         }
121                         else {
122                             CSocketBase* pClient
= (CSocketBase*)events[i].data.ptr;
123                             if (pClient != NULL)
{
124                                 Buffer data(1024
* 1024);
```

```
125                                 int r = pClient->Recv(data);
126                                 if (r <= 0) {
127                                     delete pClient;
128 
                                    mapClients[*pClient] = NULL;
129                                 }
130                                 else {
131 
                                    WriteLog(data);
132                                 }
133                             }
134                         }
135                     }
136                 }
137                 if (i != ret) {
138                     break;
139                 }
140             }
141         }
142         for (auto it = mapClients.begin(); it != mapClients.end(); it++) {
143             if (it->second) {
144                 delete it->second;
145             }
146         }
147         mapClients.clear();
148         return 0;
149     }
150     int Close() {
151         if (m_server != NULL) {
152             CSocketBase* p = m_server;
153             m_server = NULL;
154             delete p;
155         }
156         m_epoll.Close();
```

```
157         m_thread.Stop();
158         return 0;
159     }
160     //给其他非日志进程的进程和线程使用的
161     static void Trace(const LogInfo& info) {
162         static thread_local CLocalSocket client;
163         if (client == -1) {
164             int ret = 0;
165             ret =
client.Init(CSockParam("./log/server.sock", 0));
166             if (ret != 0) {
167 #ifdef _DEBUG
168                 printf("%s(%d):[%s]ret=%d\n",
__FILE__, __LINE__, __FUNCTION__, ret);
169 #endif
170                 return;
171             }
172         }
173         client.Send(info);
174     }
175     static Buffer GetTimeStr() {
176         Buffer result(128);
177         timeb tmb;
178         ftime(&tmb);
179         tm* pTm = localtime(&tmb.time);
180         int nSize = snprintf(result,
result.size(),
181             "%04d-%02d-%02d %02d-%02d-%02d
%03d",
182             pTm->tm_year + 1900, pTm->tm_mon +
1, pTm->tm_mday,
183             pTm->tm_hour, pTm->tm_min, pTm-
>tm_sec,
184             tmb.millitm
185         );
186         result.resize(nSize);
187         return result;
```

```
188     }
189 private:
190     void WriteLog(const Buffer& data) {
191         if (m_file != NULL) {
192             FILE* pFile = m_file;
193             fwrite((char*)data, 1, data.size(),
    pFile);
194             fflush(pFile);
195 #ifdef _DEBUG
196             printf("%s", (char*)data);
197 #endif
198         }
199     }
200 private:
201     CThread m_thread;
202     CEpoll m_epoll;
203     CSocketBase* m_server;
204     Buffer m_path;
205     FILE* m_file;
206 };
207
208 #ifndef TRACE
209 #define TRACEI(...)
    CLoggerServer::Trace(LogInfo(__FILE__, __LINE__,
    __FUNCTION__, getpid(), pthread_self(),
    LOG_INFO, __VA_AGRS__))
210 #define TRACED(...)
    CLoggerServer::Trace(LogInfo(__FILE__, __LINE__,
    __FUNCTION__, getpid(), pthread_self(),
    LOG_DEBUG, __VA_AGRS__))
211 #define TRACEW(...)
    CLoggerServer::Trace(LogInfo(__FILE__, __LINE__,
    __FUNCTION__, getpid(), pthread_self(),
    LOG_WARNING, __VA_AGRS__))
```

```
212 #define TRACEE(...)
    CLoggerServer::Trace(LogInfo(__FILE__, __LINE__,
    __FUNCTION__, getpid(), pthread_self(),
    LOG_ERROR, __VA_AGRS__))
213 #define TRACEF(...)
    CLoggerServer::Trace(LogInfo(__FILE__, __LINE__,
    __FUNCTION__, getpid(), pthread_self(),
    LOG_FATAL, __VA_AGRS__))
214
215 //LOGI<<"hello"<<"how are you";
216 #define LOGI LogInfo(__FILE__, __LINE__,
    __FUNCTION__, getpid(), pthread_self(),
    LOG_INFO)
217 #define LOGD LogInfo(__FILE__, __LINE__,
    __FUNCTION__, getpid(), pthread_self(),
    LOG_DEBUG)
218 #define LOGW LogInfo(__FILE__, __LINE__,
    __FUNCTION__, getpid(), pthread_self(),
    LOG_WARNING)
219 #define LOGE LogInfo(__FILE__, __LINE__,
    __FUNCTION__, getpid(), pthread_self(),
    LOG_ERROR)
220 #define LOGF LogInfo(__FILE__, __LINE__,
    __FUNCTION__, getpid(), pthread_self(),
    LOG_FATAL)
221
222 //内存导出
223 //00 01 02 65……  ; ...a……
224 //
225 #define DUMPI(data, size) LogInfo(__FILE__,
    __LINE__, __FUNCTION__, getpid(),
    pthread_self(), LOG_INFO, data, size)
226 #define DUMPD(data, size) LogInfo(__FILE__,
    __LINE__, __FUNCTION__, getpid(),
    pthread_self(), LOG_DEBUG, data, size)
```

```
227 #define DUMPW(data, size) LogInfo(__FILE__,
    __LINE__, __FUNCTION__, getpid(),
    pthread_self(), LOG_WARNING, data, size)
228 #define DUMPE(data, size) LogInfo(__FILE__,
    __LINE__, __FUNCTION__, getpid(),
    pthread_self(), LOG_ERROR, data, size)
229 #define DUMPF(data, size) LogInfo(__FILE__,
    __LINE__, __FUNCTION__, getpid(),
    pthread_self(), LOG_FATAL, data, size)
230 #endif
231
```

日志模块的测试

主线程中调用

子线程中调用

信号触发时调用

日志中包含整数、小数、字符、字符串

日志中包含英文、中文、标点符号

```
1 int LogTest()
2 {
3     char buffer[] = "hello edoyun! 冯老师";
4     usleep(1000 * 100);
5     TRACEI("here is log %d %c %f %g %s 哈哈 嘻嘻
  易道云", 10, 'A', 1.0f, 2.0, buffer);
6     DUMPD((void*)buffer, (size_t)sizeof(buffer));
7     LOGE << 100 << " " << 'S' << " " << 0.12345f
  << " " << 1.23456789 << " " << buffer << " 易道云
  编程";
8     return 0;
9 }
```

```
10
11  int main()
12  {
13      //CProcess::SwitchDeamon();
14      CProcess proclog, procclients;
15      printf("%s(%d):<%s> pid=%d\n", __FILE__,
    __LINE__, __FUNCTION__, getpid());
16      proclog.SetEntryFunction(CreateLogServer,
    &proclog);
17      int ret = proclog.CreateSubProcess();
18      if (ret != 0) {
19          printf("%s(%d):<%s> pid=%d\n", __FILE__,
    __LINE__, __FUNCTION__, getpid());
20          return -1;
21      }
22      LogTest();
23      printf("%s(%d):<%s> pid=%d\n", __FILE__,
    __LINE__, __FUNCTION__, getpid());
24      CThread thread(LogTest);
25      thread.Start();
26
    procclients.SetEntryFunction(CreateClientServer,
    &procclients);
27      ret = procclients.CreateSubProcess();
28      if (ret != 0) {
29          printf("%s(%d):<%s> pid=%d\n", __FILE__,
    __LINE__, __FUNCTION__, getpid());
30          return -2;
31      }
32      printf("%s(%d):<%s> pid=%d\n", __FILE__,
    __LINE__, __FUNCTION__, getpid());
33      usleep(100 * 000);
34      int fd = open("./test.txt", O_RDWR | O_CREAT
    | O_APPEND);
35      printf("%s(%d):<%s> fd=%d\n", __FILE__,
    __LINE__, __FUNCTION__, fd);
36      if (fd == -1)return -3;
```

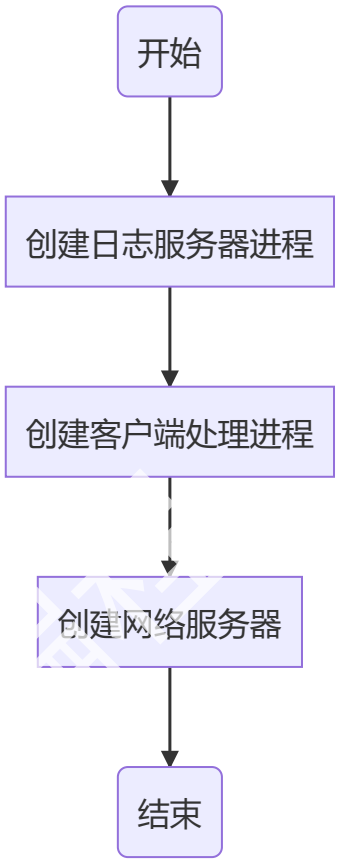
```
37      ret = procclients.SendFD(fd);
38      printf("%s(%d):<%s> ret=%d\n", __FILE__,
    __LINE__, __FUNCTION__, ret);
39      if (ret != 0)printf("errno:%d msg:%s\n",
    errno, strerror(errno));
40      write(fd, "edoyun", 6);
41      close(fd);
42      proclog.SendFD(-1);
43      (void)getchar();
44      return 0;
45  }
```

主模块的设计

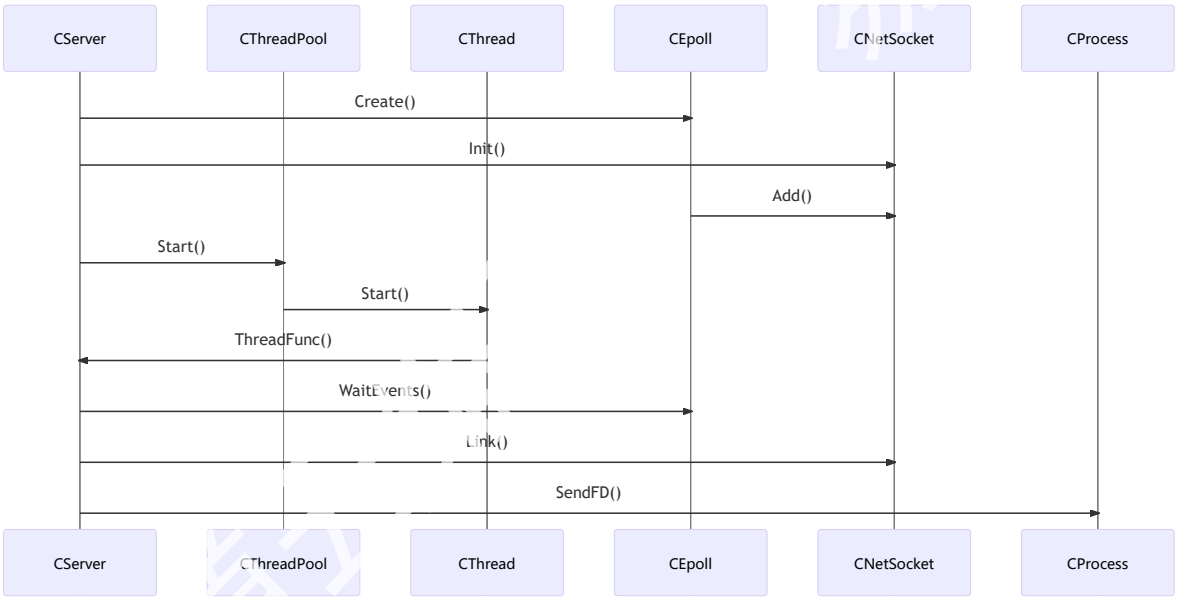
主模块主要就是**客户端的接入**，然后分发客户端到客户端处理进程去处理

所以其逻辑比较清晰（服务器每个模块的逻辑，越简单越好）

下图展示了程序的流程：

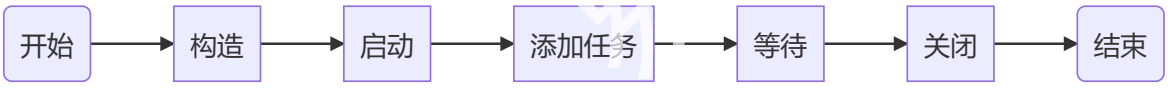


网络服务器逻辑则要复杂一点



服务器的线程函数主要是接收客户端，然后发送到客户端处理进程进行后续处理。

线程池的设计



接口设计：

```
#pragma once
#include "Epoll.h"
```

```cpp
#include "Thread.h"
#include "Function.h"

class CThreadPool
{
public:
    CThreadPool() ;
    ~CThreadPool() {
        Close();
    }
    CThreadPool(const CThreadPool&) = delete;
    CThreadPool& operator=(const CThreadPool&) = delete;
public:
    int Start(unsigned count);
    void Close();
    template<typename _FUNCTION_, typename... _ARGS_>
    int AddTask(_FUNCTION_ func, _ARGS_... args);
private:
    int TaskDispatch();
private:
    CEpoll m_epoll;
    std::vector<CThread*> m_threads;
    CSocketBase* m_server;
    Buffer m_path;
};
```

线程池的实现

```cpp
#pragma once
#include "Epoll.h"
#include "Thread.h"
#include "Function.h"
#include "Socket.h"
```

```
6
7  class CThreadPool
8  {
9  public:
10     CThreadPool() {
11         m_server = NULL;
12         timespec tp = { 0,0 };
13         clock_gettime(CLOCK_REALTIME, &tp);
14         char* buf = NULL;
15         asprintf(&buf, "%d.%d.sock", tp.tv_sec %
   100000, tp.tv_nsec % 1000000);
16         if (buf != NULL) {
17             m_path = buf;
18             free(buf);
19         }//有问题的话，在start接口里面判断m_path来解
   决问题。
20         usleep(1);
21     }
22     ~CThreadPool() {
23         Close();
24     }
25     CThreadPool(const CThreadPool&) = delete;
26     CThreadPool& operator=(const CThreadPool&) =
   delete;
27 public:
28     int Start(unsigned count) {
29         int ret = 0;
30         if (m_server != NULL)return -1;//已经初始
   化了
31         if (m_path.size() == 0)return -2;//构造函
   数失败！！！
32         m_server = new CLocalSocket();
33         if (m_server == NULL)return -3;
34         ret = m_server->Init(CSockParam(m_path,
   SOCK_ISSERVER));
35         if (ret != 0)return -4;
36         ret = m_epoll.Create(count);
```

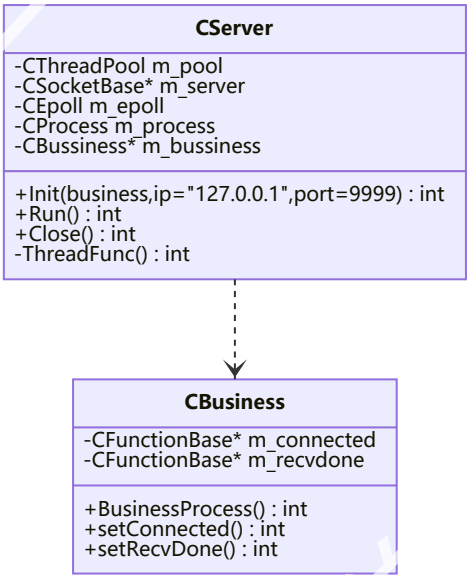
```
37          if (ret != 0)return -5;
38          ret = m_epoll.Add(*m_server,
    EpollData((void*)m_server));
39          if (ret != 0)return -6;
40          m_threads.resize(count);
41          for (unsigned i = 0; i < count; i++) {
42              m_threads[i] = new
    CThread(&CThreadPool::TaskDispatch, this);
43              if (m_threads[i] == NULL)return -7;
44              ret = m_threads[i]->Start();
45              if (ret != 0)return -8;
46          }
47          return 0;
48      }
49      void Close() {
50          m_epoll.Close();
51          if (m_server) {
52              CSocketBase* p = m_server;
53              m_server = NULL;
54              delete p;
55          }
56          for (auto thread : m_threads)
57          {
58              if (thread)delete thread;
59          }
60          m_threads.clear();
61          unlink(m_path);
62      }
63      template<typename _FUNCTION_, typename...
    _ARGS_>
64      int AddTask(_FUNCTION_ func, _ARGS_... args)
    {
65          static thread_local CLocalSocket client;
66          int ret = 0;
67          if (client == -1) {
68              ret = client.Init(CSockParam(m_path,
    0));
```

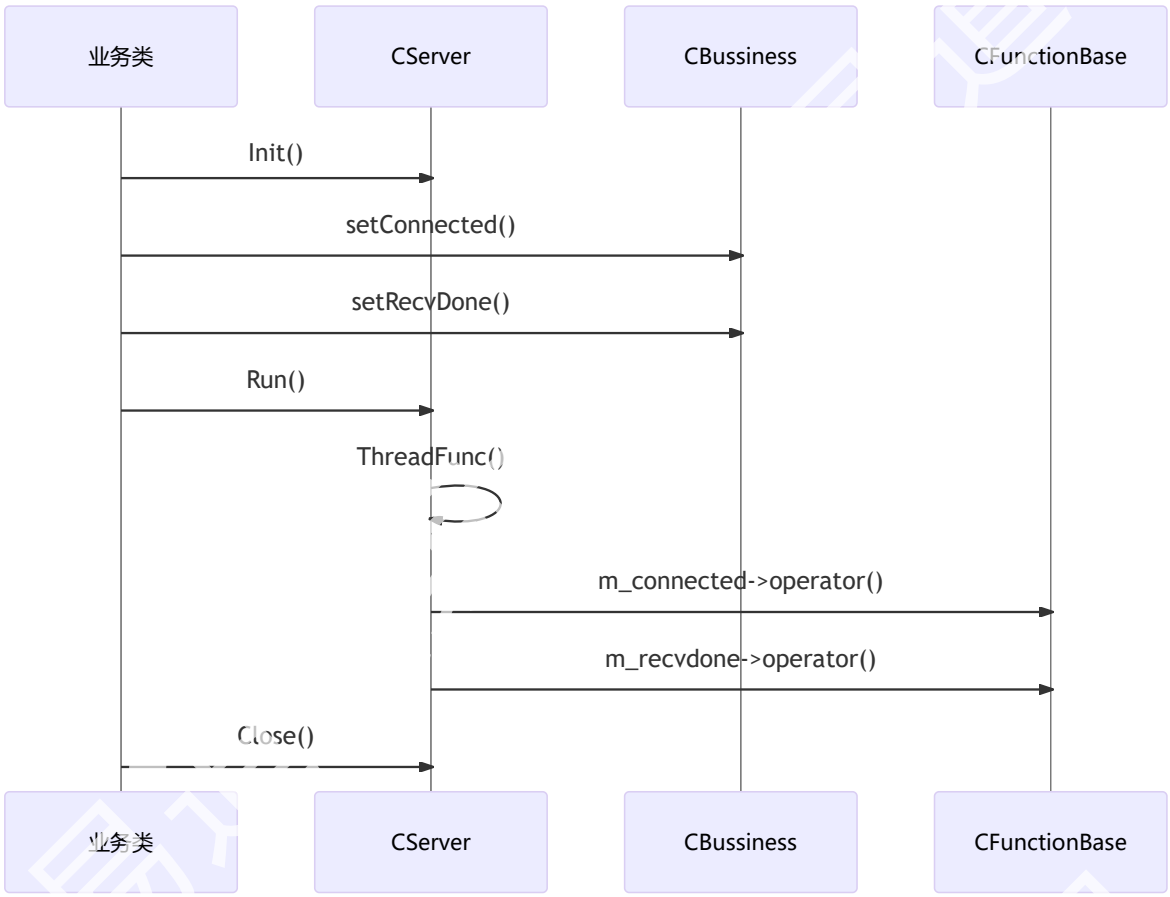
```
69              if (ret != 0)return -1;
70              ret = client.Link();
71              if (ret != 0)return -2;
72          }
73          CFunctionBase* base = new CFunction<
_FUNCTION_, _ARGS_...>(func, args...);
74          if (base == NULL)return -3;
75          Buffer data(sizeof(base));
76          memcpy(data, &base, sizeof(base));
77          ret = client.Send(data);
78          if (ret != 0) {
79              delete base;
80              return -4;
81          }
82          return 0;
83      }
84  private:
85      int TaskDispatch() {
86          while (m_epoll != -1) {
87              EPEvents events;
88              int ret = 0;
89              ssize_t esize =
m_epoll.WaitEvents(events);
90              if (esize > 0) {
91                  for (ssize_t i = 0; i < esize;
i++) {
92                      if (events[i].events &
EPOLLIN) {
93                          CSocketBase* pClient =
NULL;
94                          if (events[i].data.ptr
== m_server) {//客户端请求连接
95
96                              ret = m_server-
>Link(&pClient);
97                              if (ret !=
0)continue;
```

```
98                              ret =
m_epoll.Add(*pClient,
EpollData((void*)pClient));
99                              if (ret != 0) {
100                                 delete pClient;
101                                 continue;
102                             }
103                         }
104                         else {//客户端的数据来了
105                             pClient =
(CSocketBase*)events[i].data.ptr;
106                             if (pClient) {
107                                 CFunctionBase*
base = NULL;
108                                 Buffer
data(sizeof(base));
109                                 ret = pClient-
>Recv(data);
110                                 if (ret <= 0) {
111
m_epoll.Del(*pClient);
112                                     delete
pClient;
113                                     continue;
114                                 }
115                                 memcpy(&base,
(char*)data, sizeof(base));
116                                 if (base !=
NULL) {
117                                     (*base)();
118                                     delete base;
119                                 }
120                             }
121                         }
122                     }
123                 }
124             }
```

```
125         }
126         return 0;
127     }
128 private:
129     CEpoll m_epoll;
130     std::vector<CThread*> m_threads;
131     CSocketBase* m_server;
132     Buffer m_path;
133 };
```

主模块的实现





CServer.h

```
#pragma once
#include "Socket.h"
#include "Epoll.h"
#include "ThreadPool.h"
#include "Process.h"
class CBusiness
{
public:
    virtual int BusinessProcess() = 0;
    template<typename _FUNCTION_, typename...
_ARGS_>
    int setConnectedCallback(_FUNCTION_ func,
_ARGS_... args) {
```

```
12          m_connectedcallback = new CFunction<
_FUNCTION_, _ARGS_...>(func, args...);
13          if (m_connectedcallback == NULL)return
-1;
14          return 0;
15      }
16      template<typename _FUNCTION_, typename...
_ARGS_>
17      int setRecvCallback(_FUNCTION_ func,
_ARGS_... args) {
18          m_recvcallback = new CFunction<
_FUNCTION_, _ARGS_...>(func, args...);
19          if (m_recvcallback == NULL)return -1;
20          return 0;
21      }
22  private:
23      CFunctionBase* m_connectedcallback;
24      CFunctionBase* m_recvcallback;
25  };
26
27  class CServer
28  {
29  public:
30      CServer();
31      ~CServer() { Close(); }
32      CServer(const CServer&) = delete;
33      CServer& operator=(const CServer&) = delete;
34  public:
35      int Init(CBusiness* business, const Buffer&
ip = "127.0.0.1", short port = 9999);
36      int Run();
37      int Close();
38  private:
39      int ThreadFunc();
40  private:
41      CThreadPool m_pool;
42      CSocketBase* m_server;
```

```
43      CEpoll m_epoll;
44      CProcess m_process;
45      CBusiness* m_business;//业务模块 需要我们手动
    delete
46  };
47
48
```

CServer.cpp

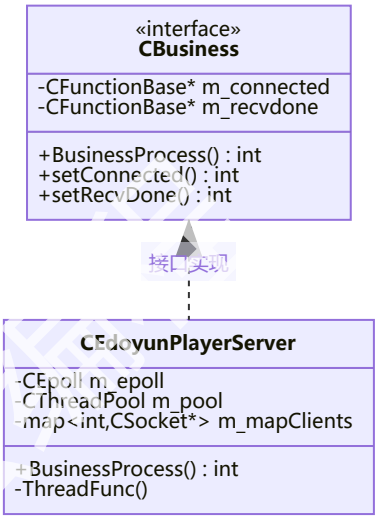
```
1   #include "CServer.h"
2   #include "Logger.h"
3
4   CServer::CServer()
5   {
6       m_server = NULL;
7       m_business = NULL;
8   }
9
10  int CServer::Init(CBusiness* business, const
    Buffer& ip, short port)
11  {
12      int ret = 0;
13      if (business == NULL)return -1;
14      m_business = business;
15      ret =
    m_process.SetEntryFunction(&CBusiness::BusinessPr
    ocess, m_business);
16      if (ret != 0)return -2;
17      ret = m_process.CreateSubProcess();
18      if (ret != 0)return -3;
19      ret = m_pool.Start(2);
20      if (ret != 0)return -4;
21      ret = m_epoll.Create(2);
22      if (ret != 0)return -5;
23      m_server = new CSocket();
24      if (m_server == NULL)return -6;
```

```
25      ret = m_server->Init(CSockParam(ip, port,
    SOCK_ISSERVER | SOCK_ISIP));
26      if (ret != 0)return -7;
27      ret = m_epoll.Add(*m_server,
    EpollData((void*)m_server));
28      if (ret != 0)return -8;
29      for (size_t i = 0; i < m_pool.Size(); i++) {
30          ret =
    m_pool.AddTask(&CServer::ThreadFunc, this);
31          if (ret != 0)return -9;
32      }
33      return 0;
34  }
35
36  int CServer::Run()
37  {
38      while (m_server != NULL) {
39          usleep(10);
40      }
41      return 0;
42  }
43
44  int CServer::Close()
45  {
46      if (m_server) {
47          CSocketBase* sock = m_server;
48          m_server = NULL;
49          m_epoll.Del(*sock);
50          delete sock;
51      }
52      m_epoll.Close();
53      m_process.SendFD(-1);
54      m_pool.Close();
55      return 0;
56  }
57
58  int CServer::ThreadFunc()
```

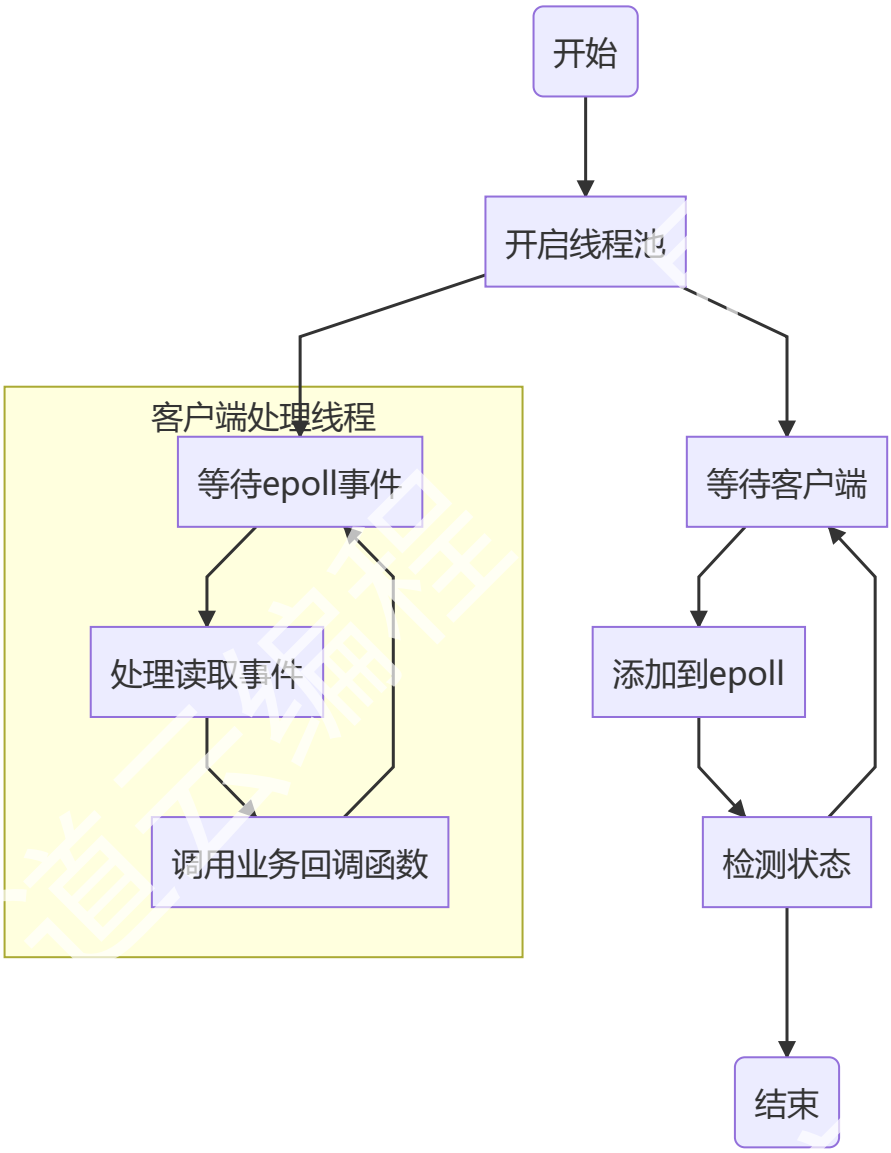
```
59 {
60     int ret = 0;
61     EPEvents events;
62     while ((m_epoll != -1) && (m_server != NULL))
{
63         ssize_t size =
m_epoll.WaitEvents(events);
64         if (size < 0)break;
65         if (size > 0) {
66             for (ssize_t i = 0; i < size; i++)
67             {
68                 if (events[i].events & EPOLLERR)
{
69                     break;
70                 }
71                 else if (events[i].events &
EPOLLIN) {
72                     if (m_server) {
73                         CSocketBase* pClient =
NULL;
74                         ret = m_server-
>Link(&pClient);
75                         if (ret != 0)continue;
76                         ret =
m_process.SendFD(*pClient);
77                         delete pClient;
78                         if (ret != 0) {
79                             TRACEE("send client
%d failed!", (int)*pClient);
80                             continue;
81                         }
82                     }
83                 }
84             }
85         }
86     }
87     return 0;
```

```
88 }
89
```

客户端处理模块的设计



基本流程图



客户端处理模块的实现

CEdoyunPlayerServer.h

```
#pragma once
#include "Logger.h"
#include "CServer.h"
#include <map>
/*
* 1. 客户端的地址问题
* 2. 连接回调的参数问题
* 3. 接收回调的参数问题
*/
```

```
#define ERR_RETURN(ret, err) if(ret!=0) {TRACEE("ret= %d errno = %d msg = [%s]", ret, errno, strerror(errno));return err;}

#define WARN_CONTINUE(ret) if(ret!=0) {TRACEW("ret= %d errno = %d msg = [%s]", ret, errno, strerror(errno));continue;}

class CEdoyunPlayerServer :
    public CBusiness
{
public:
    CEdoyunPlayerServer(unsigned count) :CBusiness() {
        m_count = count;
    }
    ~CEdoyunPlayerServer() {
        m_epoll.Close();
        m_pool.Close();
        for (auto it : m_mapClients) {
            if (it.second) {
                delete it.second;
            }
        }
        m_mapClients.clear();
    }
    virtual int BusinessProcess(CProcess* proc) {
        int ret = 0;
        ret = m_epoll.Create(m_count);
        ERR_RETURN(ret, -1);
        ret = m_pool.Start(m_count);
        ERR_RETURN(ret, -2);
        for (unsigned i = 0; i < m_count; i++) {
            ret = m_pool.AddTask(&CEdoyunPlayerServer::ThreadFunc, this);
            ERR_RETURN(ret, -3);
```

```
        }
        int sock = 0;

setRecvCallback(&CEdoyunPlayerServer::RecvDone,
this, std::placeholders::_1,
std::placeholders::_2);

setConnectedCallback(&CEdoyunPlayerServer::Connec
tedDone, this, std::placeholders::_1);
        while (m_epoll != -1) {
            ret = proc->RecvFD(sock);
            if (ret < 0 || (sock == 0))break;
            CSocketBase* pClient = new
CSocket(sock);
            if (pClient == NULL)continue;
            ret = m_epoll.Add(sock,
EpollData((void*)pClient));
            if (m_connectedcallback) {
                (*m_connectedcallback)(pClient);
            }
            WARN_CONTINUE(ret);
        }
        return 0;
    }
private:
    int ConnectedDone(CSocketBase* pClient) {
        return 0;
    }
    int RecvDone(CSocketBase* pClient, const
Buffer& data) {
        return 0;
    }
private:
    int ThreadFunc() {
        int ret = 0;
        EPEvents events;
        while (m_epoll != -1) {
```

```
69              ssize_t size =
m_epoll.WaitEvents(events);
70              if (size < 0)break;
71              if (size > 0) {
72                  for (ssize_t i = 0; i < size;
i++)
73                  {
74                      if (events[i].events &
EPOLLERR) {
75                          break;
76                      }
77                      else if (events[i].events &
EPOLLIN) {
78                          CSocketBase* pClient =
(CSocketBase*)events[i].data.ptr;
79                          if (pClient) {
80                              Buffer data;
81                              ret = pClient-
>Recv(data);
82                              WARN_CONTINUE(ret);
83                              if (m_recvcallback) {
84                                  (*m_recvcallback)
(pClient, data);
85                              }
86                          }
87                      }
88                  }
89              }
90          }
91          return 0;
92      }
93  private:
94      CEpoll m_epoll;
95      std::map<int, CSocketBase*> m_mapClients;
96      CThreadPool m_pool;
97      unsigned m_count;
98  };
```

Function.h

```
#pragma once
#include <unistd.h>
#include <sys/types.h>
#include <functional>

class CSocketBase;
class Buffer;

class CFunctionBase
{
public:
    virtual ~CFunctionBase() {}
    virtual int operator()() { return 0; }
    virtual int operator()(CSocketBase*) { return 0; }
    virtual int operator()(CSocketBase*, const Buffer&) { return 0; }
};

template<typename _FUNCTION_, typename... _ARGS_>
class CFunction :public CFunctionBase
{
public:
    CFunction(_FUNCTION_ func, _ARGS_... args)
        :m_binder(std::forward<_FUNCTION_>(func), std::forward<_ARGS_>(args)...)
    {}
    virtual ~CFunction() {}
    virtual int operator()() {
        return m_binder();
    }

    typename std::_Bindres_helper<int, _FUNCTION_, _ARGS_...>::type m_binder;
};
```

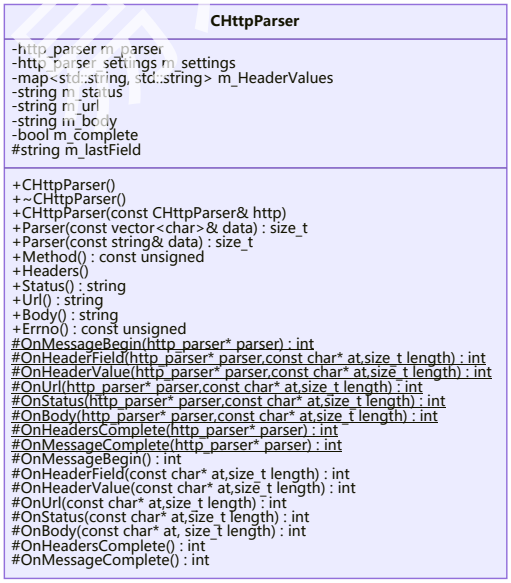
```
32
33  template<typename _FUNCTION_, typename... _ARGS_>
34  class CConnectedFunction :public CFunctionBase
35  {
36  public:
37      CConnectedFunction(_FUNCTION_ func, _ARGS_...
    args)
38          :m_binder(std::forward<_FUNCTION_>(func),
    std::forward<_ARGS_>(args)...)
39      {}
40      virtual ~CConnectedFunction() {}
41
42      virtual int operator()(CSocketBase* pClient)
    {
43          return m_binder(pClient);
44      }
45
46      typename std::_Bindres_helper<int,
    _FUNCTION_, _ARGS_...>::type m_binder;
47  };
48
49  template<typename _FUNCTION_, typename... _ARGS_>
50  class CRecvFunction :public CFunctionBase
51  {
52  public:
53      CRecvFunction(_FUNCTION_ func, _ARGS_...
    args)
54          :m_binder(std::forward<_FUNCTION_>(func),
    std::forward<_ARGS_>(args)...)
55      {}
56      virtual ~CRecvFunction() {}
57
58      virtual int operator()(CSocketBase* pClient,
    const Buffer& data) {
59          return m_binder(pClient, data);
60      }
61
```

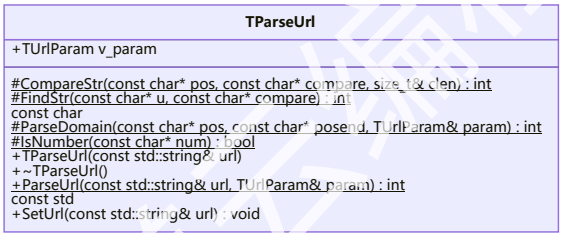
```
62      typename std::_Bindres_helper<int,
    _FUNCTION_, _ARGS_...>::type m_binder;
63  };
```

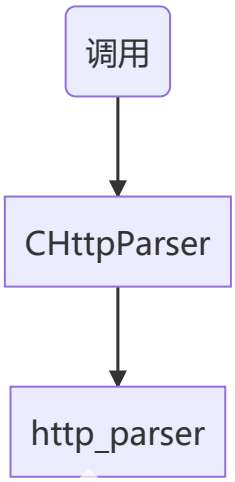
HTTP模块的设计

封装的作用

- 降低使用成本
- 对外屏蔽细节（**低耦合**）
- 增加可以移植性
- 与更多同类数据关联（**高内聚**）







HTTP模块的实现

HttpParser.h

```
#pragma once
#include "Socket.h"
#include "http_parser.h"
#include <map>

class CHttpParser
{
private:
    http_parser m_parser;
    http_parser_settings m_settings;
    std::map<Buffer, Buffer> m_HeaderValues;
    Buffer m_status;
    Buffer m_url;
    Buffer m_body;
    bool m_complete;
```

```
    Buffer m_lastField;
public:
    CHttpParser();
    ~CHttpParser();
    CHttpParser(const CHttpParser& http);
    CHttpParser& operator=(const CHttpParser&
http);
public:
    size_t Parser(const Buffer& data);
    //GET POST ... 参考http_parser.h
HTTP_METHOD_MAP宏
    unsigned Method() const { return
m_parser.method; }
    const std::map<Buffer, Buffer>& Headers() {
return m_HeaderValues; }
    const Buffer& Status() const { return
m_status; }
    const Buffer& Url() const { return m_url; }
    const Buffer& Body() const { return m_body; }
    unsigned Errno() const { return
m_parser.http_errno; }
protected:
    static int OnMessageBegin(http_parser*
parser);
    static int OnUrl(http_parser* parser, const
char* at, size_t length);
    static int OnStatus(http_parser* parser,
const char* at, size_t length);
    static int OnHeaderField(http_parser* parser,
const char* at, size_t length);
    static int OnHeaderValue(http_parser* parser,
const char* at, size_t length);
    static int OnHeadersComplete(http_parser*
parser);
    static int OnBody(http_parser* parser, const
char* at, size_t length);
```

```
39      static int OnMessageComplete(http_parser*
    parser);
40      int OnMessageBegin();
41      int OnUrl(const char* at, size_t length);
42      int OnStatus(const char* at, size_t length);
43      int OnHeaderField(const char* at, size_t
    length);
44      int OnHeaderValue(const char* at, size_t
    length);
45      int OnHeadersComplete();
46      int OnBody(const char* at, size_t length);
47      int OnMessageComplete();
48  };
49
50  class UrlParser
51  {
52  public:
53      UrlParser(const Buffer& url);
54      ~UrlParser() {}
55      int Parser();
56      Buffer operator[](const Buffer& name)const;
57      Buffer Protocol()const { return m_protocol; }
58      Buffer Host()const { return m_host; }
59      //默认返回80
60      int Port()const { return m_port; }
61      void SetUrl(const Buffer& url);
62  private:
63      Buffer m_url;
64      Buffer m_protocol;
65      Buffer m_host;
66      Buffer m_uri;
67      int m_port;
68      std::map<Buffer, Buffer> m_values;
69  };
```

HttpParser.cpp

```cpp
#include "HttpParser.h"

CHttpParser::CHttpParser()
{
    m_complete = false;
    memset(&m_parser, 0, sizeof(m_parser));
    m_parser.data = this;
    http_parser_init(&m_parser, HTTP_REQUEST);
    memset(&m_settings, 0, sizeof(m_settings));
    m_settings.on_message_begin =
&CHttpParser::OnMessageBegin;
    m_settings.on_url = &CHttpParser::OnUrl;
    m_settings.on_status =
&CHttpParser::OnStatus;
    m_settings.on_header_field =
&CHttpParser::OnHeaderField;
    m_settings.on_header_value =
&CHttpParser::OnHeaderValue;
    m_settings.on_headers_complete =
&CHttpParser::OnHeadersComplete;
    m_settings.on_body = &CHttpParser::OnBody;
    m_settings.on_message_complete =
&CHttpParser::OnMessageComplete;
}

CHttpParser::~CHttpParser()
{}

CHttpParser::CHttpParser(const CHttpParser&
http)
{
    memcpy(&m_parser, &http.m_parser,
sizeof(m_parser));
    m_parser.data = this;
    memcpy(&m_settings, &http.m_settings,
sizeof(m_settings));
    m_status = http.m_status;
```

```
29      m_url = http.m_url;
30      m_body = http.m_body;
31      m_complete = http.m_complete;
32      m_lastField = http.m_lastField;
33  }
34
35  CHttpParser& CHttpParser::operator=(const
    CHttpParser& http)
36  {
37      if (this != &http) {
38          memcpy(&m_parser, &http.m_parser,
    sizeof(m_parser));
39          m_parser.data = this;
40          memcpy(&m_settings, &http.m_settings,
    sizeof(m_settings));
41          m_status = http.m_status;
42          m_url = http.m_url;
43          m_body = http.m_body;
44          m_complete = http.m_complete;
45          m_lastField = http.m_lastField;
46      }
47      return *this;
48  }
49
50  size_t CHttpParser::Parser(const Buffer& data)
51  {
52      m_complete = false;
53      size_t ret = http_parser_execute(
54          &m_parser, &m_settings, data,
    data.size());
55      if (m_complete == false) {
56          m_parser.http_errno = 0x7F;
57          return 0;
58      }
59      return ret;
60  }
61
```

```
62 int CHttpParser::OnMessageBegin(http_parser*
   parser)
63 {
64     return ((CHttpParser*)parser->data)-
   >OnMessageBegin();
65 }
66
67 int CHttpParser::OnUrl(http_parser* parser,
   const char* at, size_t length)
68 {
69     return ((CHttpParser*)parser->data)-
   >OnUrl(at, length);
70 }
71
72 int CHttpParser::OnStatus(http_parser* parser,
   const char* at, size_t length)
73 {
74     return ((CHttpParser*)parser->data)-
   >OnStatus(at, length);
75 }
76
77 int CHttpParser::OnHeaderField(http_parser*
   parser, const char* at, size_t length)
78 {
79     return ((CHttpParser*)parser->data)-
   >OnHeaderField(at, length);
80 }
81
82 int CHttpParser::OnHeaderValue(http_parser*
   parser, const char* at, size_t length)
83 {
84     return ((CHttpParser*)parser->data)-
   >OnHeaderValue(at, length);
85 }
86
87 int CHttpParser::OnHeadersComplete(http_parser*
   parser)
```

```
88  {
89      return ((CHttpParser*)parser->data)-
    >OnHeadersComplete();
90  }
91
92  int CHttpParser::OnBody(http_parser* parser,
    const char* at, size_t length)
93  {
94      return ((CHttpParser*)parser->data)-
    >OnBody(at, length);
95  }
96
97  int CHttpParser::OnMessageComplete(http_parser*
    parser)
98  {
99      return ((CHttpParser*)parser->data)-
    >OnMessageComplete();
100 }
101
102 int CHttpParser::OnMessageBegin()
103 {
104     return 0;
105 }
106
107 int CHttpParser::OnUrl(const char* at, size_t
    length)
108 {
109     m_url = Buffer(at, length);
110     return 0;
111 }
112
113 int CHttpParser::OnStatus(const char* at, size_t
    length)
114 {
115     m_status = Buffer(at, length);
116     return 0;
117 }
```

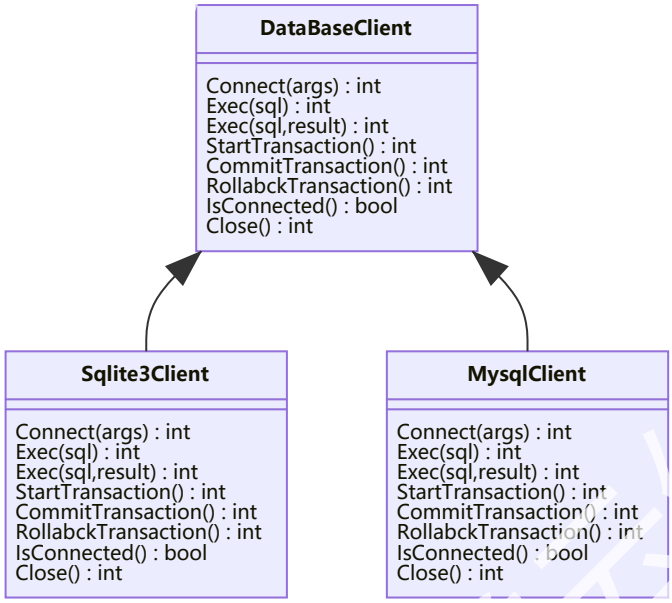
```
118
119 int CHttpParser::OnHeaderField(const char* at,
    size_t length)
120 {
121     m_lastField = Buffer(at, length);
122     return 0;
123 }
124
125 int CHttpParser::OnHeaderValue(const char* at,
    size_t length)
126 {
127     m_HeaderValues[m_lastField] = Buffer(at,
    length);
128     return 0;
129 }
130
131 int CHttpParser::OnHeadersComplete()
132 {
133     return 0;
134 }
135
136 int CHttpParser::OnBody(const char* at, size_t
    length)
137 {
138     m_body = Buffer(at, length);
139     return 0;
140 }
141
142 int CHttpParser::OnMessageComplete()
143 {
144     m_complete = true;
145     return 0;
146 }
147
148 UrlParser::UrlParser(const Buffer& url)
149 {
150     m_url = url;
```

```
}

int UrlParser::Parser()
{
    //分三步：协议、域名和端口、uri、键值对
    //解析协议
    const char* pos = m_url;
    const char* target = strstr(pos, "://");
    if (target == NULL)return -1;
    m_protocol = Buffer(pos, target);
    //解析域名和端口
    pos = target + 3;
    target = strchr(pos, '/');
    if (target == NULL) {
        if (m_protocol.size() + 3 >= m_url.size())
            return -2;
        m_host = pos;
        return 0;
    }
    Buffer value = Buffer(pos, target);
    if (value.size() == 0)return -3;
    target = strchr(value, ':');
    if (target != NULL) {
        m_host = Buffer(value, target);
        m_port = atoi(Buffer(target + 1, (char*)value + value.size()));
    }
    else {
        m_host = value;
    }
    pos = strchr(pos, '/');
    //解析uri
    target = strchr(pos, '?');
    if (target == NULL) {
        m_uri = pos;
        return 0;
```

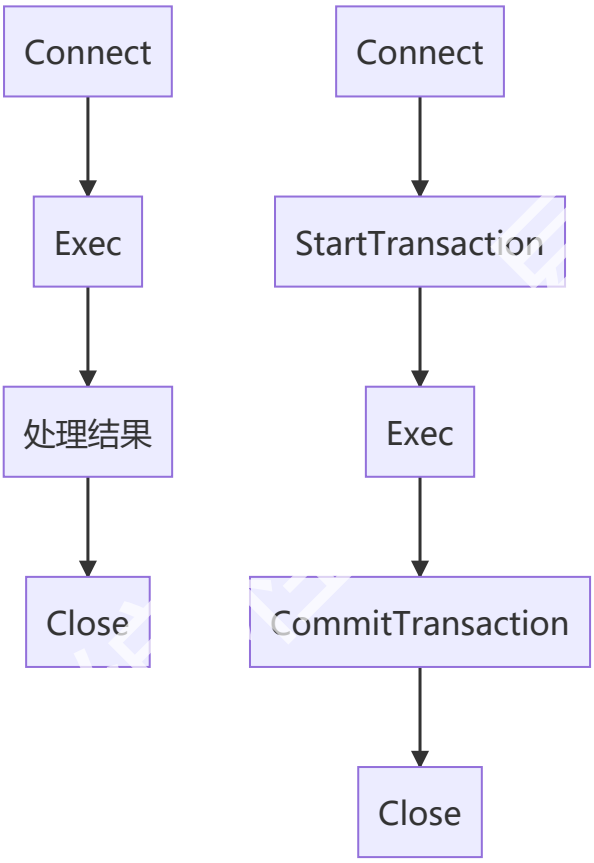
```
186     }
187     else {
188         m_uri = Buffer(pos, target);
189         //解析key和value
190         pos = target + 1;
191         const char* t = NULL;
192         do {
193             target = strchr(pos, '&');
194             if (target == NULL) {
195                 t = strchr(pos, '=');
196                 if (t == NULL)return -4;
197                 m_values[Buffer(pos, t)] =
    Buffer(t + 1);
198             }
199             else {
200                 Buffer kv(pos, target);
201                 t = strchr(kv, '=');
202                 if (t == NULL)return -5;
203                 m_values[Buffer(kv, t)] =
    Buffer(t + 1, kv + kv.size());
204                 pos = target + 1;
205             }
206         } while (target != NULL);
207     }
208
209     return 0;
210 }
211
212 Buffer UrlParser::operator[](const Buffer& name)
    const
213 {
214     auto it = m_values.find(name);
215     if (it == m_values.end())return Buffer();
216     return it->second;
217 }
218
219 void UrlParser::SetUrl(const Buffer& url)
```

```
220 {
221     m_url = url;
222     m_protocol = "";
223     m_host = "";
224     m_port = 80;
225     m_values.clear();
226 }
227
```

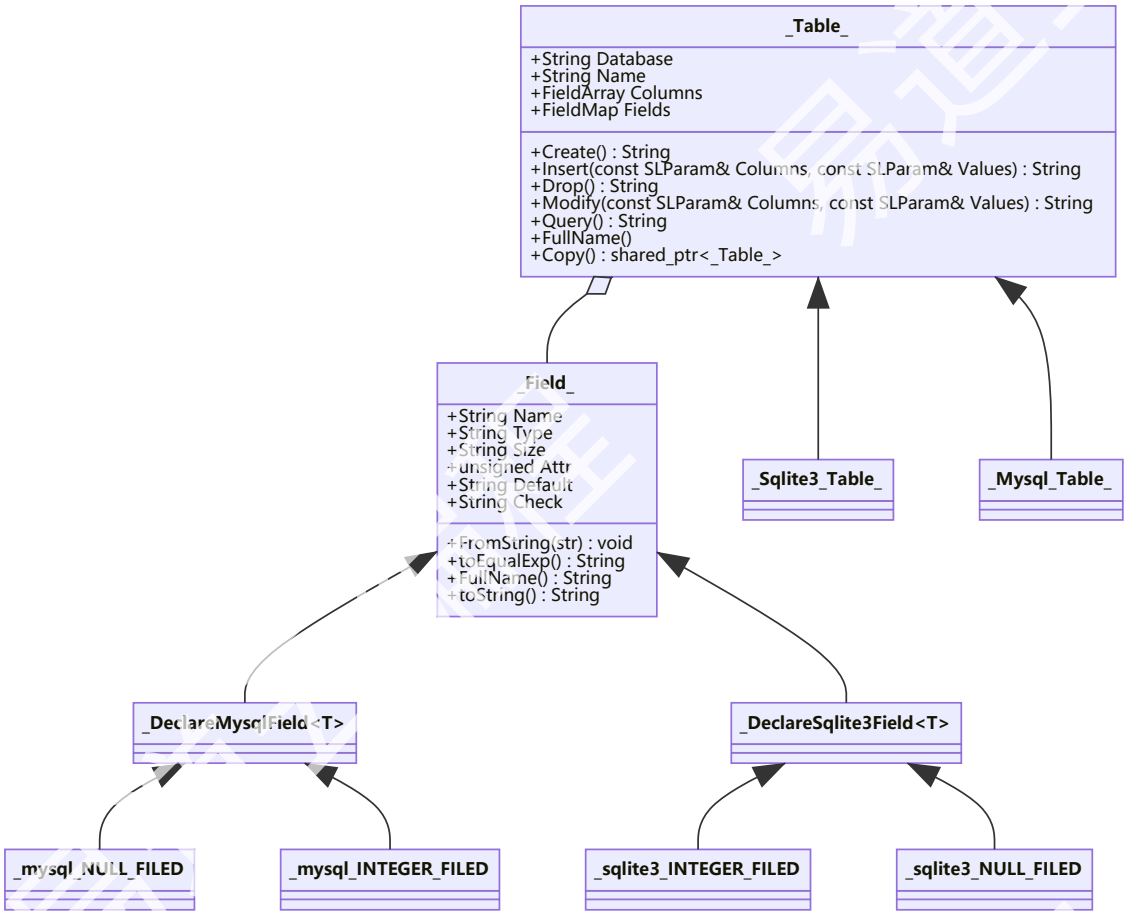
数据库模块的设计



数据库的基本流程：


Connect
Exec
处理结果
Close
Connect
StartTransaction
Exec
CommitTransaction
Close

数据类的设计：



利用宏，对数据表的快速定义：

```
#define DECLARE_TABLE_CLASS(name, base) class name:public base { \
public: \
virtual PTable Copy() const {return PTable(new name(*this));} \
name():base(){Name=#name;

#define DECLARE_MYSQL_FIELD(ntype,name,attr,type,size,default_,check) \
{PField field(new _mysql_field_(ntype, #name, attr, type, size, default_, check));FieldDefine.push_back(field);Fields[#name] = field; }
```

```
8
9  #define DECLARE_TABLE_CLASS_EDN() }};
10
11 DECLARE_TABLE_CLASS(edoyunLogin_user_mysql,
   _Mysql_Table_)
12 DECLARE_ITEM(user_id, NOT_NULL | PRIMARY_KEY |
   AUTOINCREMENT, INTEGER_FILED, "", "", "")
13 DECLARE_ITEM(user_qq, NOT_NULL, VARCHAR_FILED, "
   (15)", "", "")  //QQ号
14 DECLARE_ITEM(user_phone, DEFAULT, VARCHAR_FILED,
   "(11)", "'18888888888'", "")  //手机
15 DECLARE_ITEM(user_name, NOT_NULL, TEXT_FILED, "",
   "", "")    //姓名
16 DECLARE_ITEM(user_nick, NOT_NULL, TEXT_FILED, "",
   "", "")    //昵称
17 DECLARE_ITEM(user_wechat, DEFAULT, TEXT_FILED,
   "", "NULL", "")
18 DECLARE_ITEM(user_wechat_id, DEFAULT, TEXT_FILED,
   "", "NULL", "")
19 DECLARE_ITEM(user_address, DEFAULT, TEXT_FILED,
   "", "", "")
20 DECLARE_ITEM(user_province, DEFAULT, TEXT_FILED,
   "", "", "")
21 DECLARE_ITEM(user_country, DEFAULT, TEXT_FILED,
   "", "", "")
22 DECLARE_ITEM(user_age, DEFAULT | CHECK,
   INTEGER_FILED, "", "18", "")
23 DECLARE_ITEM(user_male, DEFAULT, BOOL_FILED, "",
   "1", "")
24 DECLARE_ITEM(user_flags, DEFAULT, TEXT_FILED, "",
   "0", "")
25 DECLARE_ITEM(user_experience, DEFAULT,
   REAL_FILED, "", "0.0", "")
26 DECLARE_ITEM(user_level, DEFAULT | CHECK,
   INTEGER_FILED, "", "0", "")
27 DECLARE_ITEM(user_class_priority, DEFAULT,
   TEXT_FILED, "", "", "")
```

```
28 DECLARE_ITEM(user_time_per_viewer, DEFAULT,
   REAL_FILED, "", "", "")
29 DECLARE_ITEM(user_career, NONE, TEXT_FILED, "",
   "", "")
30 DECLARE_ITEM(user_password, NOT_NULL, TEXT_FILED,
   "", "", "")
31 DECLARE_ITEM(user_birthday, NONE, DATETIME_FILED,
   "", "", "")
32 DECLARE_ITEM(user_describe, NONE, TEXT_FILED, "",
   "", "")
33 DECLARE_ITEM(user_education, NONE, TEXT_FILED,
   "", "", "")
34 DECLARE_ITEM(user_register_time, DEFAULT,
   DATETIME_FILED, "", "LOCALTIME()", "")
35 DECLARE_TABLE_CLASS_END()
36
37 DECLARE_TABLE_CLASS(edoyunLogin_user_test,
   _Sqlite3_Table_)
38 DECLARE_ITEM(user_id, NOT_NULL | PRIMARY_KEY |
   AUTOINCREMENT, INTEGER_FILED, "", "", "")
39 DECLARE_ITEM(user_qq, NOT_NULL, VARCHAR_FILED, "
   (15)", "", "")  //QQ号
40 DECLARE_ITEM(user_phone, DEFAULT, TEXT_FILED, "",
   "18888888888", "")  //手机
41 DECLARE_ITEM(user_name, NOT_NULL, TEXT_FILED, "",
   "", "")    //姓名
42 DECLARE_ITEM(user_nick, NOT_NULL, TEXT_FILED, "",
   "", "")    //昵称
43 DECLARE_ITEM(user_wechat, DEFAULT, TEXT_FILED,
   "", "none", "")
44 DECLARE_ITEM(user_wechat_id, DEFAULT, TEXT_FILED,
   "", "none", "")
45 DECLARE_ITEM(user_address, DEFAULT, TEXT_FILED,
   "", "\"长安大街1号\"", "")
46 DECLARE_ITEM(user_province, DEFAULT, TEXT_FILED,
   "", "\"北京\"", "")
```

```
47 DECLARE_ITEM(user_country, DEFAULT, TEXT_FILED,
   "", "\"中国\"", "")
48 DECLARE_ITEM(user_age, DEFAULT | CHECK,
   INTEGER_FILED, "", "18", "\"user_age\" >= 0")
49 DECLARE_ITEM(user_male, DEFAULT, BOOL_FILED, "",
   "1", "")
50 DECLARE_ITEM(user_flags, DEFAULT, TEXT_FILED, "",
   "0", "")
51 DECLARE_ITEM(user_experience, DEFAULT,
   REAL_FILED, "", "0.0", "")
52 DECLARE_ITEM(user_level, DEFAULT | CHECK,
   INTEGER_FILED, "", "0", "\"user_level\" >= 0")
53 DECLARE_ITEM(user_class_priority, DEFAULT,
   TEXT_FILED, "", "", "")
54 DECLARE_ITEM(user_time_per_viewer, DEFAULT,
   REAL_FILED, "", "", "")
55 DECLARE_ITEM(user_career, NONE, TEXT_FILED, "",
   "", "")
56 DECLARE_ITEM(user_password, NOT_NULL, TEXT_FILED,
   "", "", "")
57 DECLARE_ITEM(user_birthday, NONE, DATETIME_FILED,
   "", "", "")
58 DECLARE_ITEM(user_describe, NONE, TEXT_FILED, "",
   "", "")
59 DECLARE_ITEM(user_education, NONE, TEXT_FILED,
   "", "", "")
60 DECLARE_ITEM(user_register_time, DEFAULT,
   DATETIME_FILED, "", "(datetime('now',
   'localtime'))", "")
61 DECLARE_TABLE_CLASS_END()
62
```

sqlite3数据库的实现

DataBaseHelper.h

```
1 #pragma once
```

```cpp
#include "Public.h"
#include <map>
#include <list>
#include <memory>
#include <vector>

class _Table_;
using PTable = std::shared_ptr<_Table_>;

using KeyValue = std::map<Buffer, Buffer>;
using Result = std::list<PTable>;

class CDatabaseClient
{
public:
    CDatabaseClient(const CDatabaseClient&) = delete;
    CDatabaseClient& operator=(const CDatabaseClient&) = delete;
public:
    CDatabaseClient() {}
    virtual ~CDatabaseClient() {}
public:
    //连接
    virtual int Connect(const KeyValue& args) = 0;
    //执行
    virtual int Exec(const Buffer& sql) = 0;
    //带结果的执行
    virtual int Exec(const Buffer& sql, Result& result, const _Table_& table) = 0;
    //开启事务
    virtual int StartTransaction() = 0;
    //提交事务
    virtual int CommitTransaction() = 0;
    //回滚事务
    virtual int RollbackTransaction() = 0;
```

```
35      //关闭连接
36      virtual int Close() = 0;
37      //是否连接
38      virtual bool IsConnected() = 0;
39  };
40
41  //表和列的基类的实现
42  class _Field_;
43  using PField = std::shared_ptr<_Field_>;
44  using FieldArray = std::vector<PField>;
45  using FieldMap = std::map<Buffer, PField>;
46
47
48  class _Table_ {
49  public:
50      _Table_() {}
51      virtual ~_Table_() {}
52      //返回创建的SQL语句
53      virtual Buffer Create() = 0;
54      //删除表
55      virtual Buffer Drop() = 0;
56      //增删改查
57      //TODO:参数进行优化
58      virtual Buffer Insert(const _Table_& values)
    = 0;
59      virtual Buffer Delete(const _Table_& values)
    = 0;
60      //TODO:参数进行优化
61      virtual Buffer Modify(const _Table_& values)
    = 0;
62      virtual Buffer Query() = 0;
63      //创建一个基于表的对象
64      virtual PTable Copy()const = 0;
65      virtual void ClearFieldUsed() = 0;
66  public:
67      //获取表的全名
68      virtual operator const Buffer() const = 0;
```

```
69  public:
70      //表所属的DB的名称
71      Buffer Database;
72      Buffer Name;
73      FieldArray FieldDefine;//列的定义（存储查询结
    果）
74      FieldMap Fields;//列的定义映射表
75  };
76
77  enum {
78      SQL_INSERT = 1,//插入的列
79      SQL_MODIFY = 2,//修改的列
80      SQL_CONDITION = 4//查询条件列
81  };
82
83  enum {
84      NOT_NULL = 1,
85      DEFAULT = 2,
86      UNIQUE = 4,
87      PRIMARY_KEY = 8,
88      CHECK = 16,
89      AUTOINCREMENT = 32
90  };
91
92  using SqlType = enum {
93      TYPE_NULL = 0,
94      TYPE_BOOL = 1,
95      TYPE_INT = 2,
96      TYPE_DATETIME = 4,
97      TYPE_REAL = 8,
98      TYPE_VARCHAR = 16,
99      TYPE_TEXT = 32,
100     TYPE_BLOB = 64
101 };
102
103 class _Field_
104 {
```

```
105 public:
106     _Field_() {}
107     _Field_(const _Field_& field) {
108         Name = field.Name;
109         Type = field.Type;
110         Attr = field.Attr;
111         Default = field.Default;
112         Check = field.Check;
113     }
114     virtual _Field_& operator=(const _Field_&
    field) {
115         if (this != &field) {
116             Name = field.Name;
117             Type = field.Type;
118             Attr = field.Attr;
119             Default = field.Default;
120             Check = field.Check;
121         }
122         return *this;
123     }
124     virtual ~_Field_() {}
125 public:
126     virtual Buffer Create() = 0;
127     virtual void LoadFromStr(const Buffer& str)
    = 0;
128     //where 语句使用的
129     virtual Buffer toEqualExp() const = 0;
130     virtual Buffer toSqlStr() const = 0;
131     //列的全名
132     virtual operator const Buffer() const = 0;
133 public:
134     Buffer Name;
135     Buffer Type;
136     Buffer Size;
137     unsigned Attr;
138     Buffer Default;
139     Buffer Check;
```

```
public:
    //操作条件
    unsigned Condition;
};
```

Sqlite3Client.h

```
#pragma once
#include "Public.h"
#include "DatabaseHelper.h"
#include "sqlite3/sqlite3.h"

class CSqlite3Client
    :public CDatabaseClient
{
public:
    CSqlite3Client(const CSqlite3Client&) = delete;
    CSqlite3Client& operator=(const CSqlite3Client&) = delete;
public:
    CSqlite3Client() {
        m_db = NULL;
        m_stmt = NULL;
    }
    virtual ~CSqlite3Client() {
        Close();
    }
public:
    //连接
    virtual int Connect(const KeyValue& args);
    //执行
    virtual int Exec(const Buffer& sql);
```

```
25      //带结果的执行
26      virtual int Exec(const Buffer& sql, Result&
result, const _Table_& table);
27      //开启事务
28      virtual int StartTransaction();
29      //提交事务
30      virtual int CommitTransaction();
31      //回滚事务
32      virtual int RollbackTransaction();
33      //关闭连接
34      virtual int Close();
35      //是否连接 true表示连接中 false表示未连接
36      virtual bool IsConnected();
37  private:
38      static int ExecCallback(void* arg, int
count, char** names, char** values);
39      int ExecCallback(Result& result, const
_Table_& table, int count, char** names, char**
values);
40  private:
41      sqlite3_stmt* m_stmt;
42      sqlite3* m_db;
43  private:
44      class ExecParam {
45      public:
46          ExecParam(CSqlite3Client* obj, Result&
result, const _Table_& table)
47              :obj(obj), result(result),
table(table)
48          {}
49          CSqlite3Client* obj;
50          Result& result;
51          const _Table_& table;
52      };
53  };
54
55  class _sqlite3_table_ :
```

```
    public _Table_
{
public:
    _sqlite3_table_() :_Table_() {}
    _sqlite3_table_(const _sqlite3_table_&
table);
    virtual ~_sqlite3_table_();
    //返回创建的SQL语句
    virtual Buffer Create();
    //删除表
    virtual Buffer Drop();
    //增删改查
    //TODO:参数进行优化
    virtual Buffer Insert(const _Table_&
values);
    virtual Buffer Delete(const _Table_&
values);
    //TODO:参数进行优化
    virtual Buffer Modify(const _Table_&
values);
    virtual Buffer Query();
    //创建一个基于表的对象
    virtual PTable Copy()const;
    virtual void ClearFieldUsed();
public:
    //获取表的全名
    virtual operator const Buffer() const;
};

class _sqlite3_field_ :
    public _Field_
{
public:
    _sqlite3_field_();
    _sqlite3_field_(
        int ntype,
        const Buffer& name,
```

```
89          unsigned attr,
90          const Buffer& type,
91          const Buffer& size,
92          const Buffer& default_,
93          const Buffer& check
94      );
95      _sqlite3_field_(const _sqlite3_field_&
field);
96      virtual ~_sqlite3_field_();
97      virtual Buffer Create();
98      virtual void LoadFromStr(const Buffer& str);
99      //where 语句使用的
100     virtual Buffer toEqualExp() const;
101     virtual Buffer toSqlStr() const;
102     //列的全名
103     virtual operator const Buffer() const;
104 private:
105     Buffer Str2Hex(const Buffer& data) const;
106     union {
107         bool Bool;
108         int Integer;
109         double Double;
110         Buffer* String;
111     }Value;
112     int nType;
113 };
114
115 #define DECLARE_TABLE_CLASS(name, base) class
    name:public base { \
116 public: \
117 virtual PTable Copy() const {return PTable(new
    name(*this));} \
118 name():base(){Name=#name;
119
120 #define
    DECLARE_FIELD(ntype,name,attr,type,size,default_
    ,check) \
```

```
121 {PField field(new _sqlite3_field_(ntype, #name,
    attr, type, size, default_,
    check));FieldDefine.push_back(field);Fields[#nam
    e] = field; }
122
123 #define DECLARE_TABLE_CLASS_EDN() }};
```

Sqlite3Client.cpp

```
1  #include "Sqlite3Client.h"
2  #include "Logger.h"
3
4  int CSqlite3Client::Connect(const KeyValue&
   args)
5  {
6      auto it = args.find("host");
7      if (it == args.end())return -1;
8      if (m_db != NULL)return -2;
9      int ret = sqlite3_open(it->second, &m_db);
10     if (ret != 0) {
11         TRACEE("connect failed:%d [%s]", ret,
   sqlite3_errmsg(m_db));
12         return -3;
13     }
14     return 0;
15 }
16
17 int CSqlite3Client::Exec(const Buffer& sql)
18 {
19     printf("sql={%s}\n", (char*)sql);
20     if (m_db == NULL)return -1;
21     int ret = sqlite3_exec(m_db, sql, NULL,
   this, NULL);
22     if (ret != SQLITE_OK) {
23         printf("sql={%s}\n", (char*)sql);
24         printf("Exec failed:%d [%s]\n", ret,
   sqlite3_errmsg(m_db));
```

```
25          return -2;
26      }
27      return 0;
28  }
29
30  int CSqlite3Client::Exec(const Buffer& sql,
    Result& result, const _Table_& table)
31  {
32      char* errmsg = NULL;
33      if (m_db == NULL)return -1;
34      printf("sql={%s}\n", (char*)sql);
35      ExecParam param(this, result, table);
36      int ret = sqlite3_exec(m_db, sql,
37          &CSqlite3Client::ExecCallback,
    (void*)&param, &errmsg);
38      if (ret != SQLITE_OK) {
39          printf("sql={%s}\n", sql);
40          printf("Exec failed:%d [%s]\n", ret,
    errmsg);
41          if (errmsg)sqlite3_free(errmsg);
42          return -2;
43      }
44      if (errmsg)sqlite3_free(errmsg);
45      return 0;
46  }
47
48  int CSqlite3Client::StartTransaction()
49  {
50      if (m_db == NULL)return -1;
51      int ret = sqlite3_exec(m_db, "BEGIN
    TRANSACTION", 0, 0, NULL);
52      if (ret != SQLITE_OK) {
53          TRACEE("sql={BEGIN TRANSACTION}");
54          TRACEE("BEGIN failed:%d [%s]", ret,
    sqlite3_errmsg(m_db));
55          return -2;
56      }
```

```
57      return 0;
58  }
59
60  int CSqlite3Client::CommitTransaction()
61  {
62      if (m_db == NULL)return -1;
63      int ret = sqlite3_exec(m_db, "COMMIT
    TRANSACTION", 0, 0, NULL);
64      if (ret != SQLITE_OK) {
65          TRACEE("sql={COMMIT TRANSACTION}");
66          TRACEE("COMMIT failed:%d [%s]", ret,
    sqlite3_errmsg(m_db));
67          return -2;
68      }
69      return 0;
70  }
71
72  int CSqlite3Client::RollbackTransaction()
73  {
74      if (m_db == NULL)return -1;
75      int ret = sqlite3_exec(m_db, "ROLLBACK
    TRANSACTION", 0, 0, NULL);
76      if (ret != SQLITE_OK) {
77          TRACEE("sql={ROLLBACK TRANSACTION}");
78          TRACEE("ROLLBACK failed:%d [%s]", ret,
    sqlite3_errmsg(m_db));
79          return -2;
80      }
81      return 0;
82  }
83
84  int CSqlite3Client::Close()
85  {
86      if (m_db == NULL)return -1;
87      int ret = sqlite3_close(m_db);
88      if (ret != SQLITE_OK) {
```

```
89          TRACEE("Close failed:%d [%s]", ret,
    sqlite3_errmsg(m_db));
90          return -2;
91      }
92      m_db = NULL;
93      return 0;
94  }
95
96  bool CSqlite3Client::IsConnected()
97  {
98      return m_db != NULL;
99  }
100
101 int CSqlite3Client::ExecCallback(void* arg, int
    count, char** values, char** names)
102 {
103     ExecParam* param = (ExecParam*)arg;
104     return param->obj->ExecCallback(param-
    >result, param->table, count, names, values);
105 }
106
107 int CSqlite3Client::ExecCallback(Result& result,
    const _Table_& table, int count, char** names,
    char** values)
108 {
109     PTable pTable = table.Copy();
110     if (pTable == nullptr) {
111         printf("table %s error!\n", (const
    char*)(Buffer)table);
112         return -1;
113     }
114     for (int i = 0; i < count; i++) {
115         Buffer name = names[i];
116         auto it = pTable->Fields.find(name);
117         if (it == pTable->Fields.end()) {
118             printf("table %s error!\n", (const
    char*)(Buffer)table);
```

```
119             return -2;
120         }
121         if (values[i] != NULL)
122             it->second->LoadFromStr(values[i]);
123     }
124     result.push_back(pTable);
125     return 0;
126 }
127
128 _sqlite3_table_::_sqlite3_table_(const
    _sqlite3_table_& table)
129 {
130     Database = table.Database;
131     Name = table.Name;
132     for (size_t i = 0; i <
    table.FieldDefine.size(); i++)
133     {
134         PField field = PField(new
    _sqlite3_field_(*
135
    (_sqlite3_field_*)table.FieldDefine[i].get()));
136         FieldDefine.push_back(field);
137         Fields[field->Name] = field;
138     }
139 }
140
141 _sqlite3_table_::~_sqlite3_table_()
142 {}
143
144 Buffer _sqlite3_table_::Create()
145 {   //CREATE TABLE IF NOT EXISTS 表全名 (列定
    义,……);
146     //表全名 = 数据库.表名
147     Buffer sql = "CREATE TABLE IF NOT EXISTS " +
    (Buffer)*this + "(\r\n";
148     for (size_t i = 0; i < FieldDefine.size();
    i++) {
```

```
149         if (i > 0)sql += ",";
150         sql += FieldDefine[i]->Create();
151     }
152     sql += ");";
153     TRACEI("sql = %s", (char*)sql);
154     return sql;
155 }
156 
157 Buffer _sqlite3_table_::Drop()
158 {
159     Buffer sql = "DROP TABLE " + (Buffer)*this +
";";
160     TRACEI("sql = %s", (char*)sql);
161     return sql;
162 }
163 
164 Buffer _sqlite3_table_::Insert(const _Table_&
values)
165 {   //INSERT INTO 表全名 (列1,...,列n)
166     //VALUES(值1,...,值n);
167     Buffer sql = "INSERT INTO " + (Buffer)*this
+ " (";
168     bool isfirst = true;
169     for (size_t i = 0; i <
values.FieldDefine.size(); i++) {
170         if (values.FieldDefine[i]->Condition &
SQL_INSERT) {
171             if (!isfirst)sql += ",";
172             else isfirst = false;
173             sql +=
(Buffer)*values.FieldDefine[i];
174         }
175     }
176     sql += ") VALUES (";
177     isfirst = true;
178     for (size_t i = 0; i <
values.FieldDefine.size(); i++) {
```

```
179         if (values.FieldDefine[i]->Condition &
SQL_INSERT) {
180             if (!isfirst)sql += ",";
181             else isfirst = false;
182             sql += values.FieldDefine[i]-
>toSqlStr();
183         }
184     }
185     sql += " );";
186     TRACEI("sql = %s", (char*)sql);
187     return sql;
188 }
189
190 Buffer _sqlite3_table_::Delete(const _Table_&
values)
191 {// DELETE FROM 表全名 WHERE 条件
192     Buffer sql = "DELETE FROM " + (Buffer)*this
+ " ";
193     Buffer Where = "";
194     bool isfirst = true;
195     for (size_t i = 0; i < FieldDefine.size();
i++) {
196         if (FieldDefine[i]->Condition &
SQL_CONDITION) {
197             if (!isfirst)Where += " AND ";
198             else isfirst = false;
199             Where += (Buffer)*FieldDefine[i] +
"=" + FieldDefine[i]->toSqlStr();
200         }
201     }
202     if (Where.size() > 0)
203         sql += " WHERE " + Where;
204     sql += ";";
205     TRACEI("sql = %s", (char*)sql);
206     return sql;
207 }
208
```

```
209 Buffer _sqlite3_table_::Modify(const _Table_&
    values)
210 {
211     //UPDATE 表全名 SET 列1=值1 , ... , 列n=值n
    [WHERE 条件];
212     Buffer sql = "UPDATE " + (Buffer)*this + "
    SET ";
213     bool isfirst = true;
214     for (size_t i = 0; i <
    values.FieldDefine.size(); i++) {
215         if (values.FieldDefine[i]->Condition &
    SQL_MODIFY) {
216             if (!isfirst)sql += ",";
217             else isfirst = false;
218             sql +=
    (Buffer)*values.FieldDefine[i] + "=" +
    values.FieldDefine[i]->toSqlStr();
219         }
220     }
221
222     Buffer Where = "";
223     for (size_t i = 0; i <
    values.FieldDefine.size(); i++) {
224         if (values.FieldDefine[i]->Condition &
    SQL_CONDITION) {
225             if (!isfirst)Where += " AND ";
226             else isfirst = false;
227             Where +=
    (Buffer)*values.FieldDefine[i] + "=" +
    values.FieldDefine[i]->toSqlStr();
228         }
229     }
230     if (Where.size() > 0)
231         sql += " WHERE " + Where;
232     sql += " ;";
233     TRACEI("sql = %s", (char*)sql);
234     return sql;
```

```
}

Buffer _sqlite3_table_::Query()
{//SELECT 列名1 ,列名2 ,... ,列名n FROM 表全名;
    Buffer sql = "SELECT ";
    for (size_t i = 0; i < FieldDefine.size(); i++)
    {
        if (i > 0)sql += ',';
        sql += '"' + FieldDefine[i]->Name + "\"";
    }
    sql += " FROM " + (Buffer)*this + ";";
    TRACEI("sql = %s", (char*)sql);
    return sql;
}

PTable _sqlite3_table_::Copy() const
{
    return PTable(new _sqlite3_table_(*this));
}

void _sqlite3_table_::ClearFieldUsed()
{
    for (size_t i = 0; i < FieldDefine.size(); i++) {
        FieldDefine[i]->Condition = 0;
    }
}

_sqlite3_table_::operator const Buffer() const
{
    Buffer Head;
    if (Database.size())
        Head = '"' + Database + "\".";
    return Head + '"' + Name + '"';
}
```

```
269
270 _sqlite3_field_::_sqlite3_field_()
271     :_Field_() {
272     nType = TYPE_NULL;
273     Value.Double = 0.0;
274 }
275
276 _sqlite3_field_::_sqlite3_field_(int ntype,
    const Buffer& name, unsigned attr, const Buffer&
    type, const Buffer& size, const Buffer&
    default_, const Buffer& check)
277 {
278     nType = ntype;
279     switch (ntype)
280     {
281     case TYPE_VARCHAR:
282     case TYPE_TEXT:
283     case TYPE_BLOB:
284         Value.String = new Buffer();
285         break;
286     }
287
288     Name = name;
289     Attr = attr;
290     Type = type;
291     Size = size;
292     Default = default_;
293     Check = check;
294 }
295
296 _sqlite3_field_::_sqlite3_field_(const
    _sqlite3_field_& field)
297 {
298     nType = field.nType;
299     switch (field.nType)
300     {
301     case TYPE_VARCHAR:
```

```
302     case TYPE_TEXT:
303     case TYPE_BLOB:
304         Value.String = new Buffer();
305         *Value.String = *field.Value.String;
306         break;
307     }
308 
309     Name = field.Name;
310     Attr = field.Attr;
311     Type = field.Type;
312     Size = field.Size;
313     Default = field.Default;
314     Check = field.Check;
315 }
316 
317 _sqlite3_field_::~_sqlite3_field_()
318 {
319     switch (nType)
320     {
321     case TYPE_VARCHAR:
322     case TYPE_TEXT:
323     case TYPE_BLOB:
324         if (Value.String) {
325             Buffer* p = Value.String;
326             Value.String = NULL;
327             delete p;
328         }
329         break;
330     }
331 }
332 
333 Buffer _sqlite3_field_::Create()
334 {   //"名称" 类型 属性
335     Buffer sql = '"' + Name + "\" " + Type + "
    ";
336     if (Attr & NOT_NULL) {
337         sql += " NOT NULL ";
```

```
338     }
339     if (Attr & DEFAULT) {
340         sql += " DEFAULT " + Default + " ";
341     }
342     if (Attr & UNIQUE) {
343         sql += " UNIQUE ";
344     }
345     if (Attr & PRIMARY_KEY) {
346         sql += " PRIMARY KEY ";
347     }
348     if (Attr & CHECK) {
349         sql += " CHECK( " + Check + ") ";
350     }
351     if (Attr & AUTOINCREMENT) {
352         sql += " AUTOINCREMENT ";
353     }
354     return sql;
355 }
356
357 void _sqlite3_field_::LoadFromStr(const Buffer&
    str)
358 {
359     switch (nType)
360     {
361     case TYPE_NULL:
362         break;
363     case TYPE_BOOL:
364     case TYPE_INT:
365     case TYPE_DATETIME:
366         Value.Integer = atoi(str);
367         break;
368     case TYPE_REAL:
369         Value.Double = atof(str);
370         break;
371     case TYPE_VARCHAR:
372     case TYPE_TEXT:
373         *Value.String = str;
```

```
        break;
    case TYPE_BLOB:
        *Value.String = Str2Hex(str);
        break;
    default:
        TRACEW("type=%d", nType);
        break;
    }
}

Buffer _sqlite3_field_::toEqualExp() const
{
    Buffer sql = (Buffer)*this + " = ";
    std::stringstream ss;
    switch (nType)
    {
    case TYPE_NULL:
        sql += " NULL ";
        break;
    case TYPE_BOOL:
    case TYPE_INT:
    case TYPE_DATETIME:
        ss << Value.Integer;
        sql += ss.str() + " ";
        break;
    case TYPE_REAL:
        ss << Value.Double;
        sql += ss.str() + " ";
        break;
    case TYPE_VARCHAR:
    case TYPE_TEXT:
    case TYPE_BLOB:
        sql += '"' + *Value.String + "\" ";
        break;
    default:
        TRACEW("type=%d", nType);
        break;
```

```
411     }
412     return sql;
413 }
414
415 Buffer _sqlite3_field_::toSqlStr() const
416 {
417     Buffer sql = "";
418     std::stringstream ss;
419     switch (nType)
420     {
421     case TYPE_NULL:
422         sql += " NULL ";
423         break;
424     case TYPE_BOOL:
425     case TYPE_INT:
426     case TYPE_DATETIME:
427         ss << Value.Integer;
428         sql += ss.str() + " ";
429         break;
430     case TYPE_REAL:
431         ss << Value.Double;
432         sql += ss.str() + " ";
433         break;
434     case TYPE_VARCHAR:
435     case TYPE_TEXT:
436     case TYPE_BLOB:
437         sql += '"' + *Value.String + "\" ";
438         break;
439     default:
440         TRACEW("type=%d", nType);
441         break;
442     }
443     return sql;
444 }
445
446 _sqlite3_field_::operator const Buffer() const
447 {
```

```
448     return '"' + Name + '"';
449 }
450
451 Buffer _sqlite3_field_::Str2Hex(const Buffer&
    data) const
452 {
453     const char* hex = "0123456789ABCDEF";
454     std::stringstream ss;
455     for (auto ch : data)
456         ss << hex[(unsigned char)ch >> 4] <<
    hex[(unsigned char)ch & 0xF];
457     return ss.str();
458 }
459
460
461
462
```

MySQL数据库的实现

MysqlClient.h

```
1  #pragma once
2  #pragma once
3  #include "Public.h"
4  #include "DatabaseHelper.h"
5  #include <mysql/mysql.h>
6
7  class CMysqlClient
8      :public CDatabaseClient
9  {
10 public:
11     CMysqlClient(const CMysqlClient&) = delete;
12     CMysqlClient& operator=(const CMysqlClient&)
   = delete;
```

```
13 public:
14     CMysqlClient() {
15         bzero(&m_db, sizeof(m_db));
16         m_bInit = false;
17     }
18     virtual ~CMysqlClient() {
19         Close();
20     }
21 public:
22     //连接
23     virtual int Connect(const KeyValue& args);
24     //执行
25     virtual int Exec(const Buffer& sql);
26     //带结果的执行
27     virtual int Exec(const Buffer& sql, Result&
   result, const _Table_& table);
28     //开启事务
29     virtual int StartTransaction();
30     //提交事务
31     virtual int CommitTransaction();
32     //回滚事务
33     virtual int RollbackTransaction();
34     //关闭连接
35     virtual int Close();
36     //是否连接 true表示连接中 false表示未连接
37     virtual bool IsConnected();
38 private:
39     MYSQL m_db;
40     bool m_bInit;//默认是false 表示没有初始化 初始化
   之后，则为true，表示已经连接
41 private:
42     class ExecParam {
43     public:
44         ExecParam(CMysqlClient* obj, Result&
   result, const _Table_& table)
45             :obj(obj), result(result),
   table(table)
```

```
        {}
        CMysqlClient* obj;
        Result& result;
        const _Table_& table;
    };
};

class _mysql_table_ :
    public _Table_
{
public:
    _mysql_table_() :_Table_() {}
    _mysql_table_(const _mysql_table_& table);
    virtual ~_mysql_table_();
    //返回创建的SQL语句
    virtual Buffer Create();
    //删除表
    virtual Buffer Drop();
    //增删改查
    //TODO:参数进行优化
    virtual Buffer Insert(const _Table_&
values);
    virtual Buffer Delete(const _Table_&
values);
    //TODO:参数进行优化
    virtual Buffer Modify(const _Table_&
values);
    virtual Buffer Query();
    //创建一个基于表的对象
    virtual PTable Copy()const;
    virtual void ClearFieldUsed();
public:
    //获取表的全名
    virtual operator const Buffer() const;
};

class _mysql_field_ :
```

```
    public _Field_
{
public:
    _mysql_field_();
    _mysql_field_(
        int ntype,
        const Buffer& name,
        unsigned attr,
        const Buffer& type,
        const Buffer& size,
        const Buffer& default_,
        const Buffer& check
    );
    _mysql_field_(const _mysql_field_& field);
    virtual ~_mysql_field_();
    virtual Buffer Create();
    virtual void LoadFromStr(const Buffer& str);
    //where 语句使用的
    virtual Buffer toEqualExp() const;
    virtual Buffer toSqlStr() const;
    //列的全名
    virtual operator const Buffer() const;
private:
    Buffer Str2Hex(const Buffer& data) const;
    union {
        bool Bool;
        int Integer;
        double Double;
        Buffer* String;
    }Value;
    int nType;
};

#define DECLARE_TABLE_CLASS(name, base) class
name:public base { \
public: \
```

```
115 virtual PTable Copy() const {return PTable(new
    name(*this));} \
116 name():base(){Name=#name;
117
118 #define
    DECLARE_MYSQL_FIELD(ntype,name,attr,type,size,de
    fault_,check) \
119 {PField field(new _mysql_field_(ntype, #name,
    attr, type, size, default_,
    check));FieldDefine.push_back(field);Fields[#nam
    e] = field; }
120
121 #define DECLARE_TABLE_CLASS_EDN() }};
122
```

MysqlClient.cpp

```
 1 #include "MysqlClient.h"
 2 #include <sstream>
 3
 4 int CMysqlClient::Connect(const KeyValue& args)
 5 {
 6     if (m_bInit)return -1;
 7     MYSQL* ret = mysql_init(&m_db);
 8     if (ret == NULL)return -2;
 9     ret = mysql_real_connect(&m_db,
10         args.at("host"), args.at("user"),
11         args.at("password"), args.at("db"),
12         atoi(args.at("port")),
13         NULL, 0);
14     if ((ret == NULL) && (mysql_errno(&m_db) !=
    0)) {
15         printf("%s %s\n", __FUNCTION__,
    mysql_errno(&m_db));
16         mysql_close(&m_db);
17         bzero(&m_db, sizeof(m_db));
18         return -3;
```

```
19      }
20      m_bInit = true;
21      return 0;
22  }
23
24  int CMysqlClient::Exec(const Buffer& sql)
25  {
26      if (!m_bInit)return -1;
27      int ret = mysql_real_query(&m_db, sql,
    sql.size());
28      if (ret != 0) {
29          printf("%s %s\n", __FUNCTION__,
    mysql_errno(&m_db));
30          return -2;
31      }
32      return 0;
33  }
34
35  int CMysqlClient::Exec(const Buffer& sql,
    Result& result, const _Table_& table)
36  {
37      if (!m_bInit)return -1;
38      int ret = mysql_real_query(&m_db, sql,
    sql.size());
39      if (ret != 0) {
40          printf("%s %s\n", __FUNCTION__,
    mysql_errno(&m_db));
41          return -2;
42      }
43      MYSQL_RES* res = mysql_store_result(&m_db);
44      MYSQL_ROW row;
45      unsigned num_fields = mysql_num_fields(res);
46      while ((row = mysql_fetch_row(res)) != NULL)
    {
47          PTable pt = table.Copy();
48          for (unsigned i = 0; i < num_fields;
    i++) {
```

```
            if (row[i] != NULL) {
                pt->FieldDefine[i]->LoadFromStr(row[i]);
            }
        }
        result.push_back(pt);
    }
    return 0;
}

int CMysqlClient::StartTransaction()
{
    if (!m_bInit)return -1;
    int ret = mysql_real_query(&m_db, "BEGIN", 6);
    if (ret != 0) {
        printf("%s %s\n", __FUNCTION__, mysql_errno(&m_db));
        return -2;
    }
    return 0;
}

int CMysqlClient::CommitTransaction()
{
    if (!m_bInit)return -1;
    int ret = mysql_real_query(&m_db, "COMMIT", 7);
    if (ret != 0) {
        printf("%s %s\n", __FUNCTION__, mysql_errno(&m_db));
        return -2;
    }
    return 0;
}

int CMysqlClient::RollbackTransaction()
```

```
{
    if (!m_bInit)return -1;
    int ret = mysql_real_query(&m_db,
"ROLLBACK", 9);
    if (ret != 0) {
        printf("%s %s\n", __FUNCTION__,
mysql_errno(&m_db));
        return -2;
    }
    return 0;
}

int CMysqlClient::Close()
{
    if (m_bInit) {
        m_bInit = false;
        mysql_close(&m_db);
        bzero(&m_db, sizeof(m_db));
    }
    return 0;
}

bool CMysqlClient::IsConnected()
{
    return m_bInit;
}

_mysql_table_::_mysql_table_(const
_mysql_table_& table)
{
    Database = table.Database;
    Name = table.Name;
    for (size_t i = 0; i <
table.FieldDefine.size(); i++)
    {
        PField field = PField(new
_mysql_field_(*
```

```
113
(_mysql_field_*)table.FieldDefine[i].get()));
114         FieldDefine.push_back(field);
115         Fields[field->Name] = field;
116     }
117 }
118
119 _mysql_table_::~_mysql_table_()
120 {}
121
122 Buffer _mysql_table_::Create()
123 {   //CREATE TABLE IF NOT EXISTS 表全名 (列定
义,..., PRIMARY KEY `主键列名` ,UNIQUE INDEX `列名
_UNIQUE` (列名 ASC) VISIBLE );
124     Buffer sql = "CREATE TABLE IF NOT EXISTS " +
(Buffer)*this + " (\r\n";
125     for (unsigned i = 0; i < FieldDefine.size();
i++)
126     {
127         if (i > 0)sql += ",\r\n";
128         sql += FieldDefine[i]->Create();
129         if (FieldDefine[i]->Attr & PRIMARY_KEY)
{
130             sql += ",\r\n PRIMARY KEY (`" +
FieldDefine[i]->Name + "`)";
131         }
132         if (FieldDefine[i]->Attr & UNIQUE) {
133             sql += ",\r\n UNIQUE INDEX `" +
FieldDefine[i]->Name + "_UNIQUE` (";
134             sql += (Buffer)*FieldDefine[i] + "
ASC) VISIBLE ";
135         }
136     }
137     sql += ");";
138     return sql;
139 }
140
```

```
Buffer _mysql_table_::Drop()
{
    return "DROP TABLE" + (Buffer)*this;
}

Buffer _mysql_table_::Insert(const _Table_&
values)
{// INSERT INTO 表全名 (列名,...)VALUES(值,...);
    Buffer sql = "INSERT INTO " + (Buffer)*this
+ " (";
    bool isfirst = true;
    for (size_t i = 0; i <
values.FieldDefine.size(); i++) {
        if (values.FieldDefine[i]->Condition &
SQL_INSERT) {
            if (!isfirst)sql += ",";
            else isfirst = false;
            sql +=
(Buffer)*values.FieldDefine[i];
        }
    }
    sql += ") VALUES (";
    isfirst = true;
    for (size_t i = 0; i <
values.FieldDefine.size(); i++) {
        if (values.FieldDefine[i]->Condition &
SQL_INSERT) {
            if (!isfirst)sql += ",";
            else isfirst = false;
            sql += values.FieldDefine[i]-
>toSqlStr();
        }
    }
    sql += " );";
    printf("sql = %s\n", (char*)sql);
    return sql;
}
```

```
170
171 Buffer _mysql_table_::Delete(const _Table_&
    values)
172 {
173     Buffer sql = "DELETE FROM " + (Buffer)*this
    + " ";
174     Buffer Where = "";
175     bool isfirst = true;
176     for (size_t i = 0; i < FieldDefine.size();
    i++) {
177         if (FieldDefine[i]->Condition &
    SQL_CONDITION) {
178             if (!isfirst)Where += " AND ";
179             else isfirst = false;
180             Where += (Buffer)*FieldDefine[i] +
    "=" + FieldDefine[i]->toSqlStr();
181         }
182     }
183     if (Where.size() > 0)
184         sql += " WHERE " + Where;
185     sql += ";";
186     printf("sql = %s\r\n", (char*)sql);
187     return sql;
188 }
189
190 Buffer _mysql_table_::Modify(const _Table_&
    values)
191 {
192     Buffer sql = "UPDATE " + (Buffer)*this + "
    SET ";
193     bool isfirst = true;
194     for (size_t i = 0; i <
    values.FieldDefine.size(); i++) {
195         if (values.FieldDefine[i]->Condition &
    SQL_MODIFY) {
196             if (!isfirst)sql += ",";
197             else isfirst = false;
```

```
198             sql +=
(Buffer)*values.FieldDefine[i] + "=" +
values.FieldDefine[i]->toSqlStr();
199         }
200     }
201
202     Buffer Where = "";
203     for (size_t i = 0; i <
values.FieldDefine.size(); i++) {
204         if (values.FieldDefine[i]->Condition &
SQL_CONDITION) {
205             if (!isfirst)Where += " AND ";
206             else isfirst = false;
207             Where +=
(Buffer)*values.FieldDefine[i] + "=" +
values.FieldDefine[i]->toSqlStr();
208         }
209     }
210     if (Where.size() > 0)
211         sql += " WHERE " + Where;
212     sql += " ;";
213     printf("sql = %s\n", (char*)sql);
214     return sql;
215 }
216
217 Buffer _mysql_table_::Query()
218 {
219     Buffer sql = "SELECT ";
220     for (size_t i = 0; i < FieldDefine.size();
i++)
221     {
222         if (i > 0)sql += ',';
223         sql += '`' + FieldDefine[i]->Name + "`
";
224     }
225     sql += " FROM " + (Buffer)*this + ";";
226     printf("sql = %s\n", (char*)sql);
```

```
227     return sql;
228 }
229
230 PTable _mysql_table_::Copy() const
231 {
232     return PTable(new _mysql_table_(*this));
233 }
234
235 void _mysql_table_::ClearFieldUsed()
236 {
237     for (size_t i = 0; i < FieldDefine.size();
    i++) {
238         FieldDefine[i]->Condition = 0;
239     }
240 }
241
242 _mysql_table_::operator const Buffer() const
243 {
244     Buffer Head;
245     if (Database.size())
246         Head = '`' + Database + "`.";
247     return Head + '`' + Name + '`';
248 }
249
250 _mysql_field_::_mysql_field_() :_Field_()
251 {
252     nType = TYPE_NULL;
253     Value.Double = 0.0;
254 }
255
256 _mysql_field_::_mysql_field_(int ntype, const
    Buffer& name, unsigned attr, const Buffer& type,
    const Buffer& size, const Buffer& default_,
    const Buffer& check)
257 {
258     nType = ntype;
259     switch (ntype)
```

```
260     {
261     case TYPE_VARCHAR:
262     case TYPE_TEXT:
263     case TYPE_BLOB:
264         Value.String = new Buffer();
265         break;
266     }
267 
268     Name = name;
269     Attr = attr;
270     Type = type;
271     Size = size;
272     Default = default_;
273     Check = check;
274 }
275 
276 _mysql_field_::_mysql_field_(const
    _mysql_field_& field)
277 {
278     nType = field.nType;
279     switch (field.nType)
280     {
281     case TYPE_VARCHAR:
282     case TYPE_TEXT:
283     case TYPE_BLOB:
284         Value.String = new Buffer();
285         *Value.String = *field.Value.String;
286         break;
287     }
288 
289     Name = field.Name;
290     Attr = field.Attr;
291     Type = field.Type;
292     Size = field.Size;
293     Default = field.Default;
294     Check = field.Check;
295 }
```

```
296
297 _mysql_field_::~_mysql_field_()
298 {
299     switch (nType)
300     {
301     case TYPE_VARCHAR:
302     case TYPE_TEXT:
303     case TYPE_BLOB:
304         if (Value.String) {
305             Buffer* p = Value.String;
306             Value.String = NULL;
307             delete p;
308         }
309         break;
310     }
311 }
312
313 Buffer _mysql_field_::Create()
314 {
315     Buffer sql = "`" + Name + "` " + Type + Size
    + " ";
316     if (Attr & NOT_NULL) {
317         sql += "NOT NULL";
318     }
319     else {
320         sql += "NULL";
321     }
322     //BLOB TEXT GEOMETRY JSON不能有默认值的
323     if ((Attr & DEFAULT) && (Default.size() >
    0)&&(Type != "BLOB") && (Type != "TEXT") &&
    (Type != "GEOMETRY") && (Type != "JSON"))
324     {
325         sql += " DEFAULT \"" + Default + "\" ";
326     }
327     //UNIQUE PRIMARY_KEY 外面处理
328     //CHECK mysql不支持
329     if (Attr & AUTOINCREMENT) {
```

```
330         sql += " AUTO_INCREMENT ";
331     }
332     return sql;
333 }
334
335 void _mysql_field_::LoadFromStr(const Buffer&
    str)
336 {
337     switch (nType)
338     {
339     case TYPE_NULL:
340         break;
341     case TYPE_BOOL:
342     case TYPE_INT:
343     case TYPE_DATETIME:
344         Value.Integer = atoi(str);
345         break;
346     case TYPE_REAL:
347         Value.Double = atof(str);
348         break;
349     case TYPE_VARCHAR:
350     case TYPE_TEXT:
351         *Value.String = str;
352         break;
353     case TYPE_BLOB:
354         *Value.String = Str2Hex(str);
355         break;
356     default:
357         printf("type=%d\n", nType);
358         break;
359     }
360 }
361
362 Buffer _mysql_field_::toEqualExp() const
363 {
364     Buffer sql = (Buffer)*this + " = ";
365     std::stringstream ss;
```

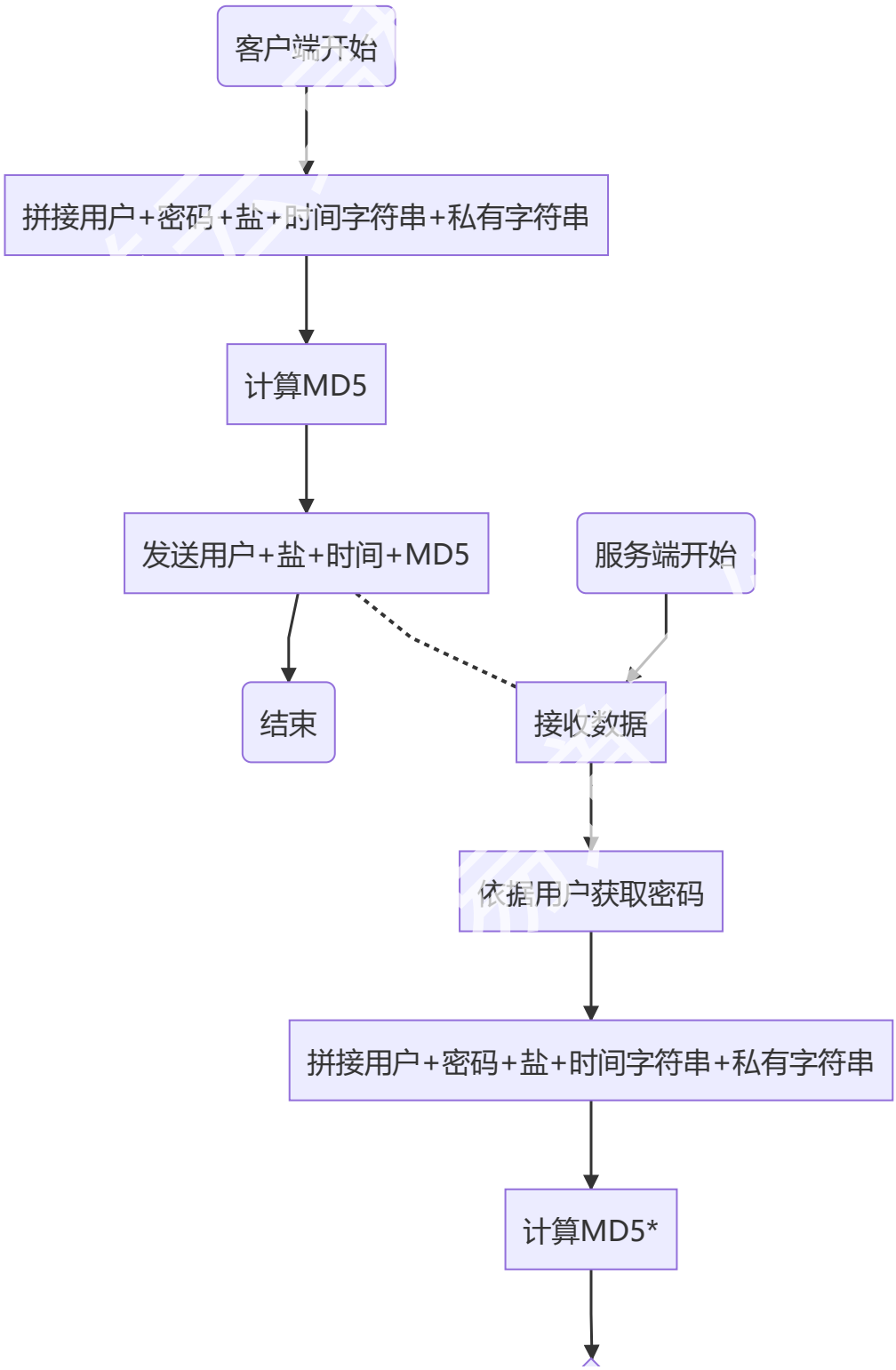
```
366     switch (nType)
367     {
368     case TYPE_NULL:
369         sql += " NULL ";
370         break;
371     case TYPE_BOOL:
372     case TYPE_INT:
373     case TYPE_DATETIME:
374         ss << Value.Integer;
375         sql += ss.str() + " ";
376         break;
377     case TYPE_REAL:
378         ss << Value.Double;
379         sql += ss.str() + " ";
380         break;
381     case TYPE_VARCHAR:
382     case TYPE_TEXT:
383     case TYPE_BLOB:
384         sql += "'" + *Value.String + "\" ";
385         break;
386     default:
387         printf("type=%d\n", nType);
388         break;
389     }
390     return sql;
391 }
392
393 Buffer _mysql_field_::toSqlStr() const
394 {
395     Buffer sql = "";
396     std::stringstream ss;
397     switch (nType)
398     {
399     case TYPE_NULL:
400         sql += " NULL ";
401         break;
402     case TYPE_BOOL:
```

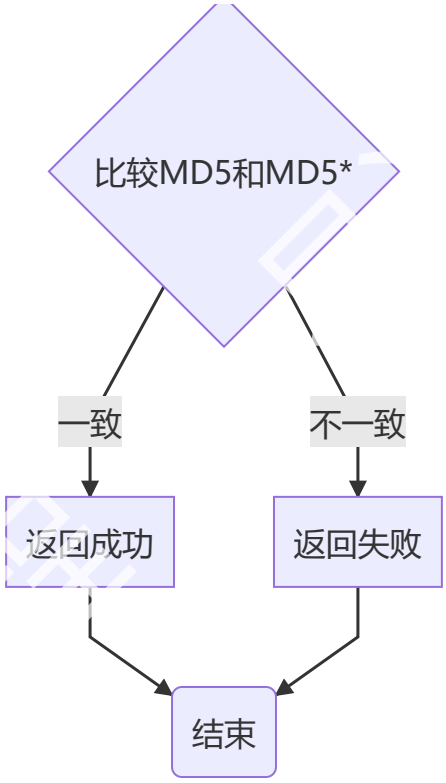
```
403     case TYPE_INT:
404     case TYPE_DATETIME:
405         ss << Value.Integer;
406         sql += ss.str() + " ";
407         break;
408     case TYPE_REAL:
409         ss << Value.Double;
410         sql += ss.str() + " ";
411         break;
412     case TYPE_VARCHAR:
413     case TYPE_TEXT:
414     case TYPE_BLOB:
415         sql += "\"" + *Value.String + "\" ";
416         break;
417     default:
418         printf("type=%d\n", nType);
419         break;
420     }
421     return sql;
422 }
423
424 _mysql_field_::operator const Buffer() const
425 {
426     return '`' + Name + '`';
427 }
428
429 Buffer _mysql_field_::Str2Hex(const Buffer&
    data) const
430 {
431     const char* hex = "0123456789ABCDEF";
432     std::stringstream ss;
433     for (auto ch : data)
434         ss << hex[(unsigned char)ch >> 4] <<
    hex[(unsigned char)ch & 0xF];
435     return ss.str();
436 }
437
```

加密模块的设计与实现

考虑到加密模块使用的方便性，工具类更合适，原因如下

- 无需声明对象
- 方法既可以相互独立，也可以相互关联
- 随取随用，无需配置或者初始化





OpenSSLHelper.h

```
class COpenSSLHelper
{
public:
    static Buffer MD5(const Buffer& text);
};

```

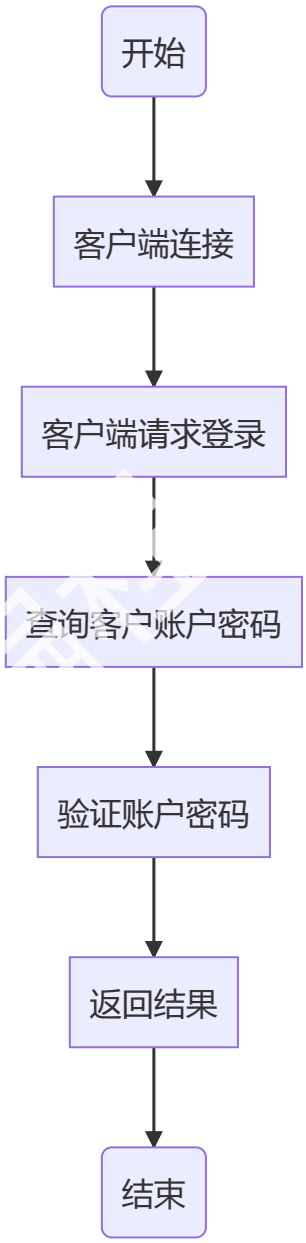
OpenSSLHelper.cpp

```
#include "OpenSSLHelper.h"
#include "openssl/md5.h"

Buffer COpenSSLHelper::MD5(const Buffer& text)
{

    Buffer result;
    std::vector<unsigned char> data;
    data.resize(16);
```

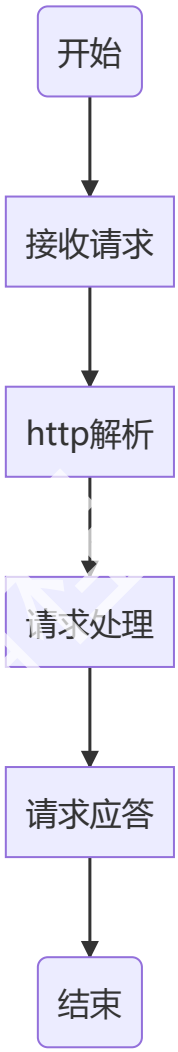
```
    MD5_CTX md5;
    MD5_Init(&md5);
    MD5_Update(&md5, text, text.size());
    MD5_Final(data.data(), &md5);
    char temp[3] = "";
    for (size_t i = 0; i < data.size(); i++)
    {
        snprintf(temp, sizeof(temp), "%02x", data[i] & 0xFF);
        result += temp;
    }
    return result;
}
```

业务功能的实现

业务流程



服务器处理流程



业务的实现

```
#pragma once
#include "Logger.h"
#include "CServer.h"
#include "HttpParser.h"
#include "Crypto.h"
#include "MysqlClient.h"
#include "jsoncpp/json.h"
#include <map>

DECLARE_TABLE_CLASS(edoyunLogin_user_mysql,
_mysql_table_)
```

```
11 DECLARE_MYSQL_FIELD(TYPE_INT, user_id, NOT_NULL
   | PRIMARY_KEY | AUTOINCREMENT, "INTEGER", "",
   "", "")
12 DECLARE_MYSQL_FIELD(TYPE_VARCHAR, user_qq,
   NOT_NULL, "VARCHAR", "(15)", "", "")  //QQ号
13 DECLARE_MYSQL_FIELD(TYPE_VARCHAR, user_phone,
   DEFAULT, "VARCHAR", "(11)", "'18888888888'", "")
    //手机
14 DECLARE_MYSQL_FIELD(TYPE_TEXT, user_name,
   NOT_NULL, "TEXT", "", "", "")    //姓名
15 DECLARE_MYSQL_FIELD(TYPE_TEXT, user_nick,
   NOT_NULL, "TEXT", "", "", "")    //昵称
16 DECLARE_MYSQL_FIELD(TYPE_TEXT, user_wechat,
   DEFAULT, "TEXT", "", "NULL", "")
17 DECLARE_MYSQL_FIELD(TYPE_TEXT, user_wechat_id,
   DEFAULT, "TEXT", "", "NULL", "")
18 DECLARE_MYSQL_FIELD(TYPE_TEXT, user_address,
   DEFAULT, "TEXT", "", "", "")
19 DECLARE_MYSQL_FIELD(TYPE_TEXT, user_province,
   DEFAULT, "TEXT", "", "", "")
20 DECLARE_MYSQL_FIELD(TYPE_TEXT, user_country,
   DEFAULT, "TEXT", "", "", "")
21 DECLARE_MYSQL_FIELD(TYPE_INT, user_age, DEFAULT
   | CHECK, "INTEGER", "", "18", "")
22 DECLARE_MYSQL_FIELD(TYPE_INT, user_male,
   DEFAULT, "BOOL", "", "1", "")
23 DECLARE_MYSQL_FIELD(TYPE_TEXT, user_flags,
   DEFAULT, "TEXT", "", "0", "")
24 DECLARE_MYSQL_FIELD(TYPE_REAL, user_experience,
   DEFAULT, "REAL", "", "0.0", "")
25 DECLARE_MYSQL_FIELD(TYPE_INT, user_level,
   DEFAULT | CHECK, "INTEGER", "", "0", "")
26 DECLARE_MYSQL_FIELD(TYPE_TEXT,
   user_class_priority, DEFAULT, "TEXT", "", "",
   "")
```

```
27 DECLARE_MYSQL_FIELD(TYPE_REAL,
   user_time_per_viewer, DEFAULT, "REAL", "", "",
   "")
28 DECLARE_MYSQL_FIELD(TYPE_TEXT, user_career,
   NONE, "TEXT", "", "", "")
29 DECLARE_MYSQL_FIELD(TYPE_TEXT, user_password,
   NOT_NULL, "TEXT", "", "", "")
30 DECLARE_MYSQL_FIELD(TYPE_INT, user_birthday,
   NONE, "DATETIME", "", "", "")
31 DECLARE_MYSQL_FIELD(TYPE_TEXT, user_describe,
   NONE, "TEXT", "", "", "")
32 DECLARE_MYSQL_FIELD(TYPE_TEXT, user_education,
   NONE, "TEXT", "", "", "")
33 DECLARE_MYSQL_FIELD(TYPE_INT,
   user_register_time, DEFAULT, "DATETIME", "",
   "LOCALTIME()", "")
34 DECLARE_TABLE_CLASS_EDN()
35
36 /*
37 * 1. 客户端的地址问题
38 * 2. 连接回调的参数问题
39 * 3. 接收回调的参数问题
40 */
41 #define ERR_RETURN(ret, err) if(ret!=0)
   {TRACEE("ret= %d errno = %d msg = [%s]", ret,
   errno, strerror(errno));return err;}
42
43 #define WARN_CONTINUE(ret) if(ret!=0)
   {TRACEW("ret= %d errno = %d msg = [%s]", ret,
   errno, strerror(errno));continue;}
44
45 class CEdoyunPlayerServer :
46     public CBusiness
47 {
48 public:
49     CEdoyunPlayerServer(unsigned count)
   :CBusiness() {
```

```
50          m_count = count;
51      }
52      ~CEdoyunPlayerServer() {
53          if (m_db) {
54              CDatabaseClient* db = m_db;
55              m_db = NULL;
56              db->Close();
57              delete db;
58          }
59          m_epoll.Close();
60          m_pool.Close();
61          for (auto it : m_mapClients) {
62              if (it.second) {
63                  delete it.second;
64              }
65          }
66          m_mapClients.clear();
67      }
68      virtual int BusinessProcess(CProcess* proc)
    {
69          using namespace std::placeholders;
70          int ret = 0;
71          m_db = new CMysqlClient();
72          if (m_db == NULL) {
73              TRACEE("no more memory!");
74              return -1;
75          }
76          KeyValue args;
77          args["host"] = "192.168.1.100";
78          args["user"] = "root";
79          args["password"] = "123456";
80          args["port"] = 3306;
81          args["db"] = "edoyun";
82          ret = m_db->Connect(args);
83          ERR_RETURN(ret, -2);
84          edoyunLogin_user_mysql user;
85          ret = m_db->Exec(user.Create());
```

```
86          ERR_RETURN(ret, -3);
87          ret =
setConnectedCallback(&CEdoyunPlayerServer::Conne
cted, this, _1);
88          ERR_RETURN(ret, -4);
89          ret =
setRecvCallback(&CEdoyunPlayerServer::Received,
this, _1, _2);
90          ERR_RETURN(ret, -5);
91          ret = m_epoll.Create(m_count);
92          ERR_RETURN(ret, -6);
93          ret = m_pool.Start(m_count);
94          ERR_RETURN(ret, -7);
95          for (unsigned i = 0; i < m_count; i++) {
96              ret =
m_pool.AddTask(&CEdoyunPlayerServer::ThreadFunc,
this);
97              ERR_RETURN(ret, -8);
98          }
99          int sock = 0;
100         sockaddr_in addrin;
101         while (m_epoll != -1) {
102             ret = proc->RecvSocket(sock,
&addrin);
103             TRACEI("RecvSocket ret=%d", ret);
104             if (ret < 0 || (sock == 0))break;
105             CSocketBase* pClient = new
CSocket(sock);
106             if (pClient == NULL)continue;
107             ret = pClient-
>Init(CSockParam(&addrin, SOCK_ISIP));
108             WARN_CONTINUE(ret);
109             ret = m_epoll.Add(sock,
EpollData((void*)pClient));
110             if (m_connectedcallback) {
111                 (*m_connectedcallback)(pClient);
112             }
```

```
113             WARN_CONTINUE(ret);
114         }
115         return 0;
116     }
117 private:
118     int Connected(CSocketBase* pClient) {
119         //TODO:客户端连接处理 简单打印一下客户端信息
120         sockaddr_in* paddr = *pClient;
121         TRACEI("client connected addr %s
    port:%d", inet_ntoa(paddr->sin_addr), paddr-
    >sin_port);
122         return 0;
123     }
124 
125     int Received(CSocketBase* pClient, const
    Buffer& data) {
126         TRACEI("接收到数据! ");
127         //TODO:主要业务，在此处理
128         //HTTP 解析
129         int ret = 0;
130         Buffer response = "";
131         ret = HttpParser(data);
132         TRACEI("HttpParser ret=%d", ret);
133         //验证结果的反馈
134         if (ret != 0) {//验证失败
135             TRACEE("http parser failed!%d",
    ret);
136         }
137         response = MakeResponse(ret);
138         ret = pClient->Send(response);
139         if (ret != 0) {
140             TRACEE("http response failed!%d
    [%s]", ret, (char*)response);
141         }
142         else {
143             TRACEI("http response success!%d",
    ret);
```

```
144         }
145         return 0;
146     }
147     int HttpParser(const Buffer& data) {
148         CHttpParser parser;
149         size_t size = parser.Parser(data);
150         if (size == 0 || (parser.Errno() != 0))
{
151             TRACEE("size %llu errno:%u", size,
parser.Errno());
152             return -1;
153         }
154         if (parser.Method() == HTTP_GET) {
155             //get 处理
156             UrlParser
url("https://192.168.1.100" + parser.Url());
157             int ret = url.Parser();
158             if (ret != 0) {
159                 TRACEE("ret = %d url[%s]", ret,
"https://192.168.1.100" + parser.Url());
160                 return -2;
161             }
162             Buffer uri = url.Uri();
163             TRACEI("**** uri = %s", (char*)uri);
164             if (uri == "login") {
165                 //处理登录
166                 Buffer time = url["time"];
167                 Buffer salt = url["salt"];
168                 Buffer user = url["user"];
169                 Buffer sign = url["sign"];
170                 TRACEI("time %s salt %s user %s
sign %s", (char*)time, (char*)salt, (char*)user,
(char*)sign);
171                 //数据库的查询
172                 edoyunLogin_user_mysql dbuser;
173                 Result result;
```

```
174                 Buffer sql =
dbuser.Query("user_name=\"" + user + "\"");
175                 ret = m_db->Exec(sql, result,
dbuser);
176                 if (ret != 0) {
177                     TRACEE("sql=%s ret=%d",
(char*)sql, ret);
178                     return -3;
179                 }
180                 if (result.size() == 0) {
181                     TRACEE("no result sql=%s
ret=%d", (char*)sql, ret);
182                     return -4;
183                 }
184                 if (result.size() != 1) {
185                     TRACEE("more than one sql=%s
ret=%d", (char*)sql, ret);
186                     return -5;
187                 }
188                 auto user1 = result.front();
189                 Buffer pwd = *user1-
>Fields["user_password"]->Value.String;
190                 TRACEI("password = %s",
(char*)pwd);
191                 //登录请求的验证
192                 const char* MD5_KEY =
"*&^%$#@b.v+h-b*g/h@n!h#n$d^ssx,.kl<kl";
193                 Buffer md5str = time + MD5_KEY +
pwd + salt;
194                 Buffer md5 =
Crypto::MD5(md5str);
195                 TRACEI("md5 = %s", (char*)md5);
196                 if (md5 == sign) {
197                     return 0;
198                 }
199                 return -6;
200             }
```

```
201         }
202         else if (parser.Method() == HTTP_POST) {
203             //post 处理
204         }
205         return -7;
206     }
207     Buffer MakeResponse(int ret) {
208         Json::Value root;
209         root["status"] = ret;
210         if (ret != 0) {
211             root["message"] = "登录失败，可能是用户名或者密码错误！";
212         }
213         else {
214             root["message"] = "success";
215         }
216         Buffer json = root.toStyledString();
217         Buffer result = "HTTP/1.1 200 OK\r\n";
218         time_t t;
219         time(&t);
220         tm* ptm = localtime(&t);
221         char temp[64] = "";
222         strftime(temp, sizeof(temp), "%a, %d %b %G %T GMT\r\n", ptm);
223         Buffer Date = Buffer("Date: ") + temp;
224         Buffer Server = "Server: Edoyun/1.0\r\nContent-Type: text/html; charset=utf-8\r\nX-Frame-Options: DENY\r\n";
225         snprintf(temp, sizeof(temp), "%d", json.size());
226         Buffer Length = Buffer("Content-Length: ") + temp + "\r\n";
227         Buffer Stub = "X-Content-Type-Options: nosniff\r\nReferrer-Policy: same-origin\r\n\r\n";
228         result += Date + Server + Length + Stub + json;
```

```
229         TRACEI("response: %s", (char*)result);
230         return result;
231     }
232 private:
233     int ThreadFunc() {
234         int ret = 0;
235         EPEvents events;
236         while (m_epoll != -1) {
237             ssize_t size =
    m_epoll.WaitEvents(events);
238             if (size < 0)break;
239             if (size > 0) {
240                 for (ssize_t i = 0; i < size;
    i++)
241                 {
242                     if (events[i].events &
    EPOLLERR) {
243                         break;
244                     }
245                     else if (events[i].events &
    EPOLLIN) {
246                         CSocketBase* pClient =
    (CSocketBase*)events[i].data.ptr;
247                         if (pClient) {
248                             Buffer data;
249                             ret = pClient-
    >Recv(data);
250                             TRACEI("recv data
    size %d", ret);
251                             if (ret <= 0) {
252                                 TRACEW("ret= %d
    errno = %d msg = [%s]", ret, errno,
    strerror(errno));
253
    m_epoll.Del(*pClient);
254                                 continue;
255                             }
```

```
256                                 if (m_recvcallback)
    {
257
    (*m_recvcallback)(pClient, data);
258                                 }
259                             }
260                         }
261                     }
262                 }
263             }
264             return 0;
265         }
266 private:
267     CEpoll m_epoll;
268     std::map<int, CSocketBase*> m_mapClients;
269     CThreadPool m_pool;
270     unsigned m_count;
271     CDatabaseClient* m_db;
272 };
```

项目测试

测试是贯穿整个项目开发的一项~~重要~~**必要**工作。

没有测试过的代码，是一个黑洞，你永远不知道里面隐藏了多少bug

虽然经过了测试的代码，也不是绝对可靠。

但是我们可以通过测试明白，在哪些情况下，代码是可以靠的。

所以测试的越全面，代码越可靠。

测试的设计

对于开发人员，测试一般分为功能测试和性能测试。

有的书也会提到可靠性测试、安全测试。

但是这两种我认为是性能的一种，在这里就不单独论述了。

此外，测试也可以分为黑盒测试、白盒测试和灰盒测试

也有动态测试和静态测试、单元测试和集成测试、等等之分

功能的测试

功能测试一般是指单元测试和模块测试。

主要目的是**验证项目的功能是否正确实现，和预期一致。**

性能的测试

性能测试包括：稳定性测试和压力测试

稳定性测试一般是写固定的脚本或者程序，反复触发被测试程序的功能或接口。

触发可以按照次数触发或者按照时间触发。

比如接口类的，会按照次数来计算。每千/万/十万/百万次调用，失败的次数。

比如时间类的，会按照系统使用多少小时/天，出现错误/崩溃的次数来计算。