易播Linux后台服务器

开发环境搭建

预备程序

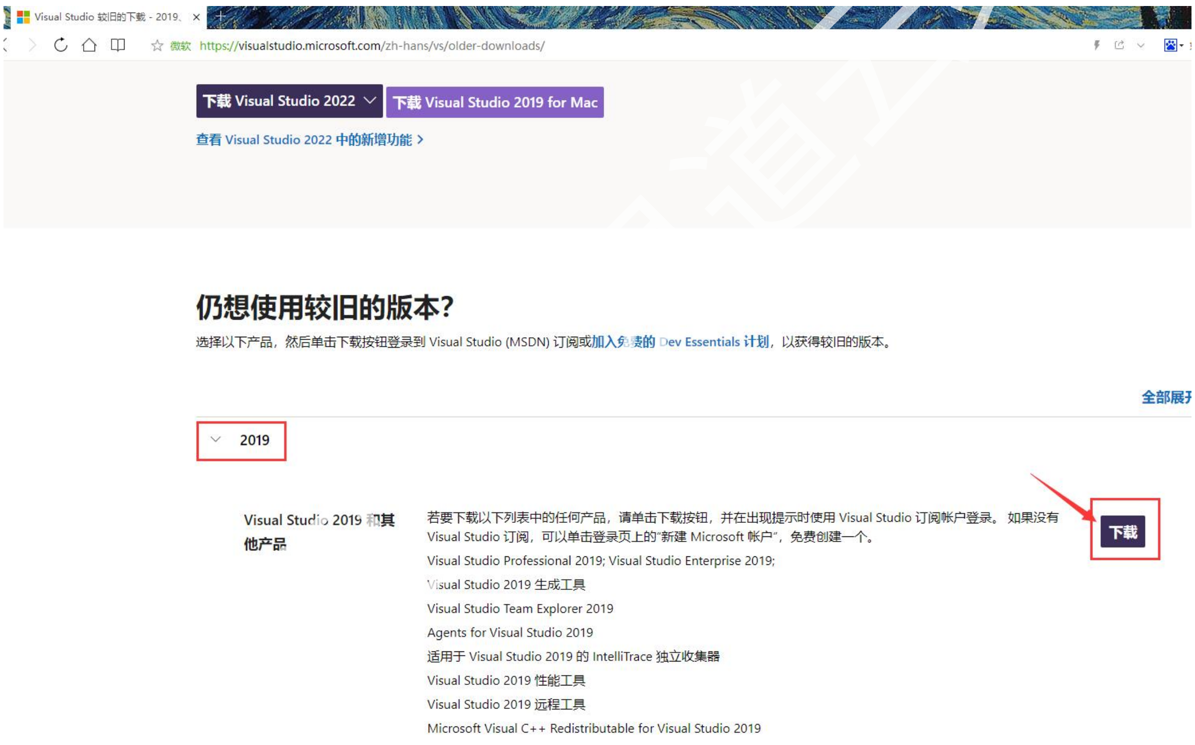
以下程序联系助教获取

- 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



Visual Studio 较旧的下载 - 2019、
微软 https://visualstudio.microsoft.com/zh-hans/vs/older-downloads/
下载 Visual Studio 2022
下载 Visual Studio 2019 for Mac
查看 Visual Studio 2022 中的新增功能 >
全部展开
2019
下载

仍想使用较旧的版本?

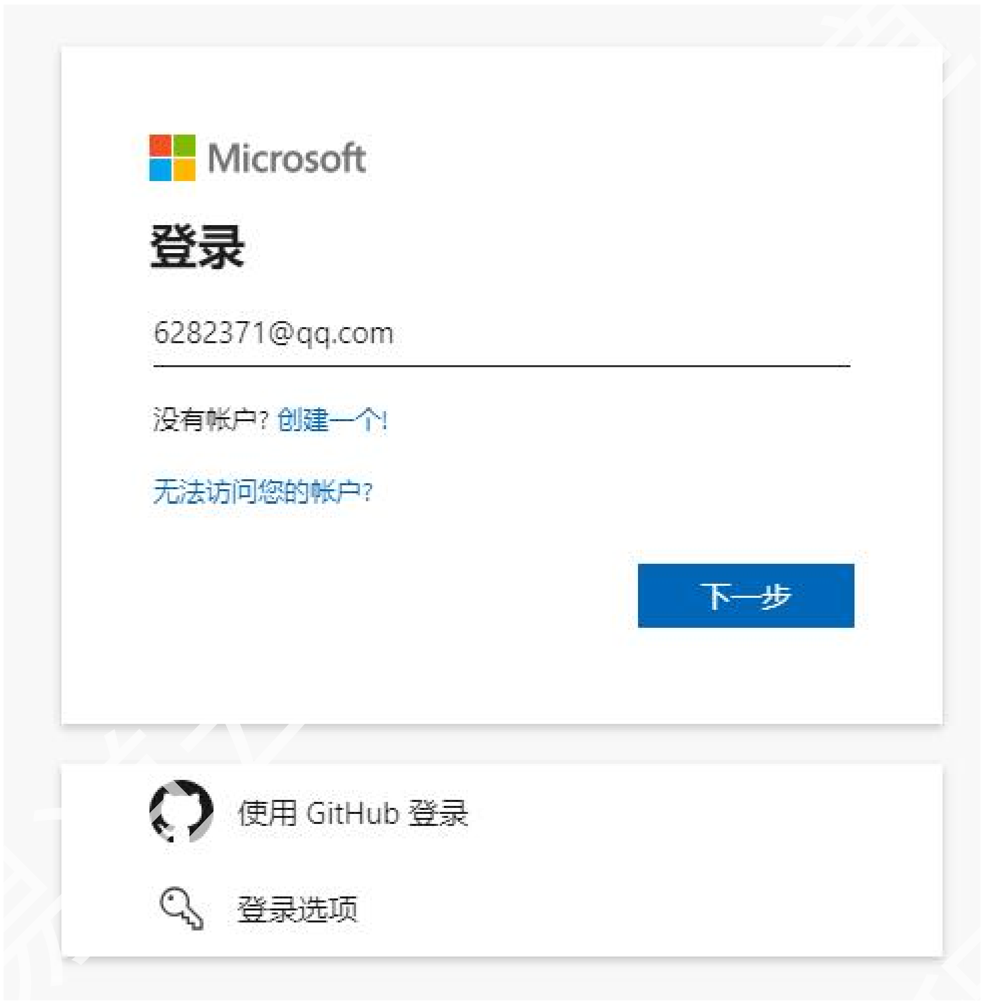
选择以下产品，然后单击下载按钮登录到 Visual Studio (MSDN) 订阅或加入免费的 Dev Essentials 计划，以获得较旧的版本。

Visual Studio 2019 和其他产品

若要下载以下列表中的任何产品，请单击下载按钮，并在出现提示时使用 Visual Studio 订阅帐户登录。 如果没有 Visual Studio 订阅，可以单击登录页上的"新建 Microsoft 帐户"，免费创建一个。

Visual Studio Professional 2019; Visual Studio Enterprise 2019;
Visual Studio 2019 生成工具
Visual Studio Team Explorer 2019
Agents for Visual Studio 2019
适用于 Visual Studio 2019 的 IntelliTrace 独立收集器
Visual Studio 2019 性能工具
Visual Studio 2019 远程工具
Microsoft Visual C++ Redistributable for Visual Studio 2019

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


Microsoft

登录

6282371@qq.com

没有帐户? 创建一个!

无法访问您的帐户?

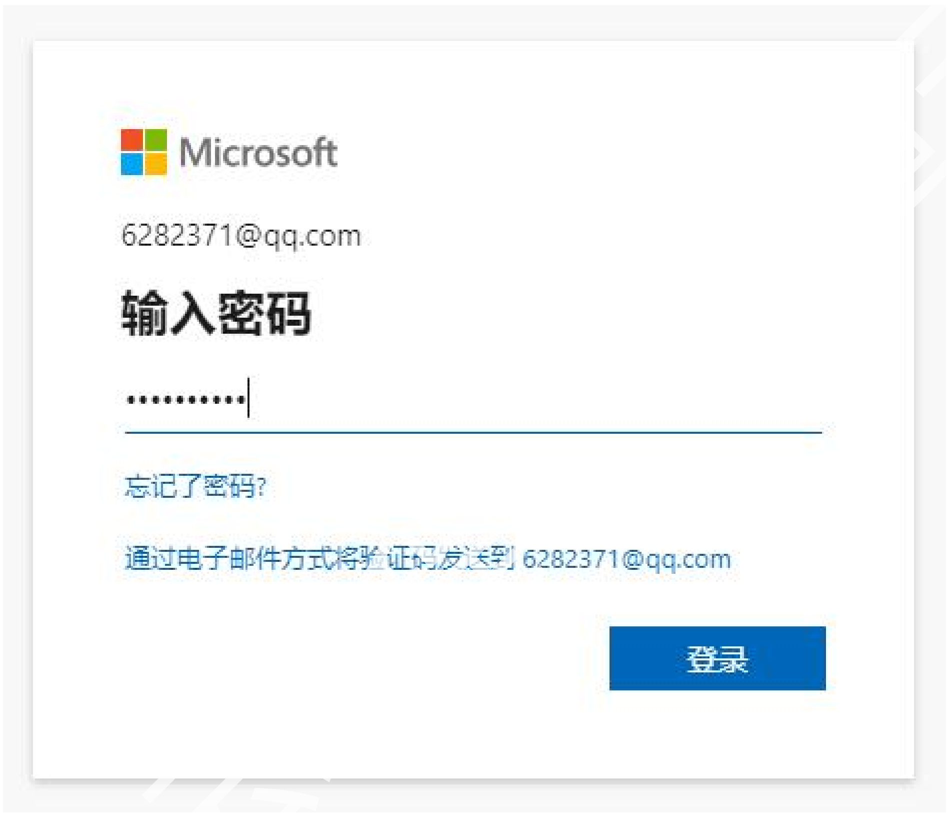
下一步

使用 GitHub 登录

登录选项

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



6282371@qq.com

输入密码

••••••••••

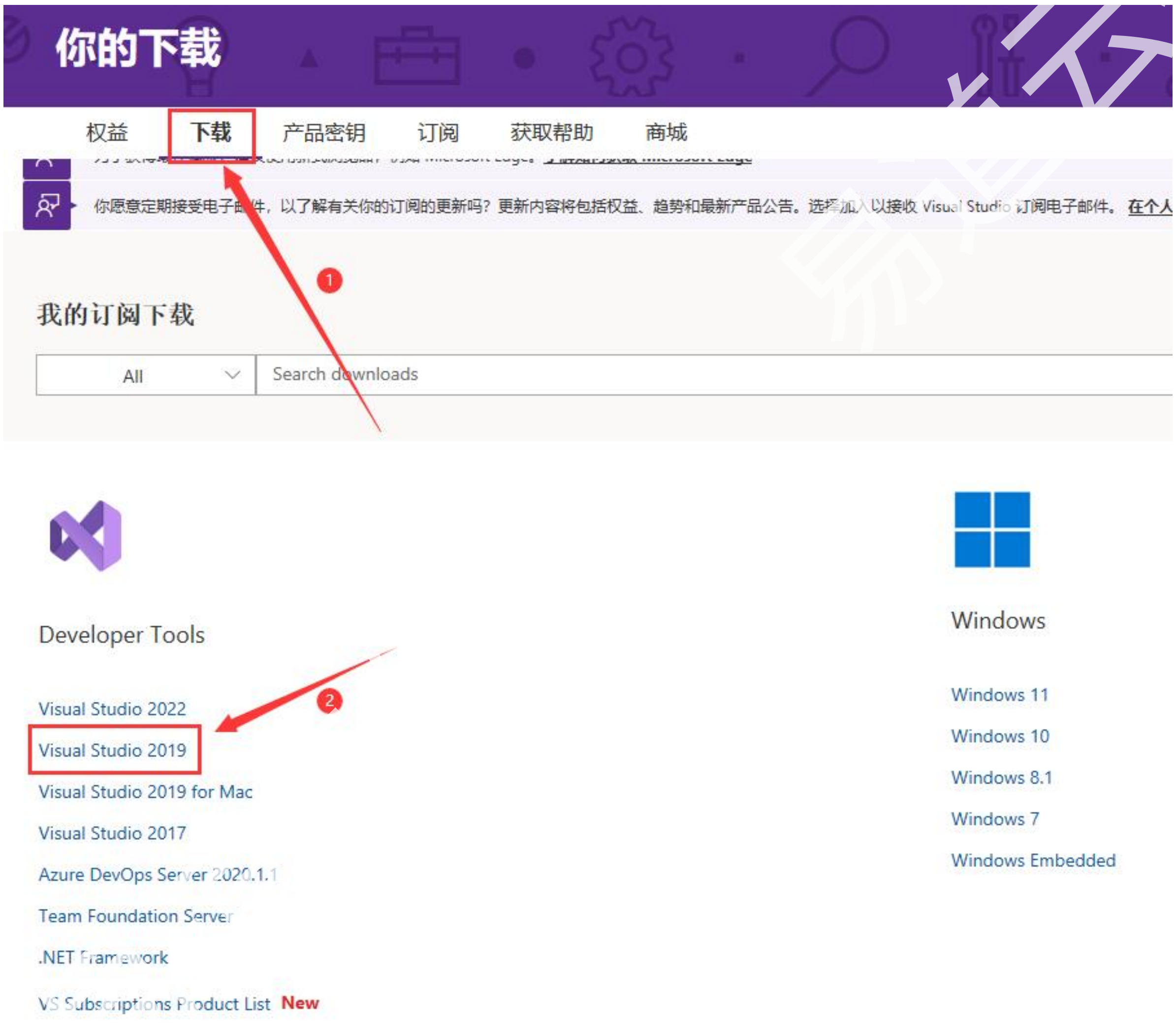
忘记了密码?

通过电子邮件方式将验证码发送到 6282371@qq.com

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



你的下载

权益 下载 产品密钥 订阅 获取帮助 商城

你愿意定期接受电子邮件，以了解有关你的订阅的更新吗？更新内容将包括权益、趋势和最新产品公告。选择加入以接收 Visual Studio 订阅电子邮件。在个人

我的订阅下载

All | Search downloads

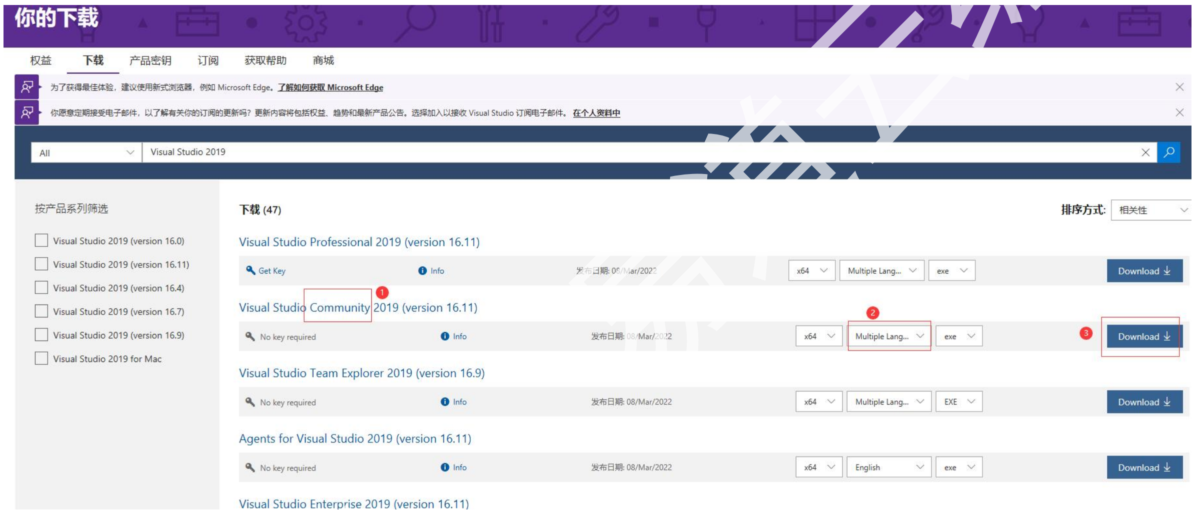
Developer Tools

Visual Studio 2022
Visual Studio 2019
Visual Studio 2019 for Mac
Visual Studio 2017
Azure DevOps Server 2020.1.1
Team Foundation Server
.NET Framework
VS Subscriptions Product List New

Windows

Windows 11
Windows 10
Windows 8.1
Windows 7
Windows Embedded

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



你的下载

权益 下载 产品密钥 订阅 获取帮助 商城

为了获得最佳体验，建议使用新式浏览器，例如 Microsoft Edge。了解如何获取 Microsoft Edge

你愿意定期接受电子邮件，以了解有关你的订阅的更新吗？更新内容将包括权益、趋势和最新产品公告。选择加入以接收 Visual Studio 订阅电子邮件。在个人资料中

All | Visual Studio 2019

按产品系列筛选

- Visual Studio 2019 (version 16.0)
- Visual Studio 2019 (version 16.11)
- Visual Studio 2019 (version 16.4)
- Visual Studio 2019 (version 16.7)
- Visual Studio 2019 (version 16.9)
- Visual Studio 2019 for Mac

下载 (47) 排序方式: 相关性

Visual Studio Professional 2019 (version 16.11)
Get Key | Info | 发布日期: 08/Mar/2022 | x64 | Multiple Lang... | exe | Download

Visual Studio Community 2019 (version 16.11)
No key required | Info | 发布日期: 08/Mar/2022 | x64 | Multiple Lang... | exe | Download

Visual Studio Team Explorer 2019 (version 16.9)
No key required | Info | 发布日期: 08/Mar/2022 | x64 | Multiple Lang... | EXE | Download

Agents for Visual Studio 2019 (version 16.11)
No key required | Info | 发布日期: 08/Mar/2022 | x64 | English | exe | Download

Visual Studio Enterprise 2019 (version 16.11)

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。

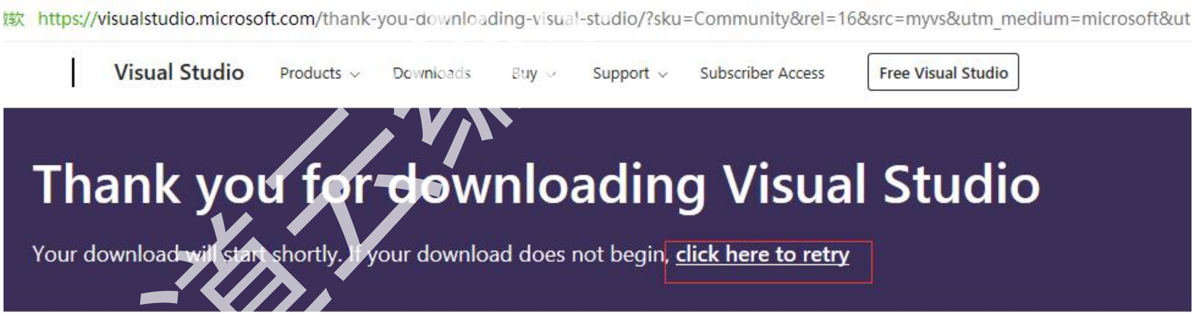
初学者强烈建议使用该版本！！！

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！

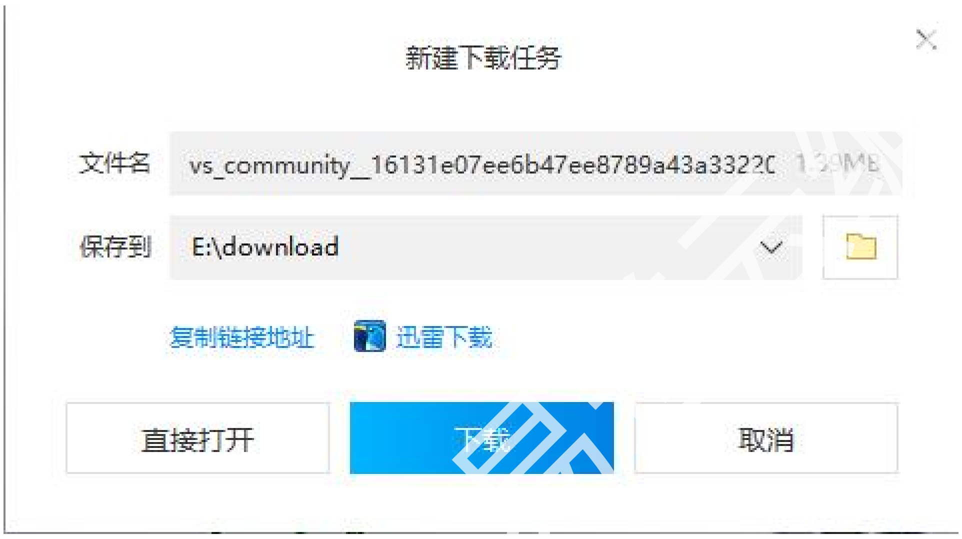
如果下载没有开始，并且长时间停留在这个页面，那么可以点击下图中的连接，再次激发下载


https://visualstudio.microsoft.com/thank-you-downloading-visual-studio/?sku=Community&rel=16&src=myvs&utm_medium=microsoft&ut

Thank you for downloading Visual Studio

Your download will start shortly. If your download does not begin, click here to retry

可以看到下载界面


新建下载任务

文件名 vs_community__16131e07ee6b47ee8789a43a33220 1.39MB

保存到 E:\download

下载完成后，点击程序开始安装

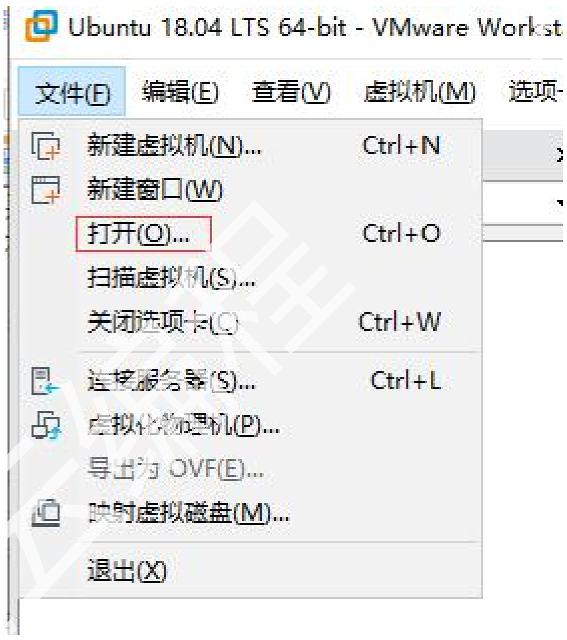
注意，不要随意修改安装路径

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！！！**

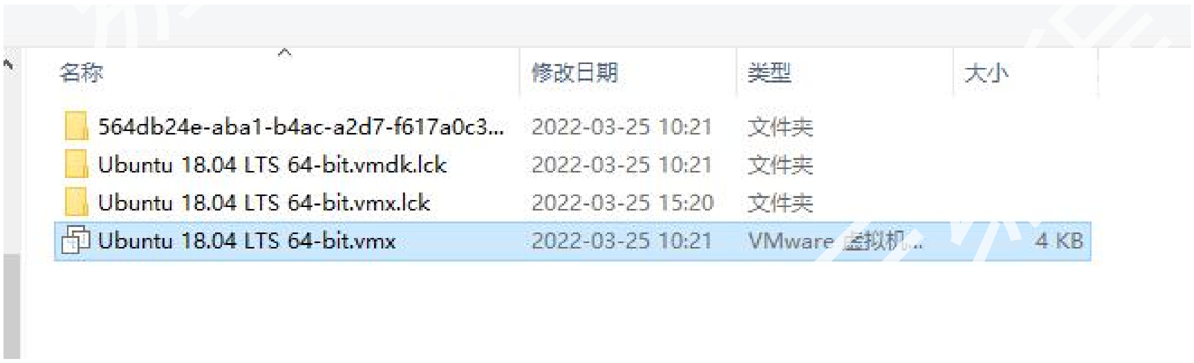
2019的下载就介绍到这里，详细的安装过程，可以参考VS的安装视频。

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

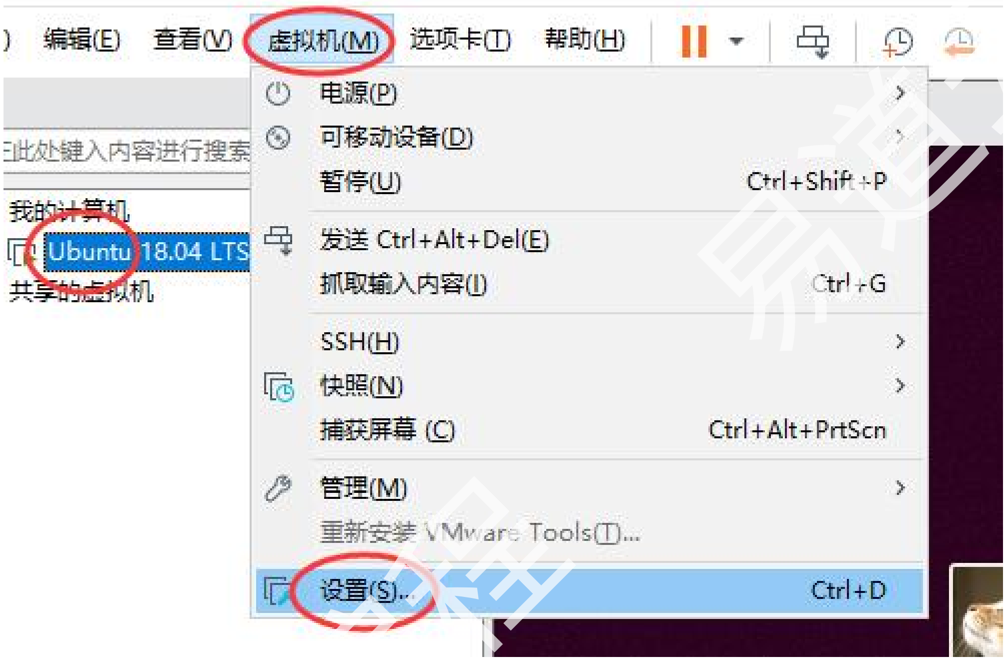


名称	修改日期	类型	大小
564db24e-aba1-b4ac-a2d7-f617a0c3...	2022-03-25 10:21	文件夹	
Ubuntu 18.04 LTS 64-bit.vmdk.lck	2022-03-25 10:21	文件夹	
Ubuntu 18.04 LTS 64-bit.vmx.lck	2022-03-25 15:20	文件夹	
Ubuntu 18.04 LTS 64-bit.vmx	2022-03-25 10:21	VMware 虚拟机...	4 KB

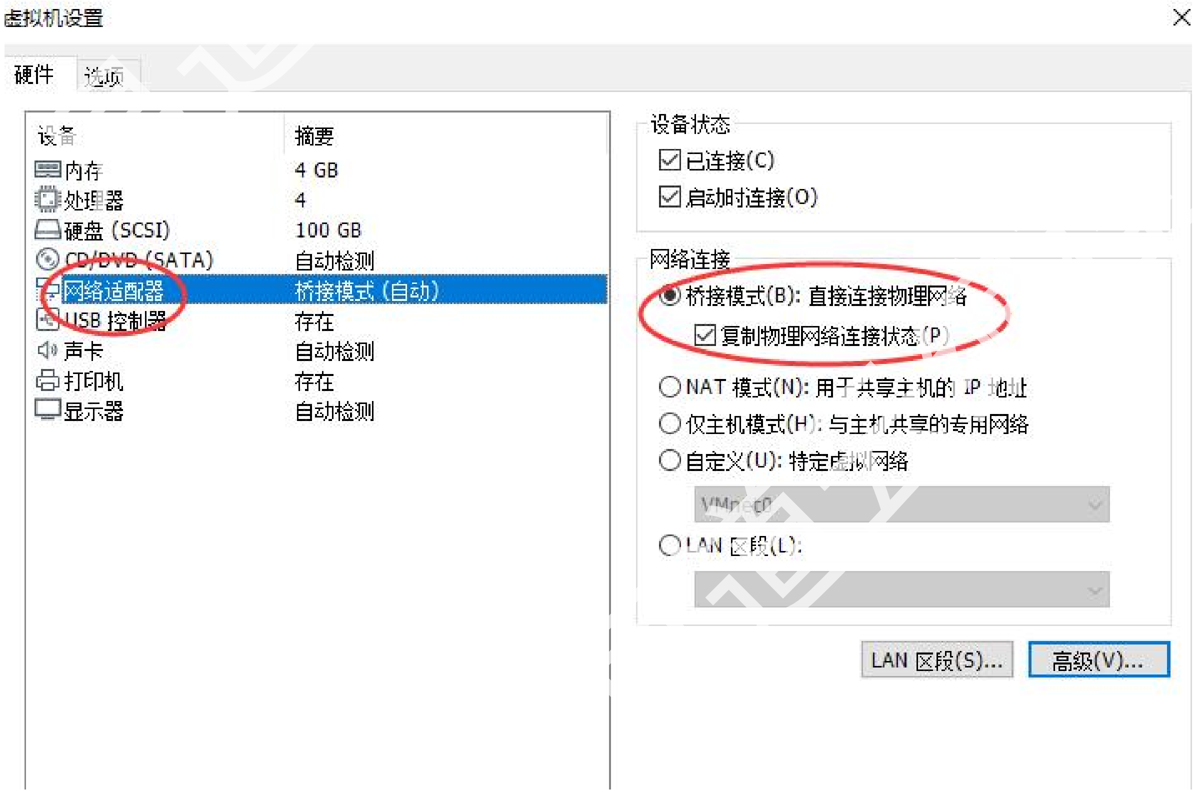
注意，VM会问你虚拟机是哪里来的。一个是复制来的，一个是移动来的。

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：


虚拟机设置

硬件　选项

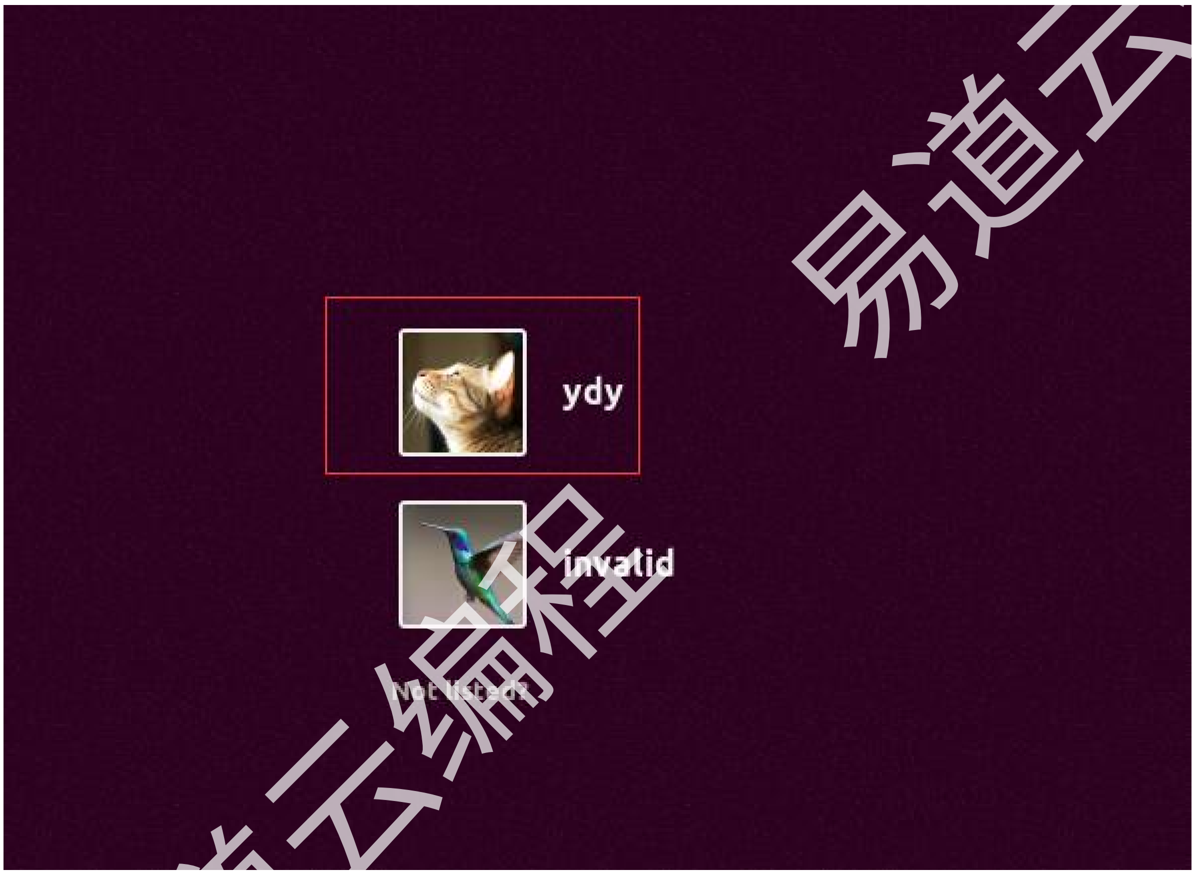
设备	摘要
内存	4 GB
处理器	4
硬盘 (SCSI)	100 GB
CD/DVD (SATA)	自动检测
网络适配器	桥接模式 (自动)
USB 控制器	存在
声卡	自动检测
打印机	存在
显示器	自动检测

设备状态
- ☑ 已连接(C)
- ☑ 启动时连接(O)

网络连接
- ◉ 桥接模式(B): 直接连接物理网络
  - ☑ 复制物理网络连接状态(P)
- ○ NAT 模式(N): 用于共享主机的 IP 地址
- ○ 仅主机模式(H): 与主机共享的专用网络
- ○ 自定义(U): 特定虚拟网络
  - VMnet0
- ○ LAN 区段(L):

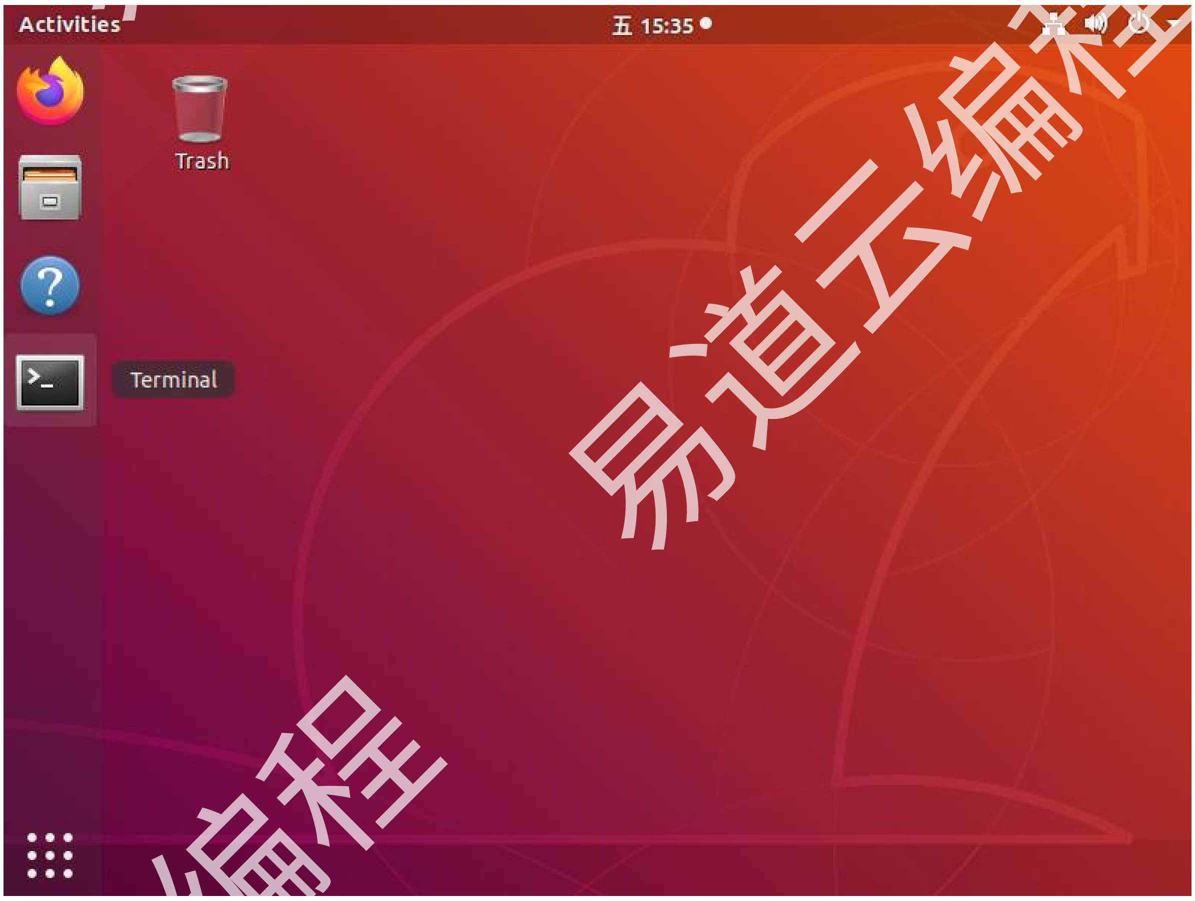
另外依据自己的机器调整**内存大小**和**处理器数量**

然后启动虚拟机，来到登录界面：

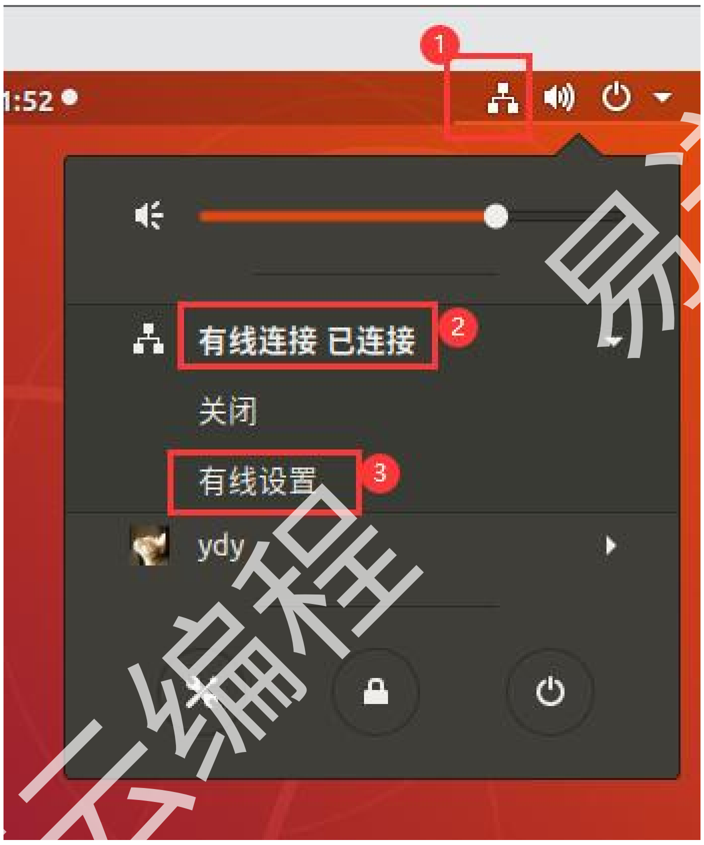


点击ydy，输入密码ydy619619进行登录

登录后，大概是这个样子：

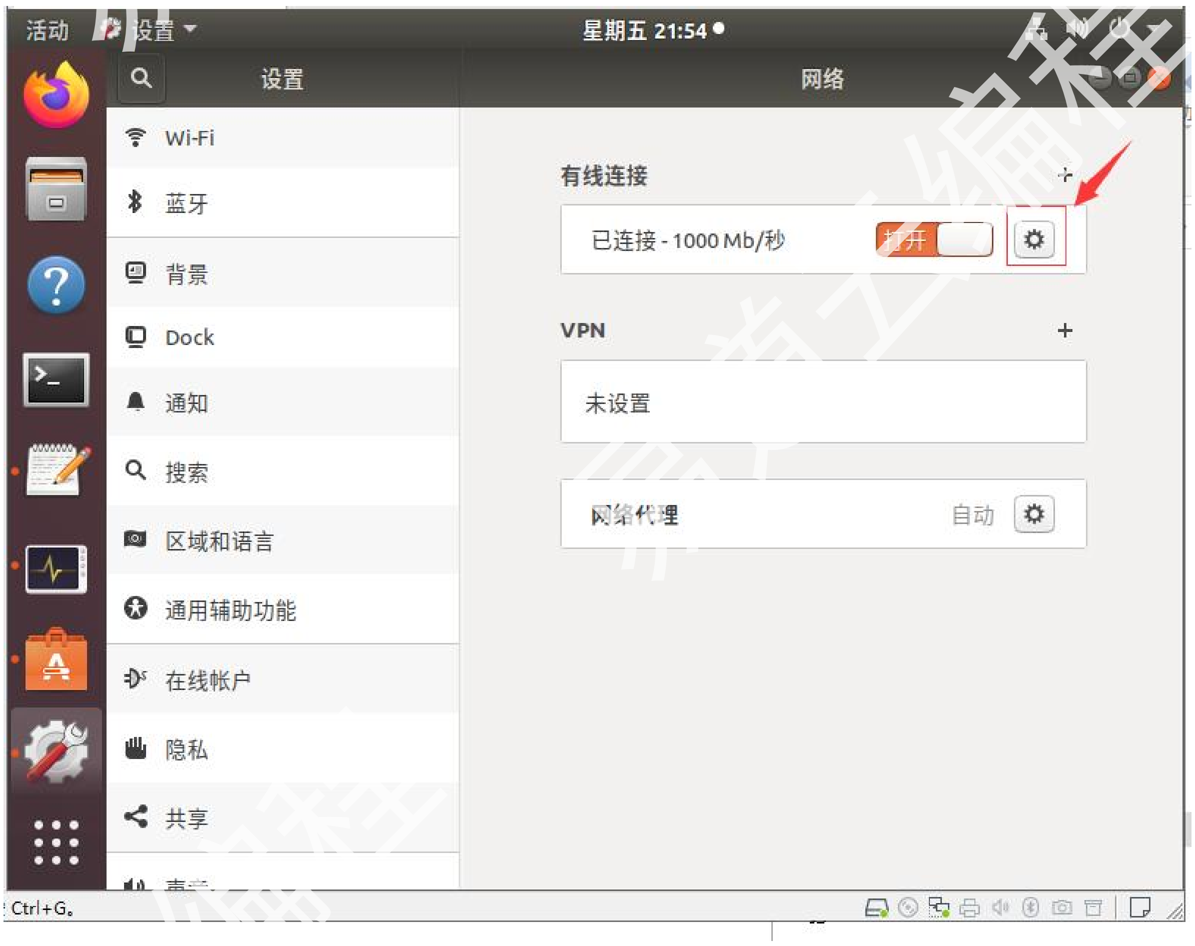


点击左上角的网络按钮

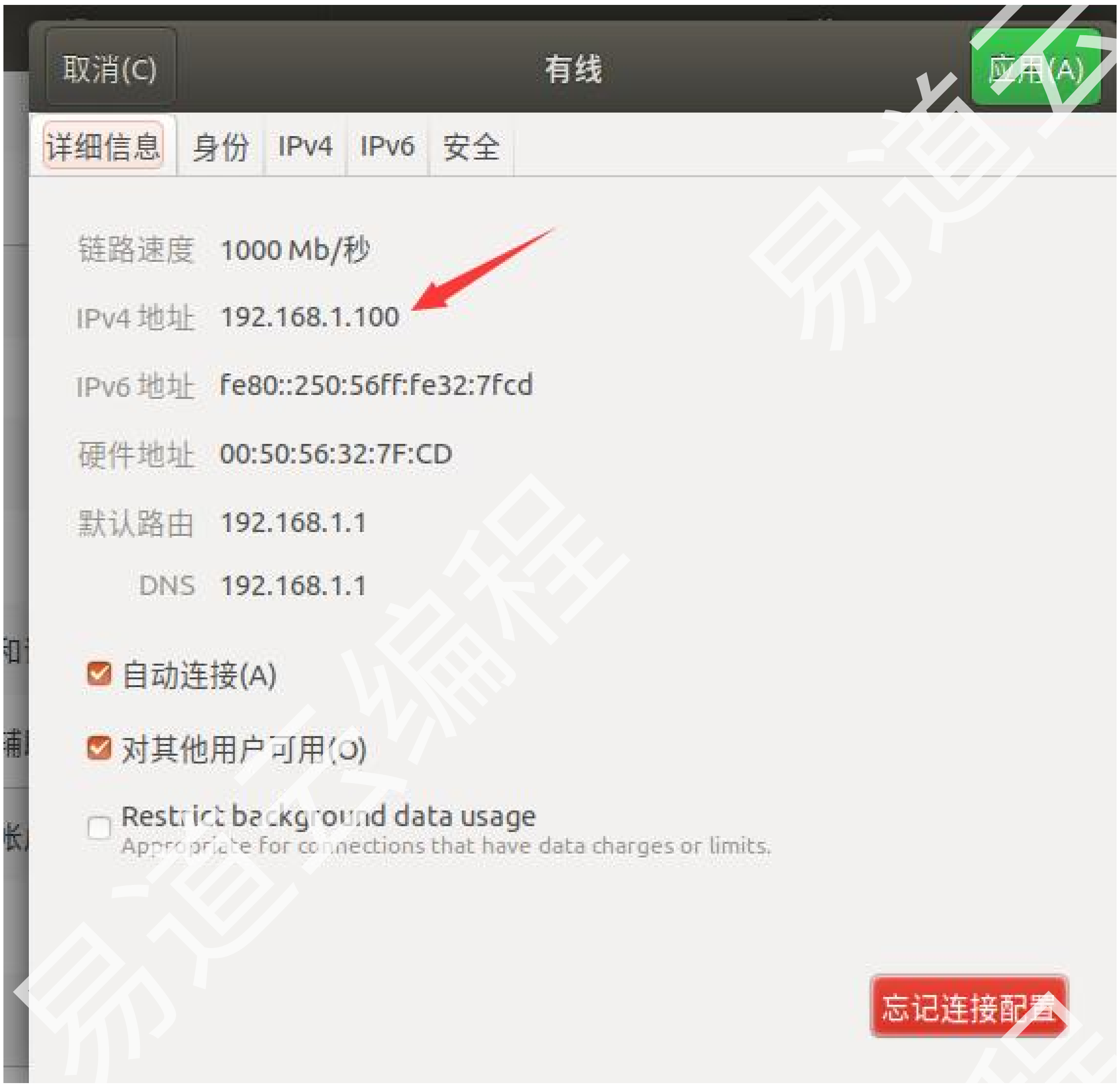


点击有线连接→有线设置

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



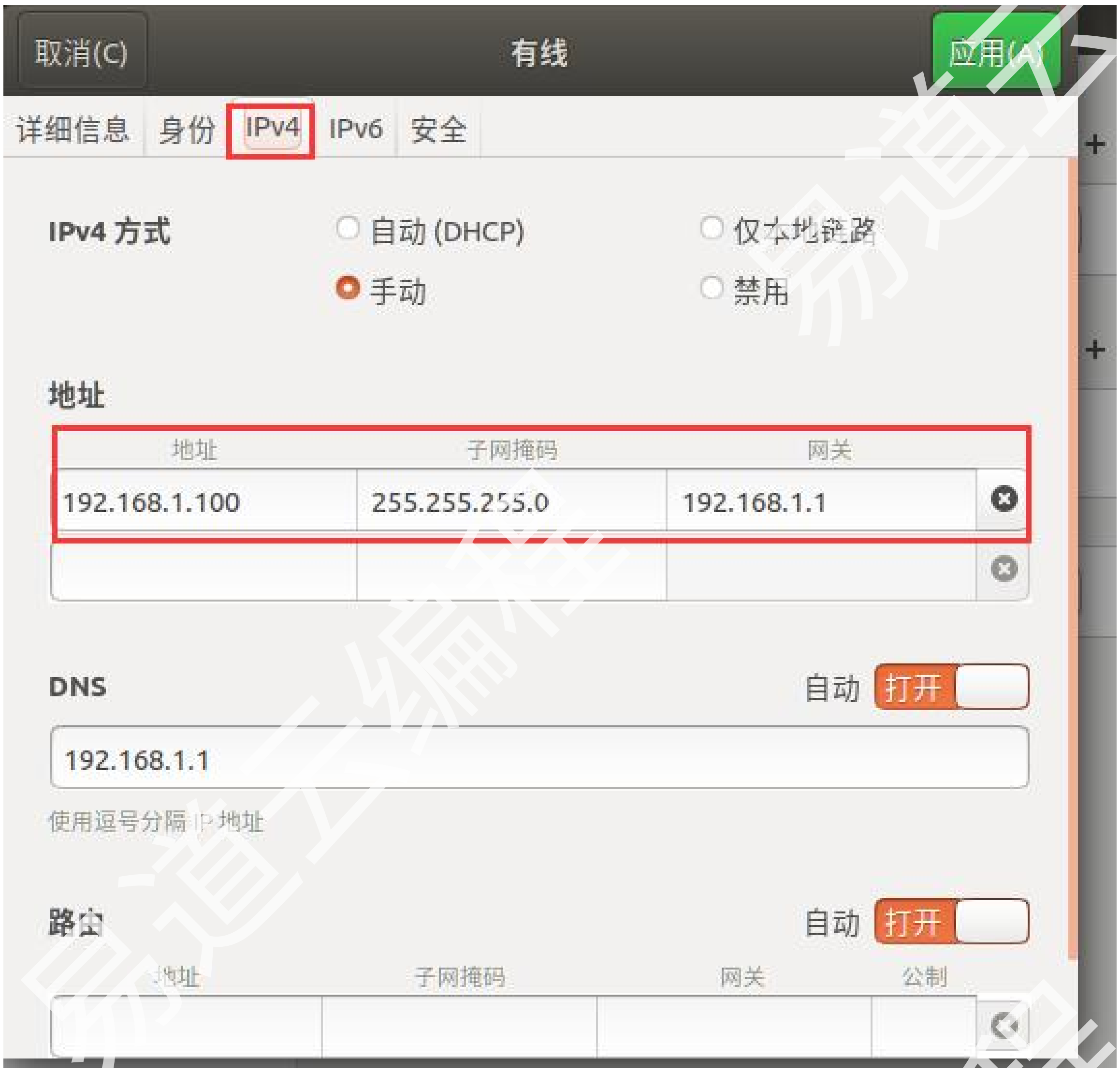
链路速度 1000 Mb/秒
IPv4 地址 192.168.1.100
IPv6 地址 fe80::250:56ff:fe32:7fcd
硬件地址 00:50:56:32:7F:CD
默认路由 192.168.1.1
DNS 192.168.1.1

☑ 自动连接(A)
☑ 对其他用户可用(O)
☐ Restrict background data usage
Appropriate for connections that have data charges or limits.

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



IPv4 方式: 自动 (DHCP) [✓ 手动] 仅本地链路 禁用

地址

地址	子网掩码	网关
192.168.1.100	255.255.255.0	192.168.1.1

DNS 自动 打开

192.168.1.1

使用逗号分隔 IP 地址

路由 自动 打开

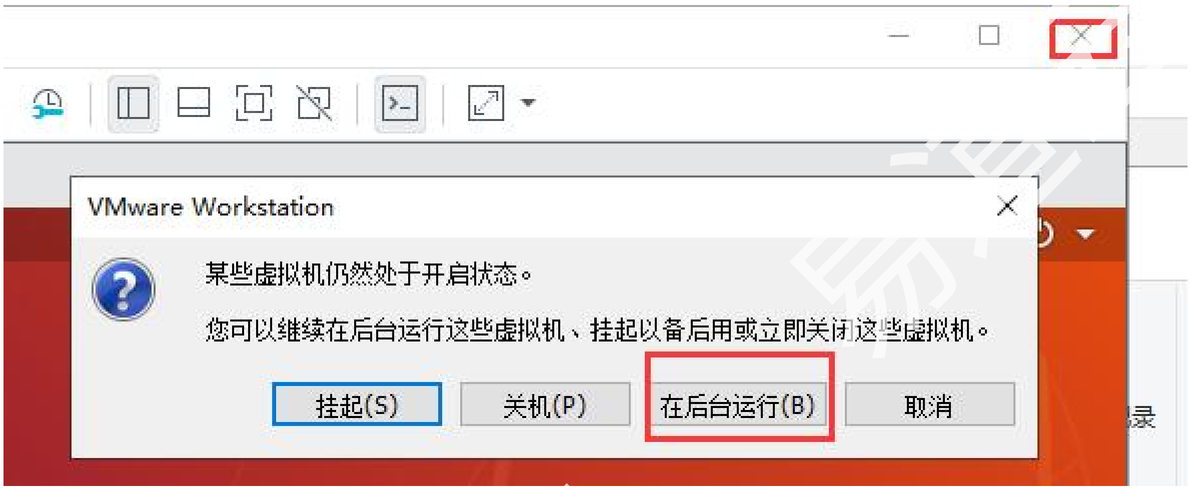
地址	子网掩码	网关	公制

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


VMware Workstation

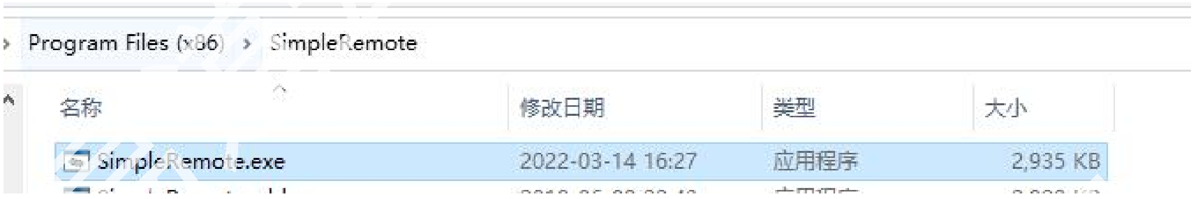
某些虚拟机仍然处于开启状态。

您可以继续在后台运行这些虚拟机、挂起以备后用或立即关闭这些虚拟机。

如上图所示。

这样可以节约一定的系统资源。

SimpleRemote的使用

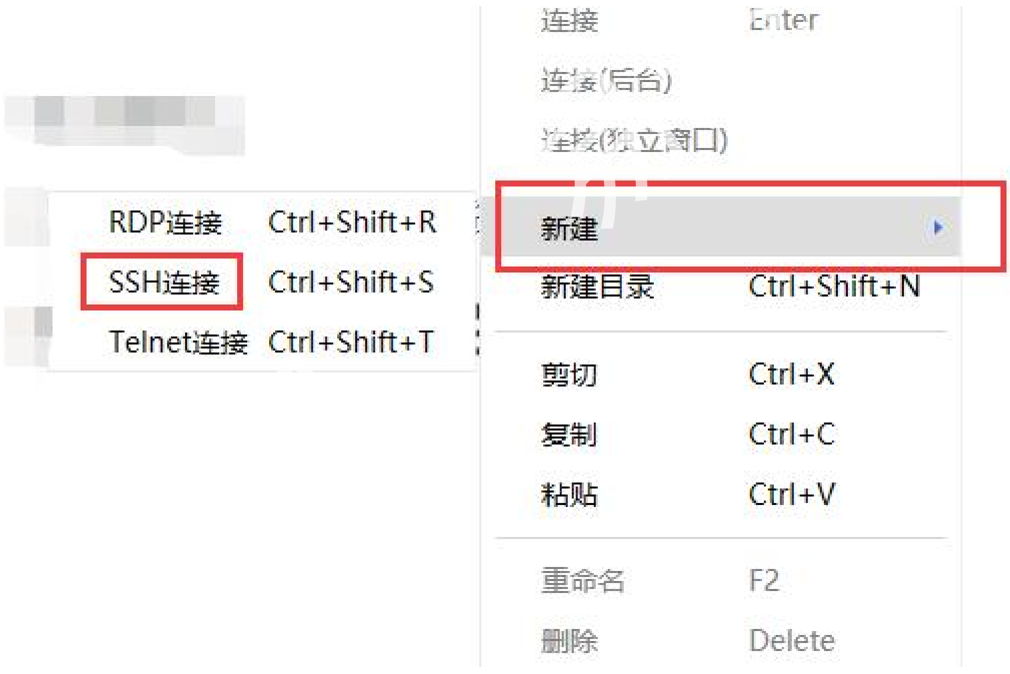

Program Files (x86) › SimpleRemote

名称	修改日期	类型	大小
SimpleRemote.exe	2022-03-14 16:27	应用程序	2,935 KB

在任意目录建立一个SimpleRemote的文件夹

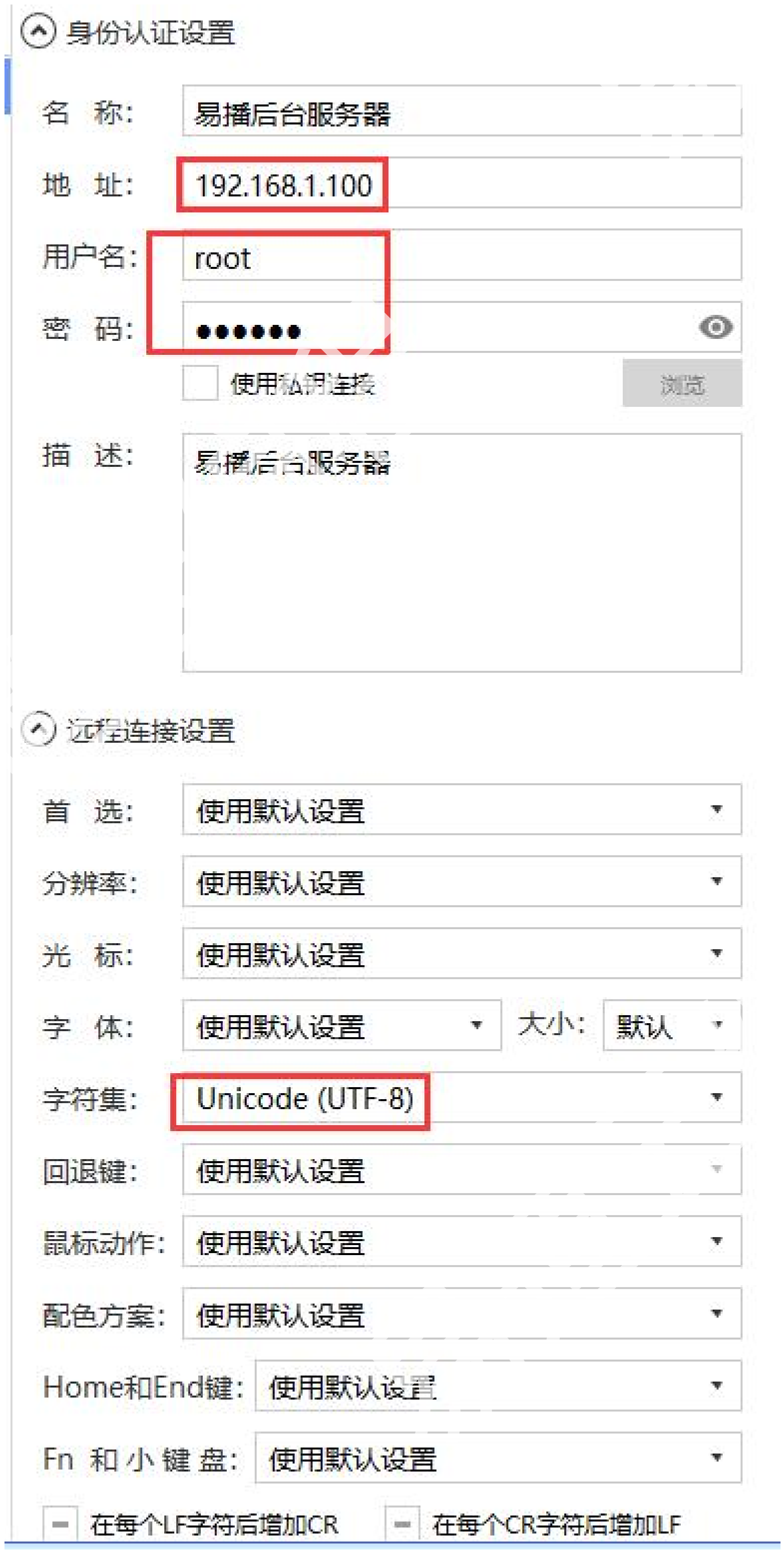
然后将SimpleRemote.exe复制到该文件夹，即完成SimpleRemote的安装

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：



身份认证设置

名　称：易播后台服务器

地　址：192.168.1.100

用户名：root

密　码：●●●●●●

使用私钥连接　浏览

描　述：易播后台服务器

远程连接设置

首　选：使用默认设置

分辨率：使用默认设置

光　标：使用默认设置

字　体：使用默认设置　大小：默认

字符集：Unicode (UTF-8)

回退键：使用默认设置

鼠标动作：使用默认设置

配色方案：使用默认设置

Home和End键：使用默认设置

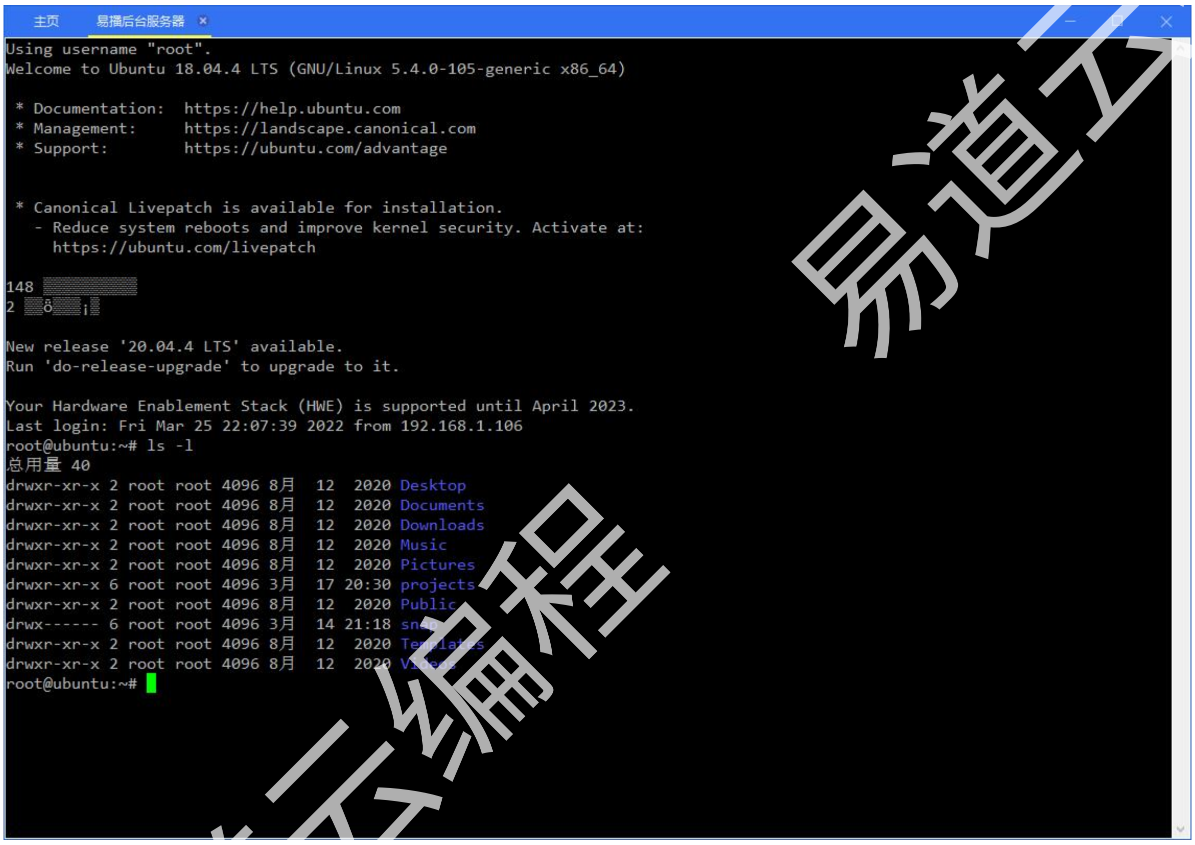
Fn 和 小 键 盘：使用默认设置

在每个LF字符后增加CR　在每个CR字符后增加LF

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：



```
Using username "root".
Welcome to Ubuntu 18.04.4 LTS (GNU/Linux 5.4.0-105-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage


 * Canonical Livepatch is available for installation.
   - Reduce system reboots and improve kernel security. Activate at:
     https://ubuntu.com/livepatch

148 ▒▒▒▒▒▒▒▒▒▒
2 ▒▒ő▒▒▒¡▒

New release '20.04.4 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Your Hardware Enablement Stack (HWE) is supported until April 2023.
Last login: Fri Mar 25 22:07:39 2022 from 192.168.1.106
root@ubuntu:~# ls -l
总用量 40
drwxr-xr-x 2 root root 4096 8月  12  2020 Desktop
drwxr-xr-x 2 root root 4096 8月  12  2020 Documents
drwxr-xr-x 2 root root 4096 8月  12  2020 Downloads
drwxr-xr-x 2 root root 4096 8月  12  2020 Music
drwxr-xr-x 2 root root 4096 8月  12  2020 Pictures
drwxr-xr-x 6 root root 4096 3月  17 20:30 projects
drwxr-xr-x 2 root root 4096 8月  12  2020 Public
drwx------ 6 root root 4096 3月  14 21:18 snap
drwxr-xr-x 2 root root 4096 8月  12  2020 Templates
drwxr-xr-x 2 root root 4096 8月  12  2020 Videos
root@ubuntu:~#
```

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`

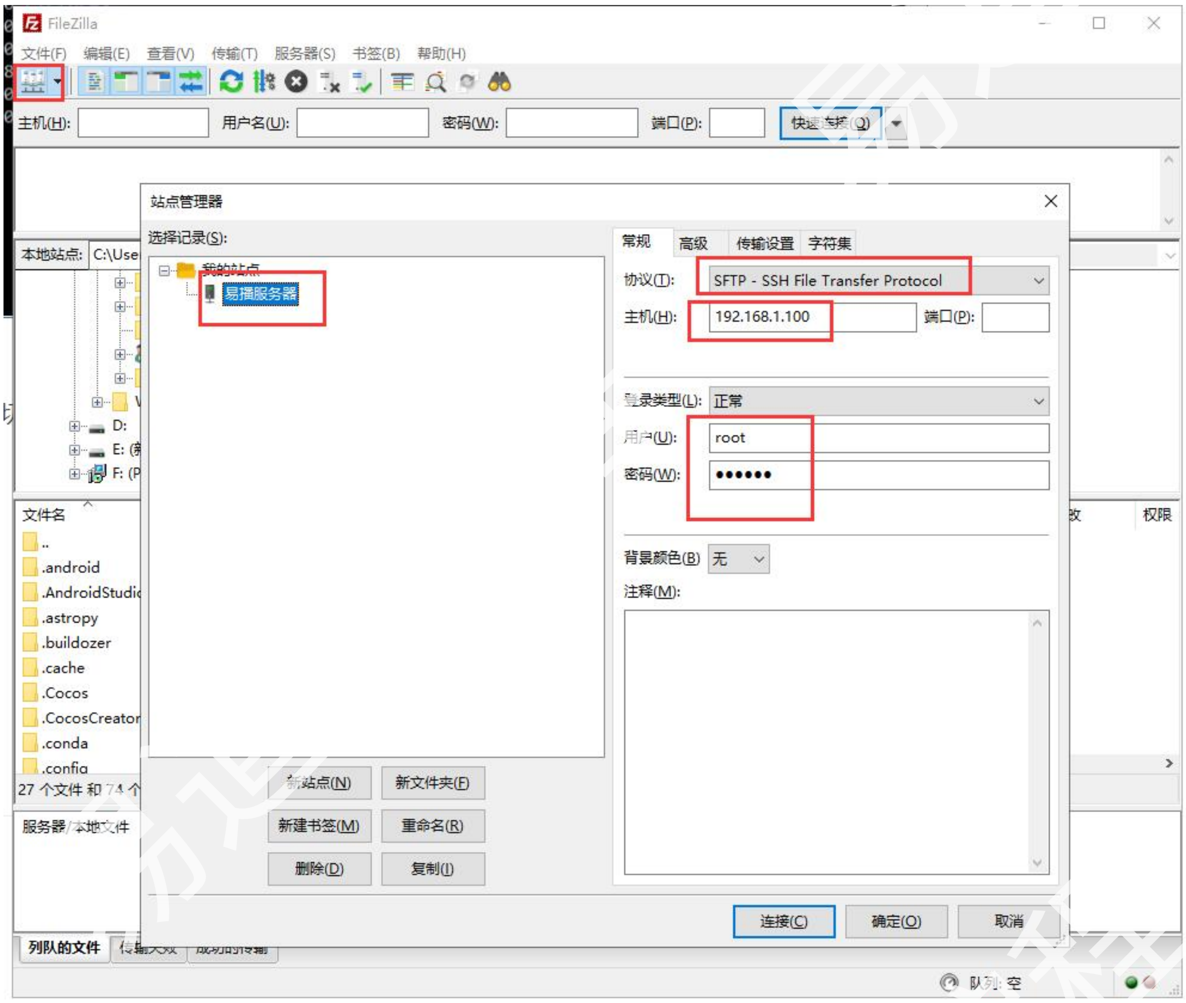
则虚拟机会在一分钟内（依据主机性能来决定，可能会超过一分钟）自动关闭。

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）


FileZilla
文件(F) 编辑(E) 查看(V) 传输(T) 服务器(S) 书签(B) 帮助(H)
主机(H): 用户名(U): 密码(W): 端口(P): 快速连接(Q)
站点管理器
新站点(N) 新文件夹(F) 新建书签(M) 重命名(R) 删除(D) 复制(I)
常规 高级 传输设置 字符集
连接(C) 确定(O) 取消
本地站点: C:\Use
文件名
权限
列队的文件 传输失败 成功的传输
队列: 空

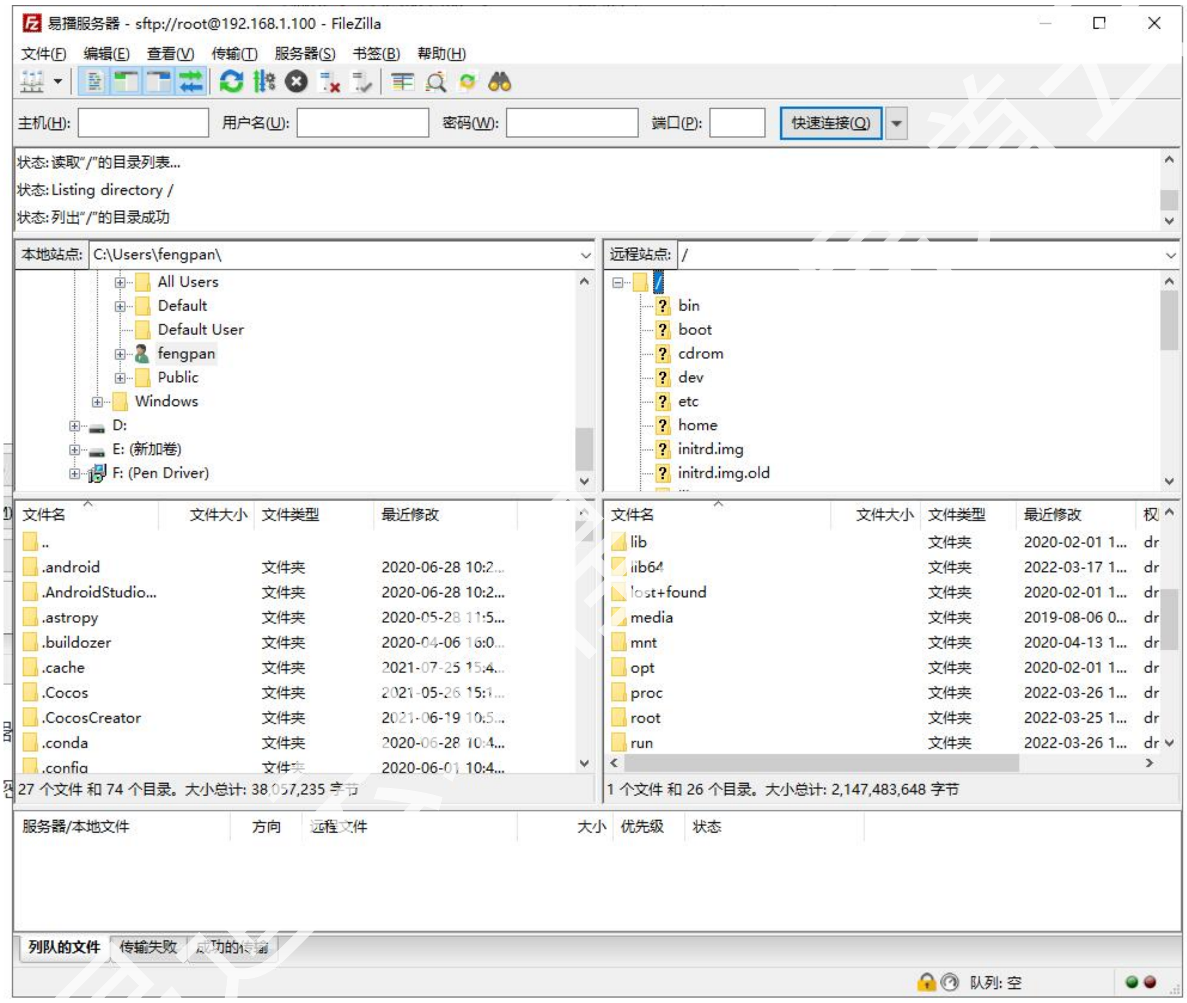
选择记录(S):
我的站点
易播服务器

协议(T): SFTP - SSH File Transfer Protocol
主机(H): 192.168.1.100 端口(P):
登录类型(L): 正常
用户(U): root
密码(W): ••••••
背景颜色(B) 无
注释(M):

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：


易播服务器 - sftp://root@192.168.1.100 - FileZilla
文件(F) 编辑(E) 查看(V) 传输(T) 服务器(S) 书签(B) 帮助(H)
主机(H):
用户名(U):
密码(W):
端口(P):
快速连接(Q)
服务器/本地文件 方向 远程文件 大小 优先级 状态
列队的文件
传输失败
成功的传输
队列: 空

状态: 读取"/"的目录列表...
状态: Listing directory /
状态: 列出"/"的目录成功

本地站点: C:\Users\fengpan\

- All Users
- Default
- Default User
- fengpan
- Public
- Windows
- D:
- E: (新加卷)
- F: (Pen Driver)

文件名	文件大小	文件类型	最近修改
..			
.android		文件夹	2020-06-28 10:2...
.AndroidStudio...		文件夹	2020-06-28 10:2...
.astropy		文件夹	2020-05-28 11:5...
.buildozer		文件夹	2020-04-06 16:0...
.cache		文件夹	2021-07-25 15:4...
.Cocos		文件夹	2021-05-26 15:1...
.CocosCreator		文件夹	2021-06-19 10:5...
.conda		文件夹	2020-06-28 10:4...
.config		文件夹	2020-06-01 10:4...

27 个文件 和 74 个目录。大小总计: 38,057,235 字节

远程站点: /

- /
  - bin
  - boot
  - cdrom
  - dev
  - etc
  - home
  - initrd.img
  - initrd.img.old

文件名	文件大小	文件类型	最近修改	权
lib		文件夹	2020-02-01 1...	dr
lib64		文件夹	2022-03-17 1...	dr
lost+found		文件夹	2020-02-01 1...	dr
media		文件夹	2019-08-06 0...	dr
mnt		文件夹	2020-04-13 1...	dr
opt		文件夹	2020-02-01 1...	dr
proc		文件夹	2022-03-26 1...	dr
root		文件夹	2022-03-25 1...	dr
run		文件夹	2022-03-26 1...	dr

1 个文件 和 26 个目录。大小总计: 2,147,483,648 字节

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。

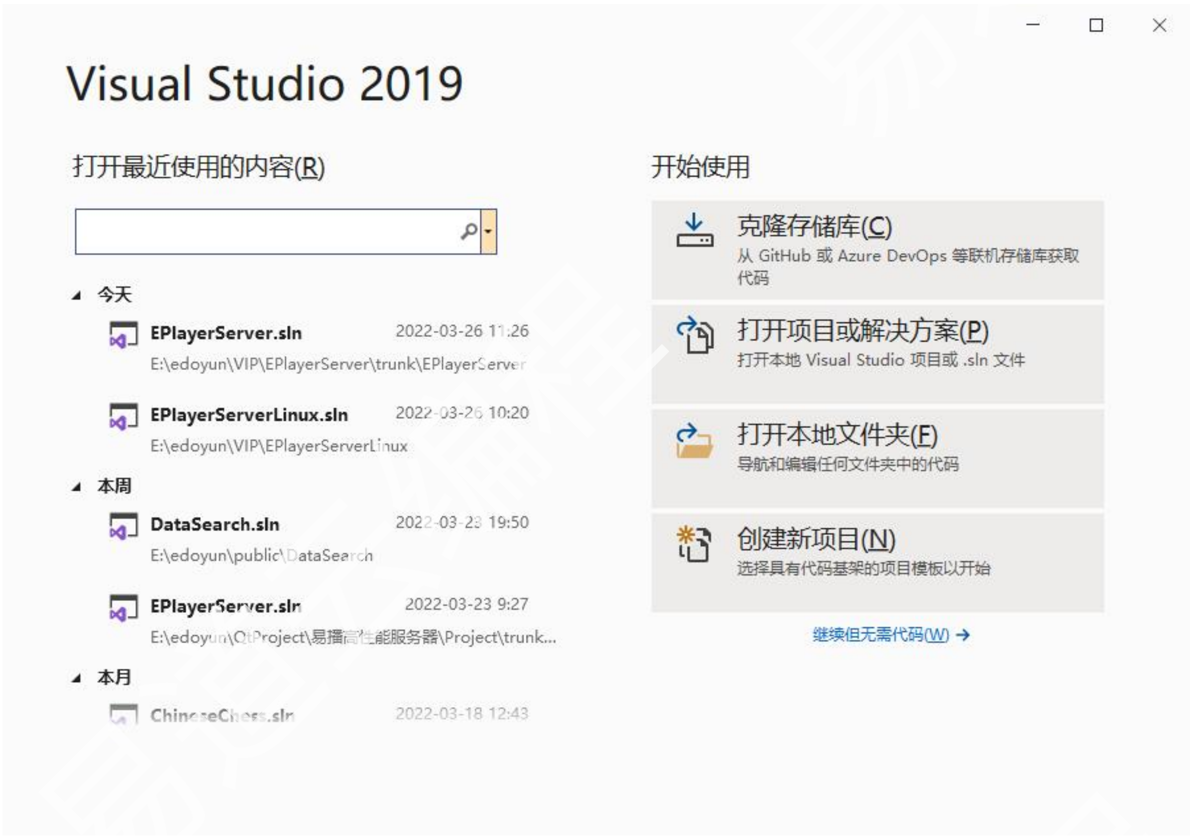
项目的开发

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



Visual Studio 2019

打开最近使用的内容(R)

- 今天
  - EPlayerServer.sln　2022-03-26 11:26　E:\edoyun\VIP\EPlayerServer\trunk\EPlayerServer
  - EPlayerServerLinux.sln　2022-03-26 10:20　E:\edoyun\VIP\EPlayerServerLinux
- 本周
  - DataSearch.sln　2022-03-23 19:50　E:\edoyun\public\DataSearch
  - EPlayerServer.sln　2022-03-23 9:27　E:\edoyun\QtProject\易播高性能服务器\Project\trunk...
- 本月
  - ChineseChess.sln　2022-03-18 12:43

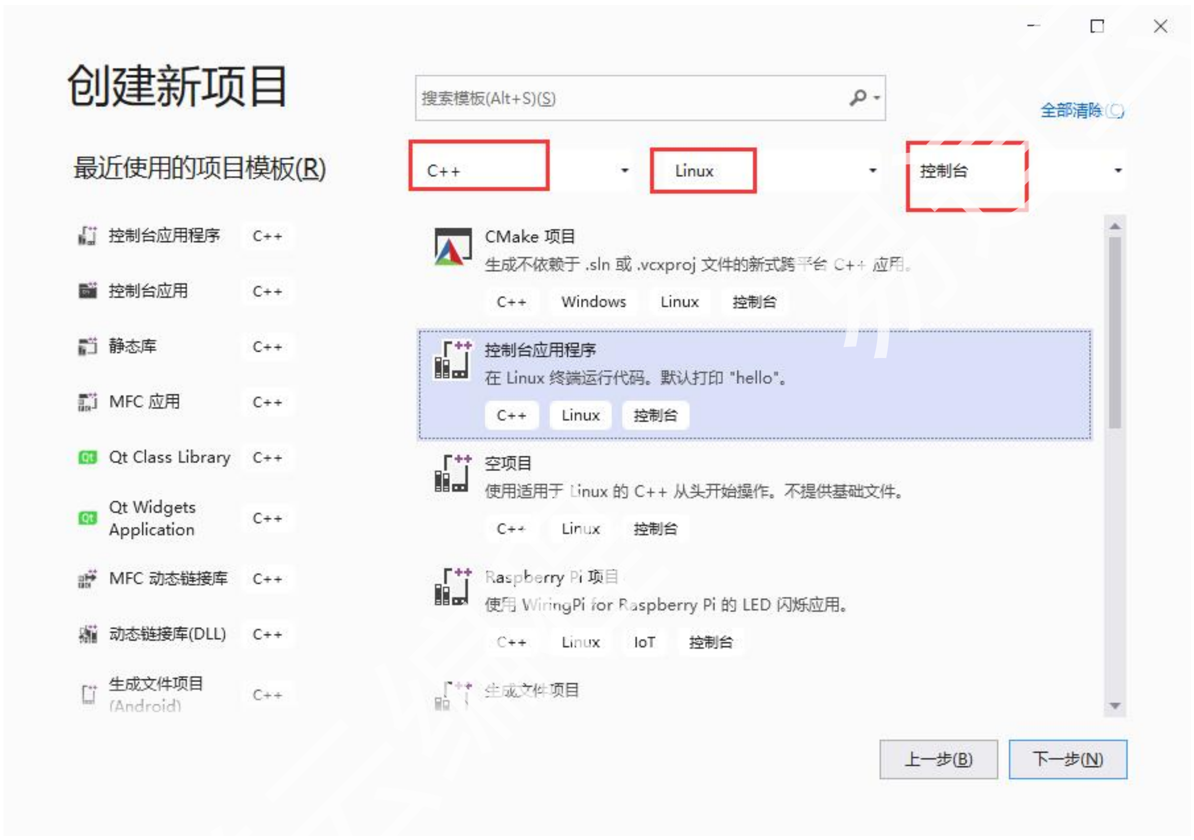
开始使用

- 克隆存储库(C)　从 GitHub 或 Azure DevOps 等联机存储库获取代码
- 打开项目或解决方案(P)　打开本地 Visual Studio 项目或 .sln 文件
- 打开本地文件夹(F)　导航和编辑任何文件夹中的代码
- 创建新项目(N)　选择具有代码基架的项目模板以开始

继续但无需代码(W) →

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**


创建新项目

最近使用的项目模板(R)

控制台应用程序	C++
控制台应用	C++
静态库	C++
MFC 应用	C++
Qt Class Library	C++
Qt Widgets Application	C++
MFC 动态链接库	C++
动态链接库(DLL)	C++
生成文件项目 (Android)	C++

搜索模板(Alt+S)(S)　全部清除

C++　Linux　控制台

CMake 项目
生成不依赖于 .sln 或 .vcxproj 文件的新式跨平台 C++ 应用。
C++ Windows Linux 控制台

控制台应用程序
在 Linux 终端运行代码。默认打印 "hello"。
C++ Linux 控制台

空项目
使用适用于 Linux 的 C++ 从头开始操作。不提供基础文件。
C++ Linux 控制台

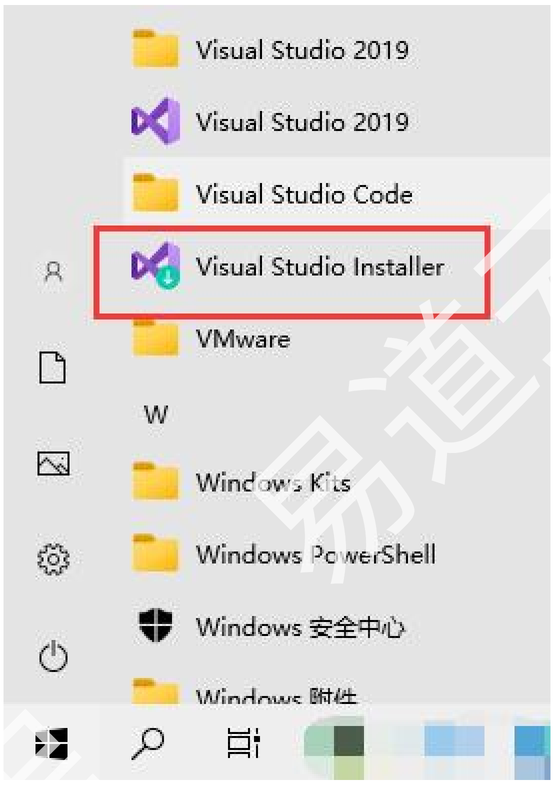
Raspberry Pi 项目
使用 WiringPi for Raspberry Pi 的 LED 闪烁应用。
C++ Linux IoT 控制台

生成文件项目

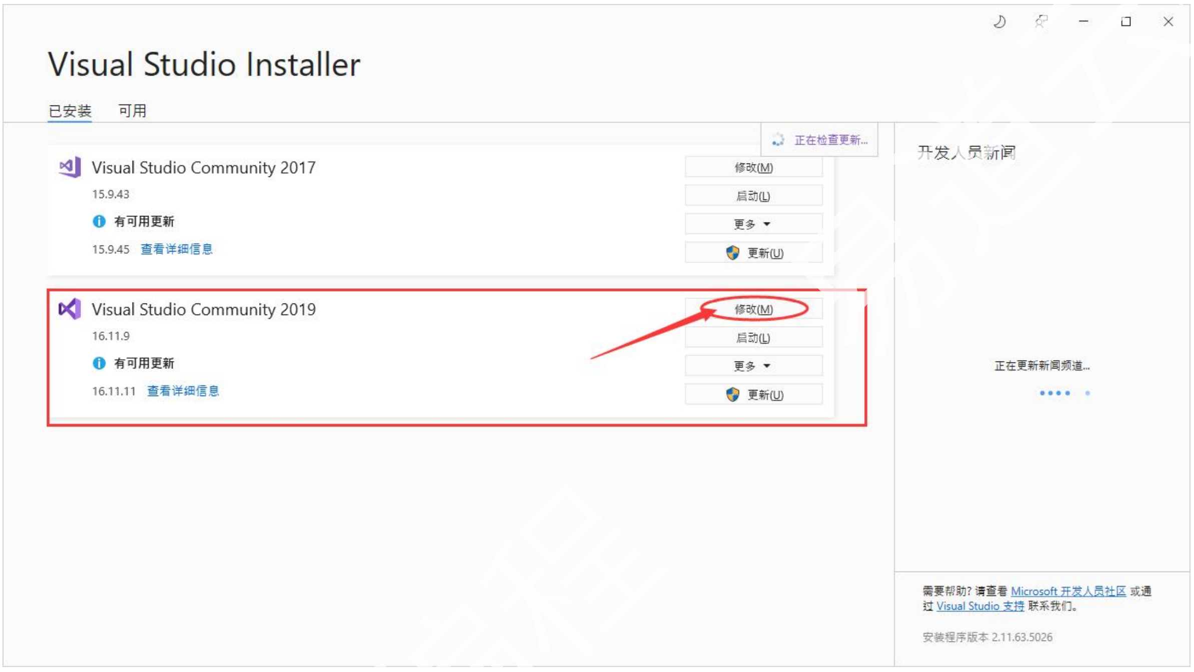
上一步(B)　下一步(N)

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

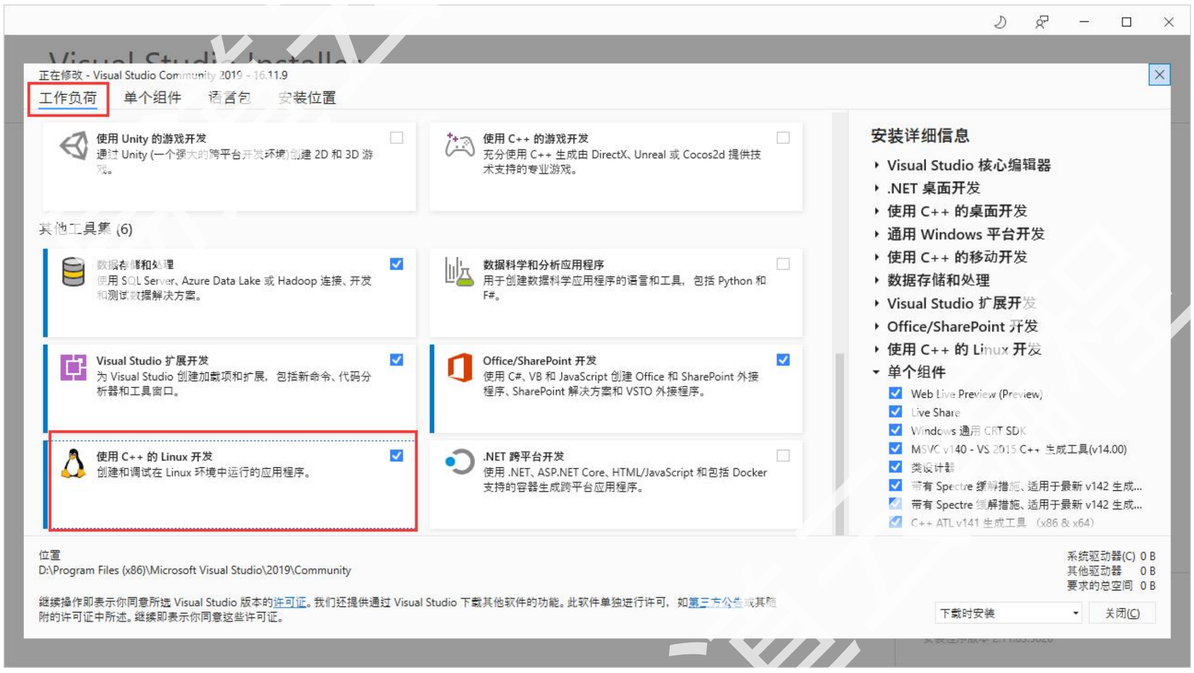


打开后，找到2019的社区版，点击修改：



需要帮助? 请查看 Microsoft 开发人员社区 或通过 Visual Studio 支持 联系我们。

然后看看**使用C++的Linux开发**是否勾选：



通过 Unity (一个强大的跨平台开发环境) 创建 2D 和 3D 游戏。

充分使用 C++ 生成由 DirectX、Unreal 或 Cocos2d 提供技术支持的专业游戏。

使用 SQL Server、Azure Data Lake 或 Hadoop 连接、开发和测试数据解决方案。

用于创建数据科学应用程序的语言和工具，包括 Python 和 F#。

为 Visual Studio 创建加载项和扩展，包括新命令、代码分析器和工具窗口。

使用 C#、VB 和 JavaScript 创建 Office 和 SharePoint 外接程序、SharePoint 解决方案和 VSTO 外接程序。

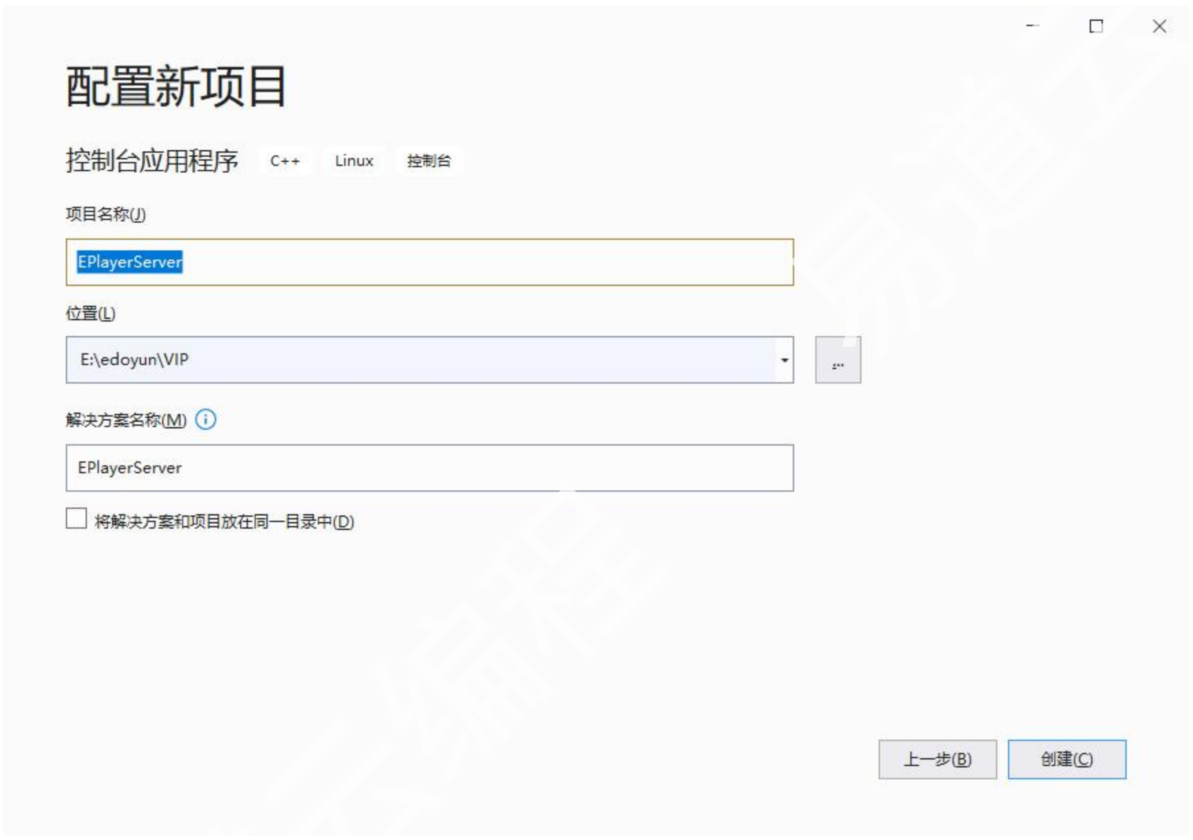
创建和调试在 Linux 环境中运行的应用程序。

使用 .NET、ASP.NET Core、HTML/JavaScript 和包括 Docker 支持的容器生成跨平台应用程序。

继续操作即表示你同意所选 Visual Studio 版本的许可证。我们还提供通过 Visual Studio 下载其他软件的功能。此软件单独进行许可，如第三方公告或其随附的许可证中所述。继续即表示你同意这些许可证。

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：


配置新项目

控制台应用程序 C++ Linux 控制台

项目名称(J)

EPlayerServer

位置(L)

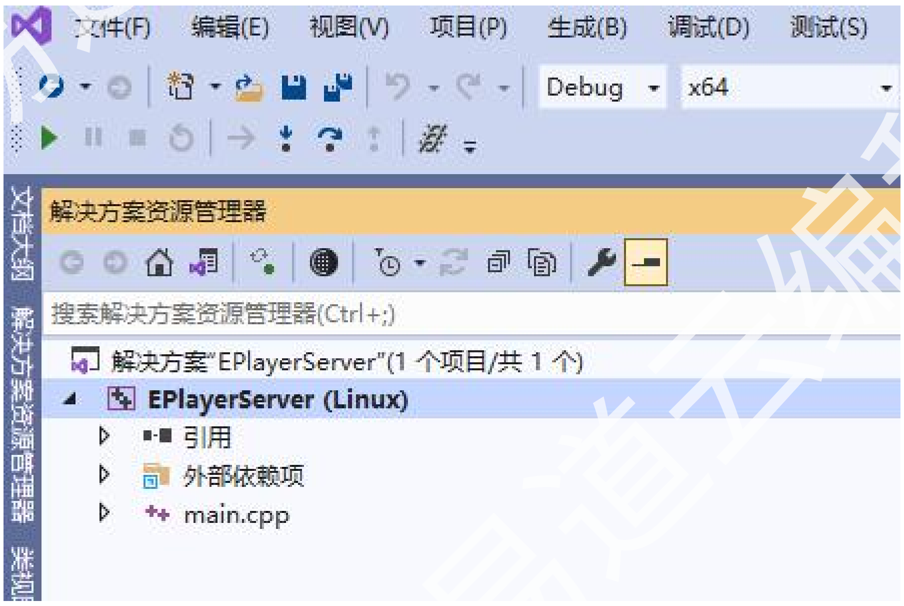
E:\edoyun\VIP

解决方案名称(M)

EPlayerServer

☐ 将解决方案和项目放在同一目录中(D)

然后点击创建按钮，即可进入项目目录


文件(F) 编辑(E) 视图(V) 项目(P) 生成(B) 调试(D) 测试(S)
Debug
x64
文档大纲
解决方案资源管理器
类视图
解决方案资源管理器
搜索解决方案资源管理器(Ctrl+;)

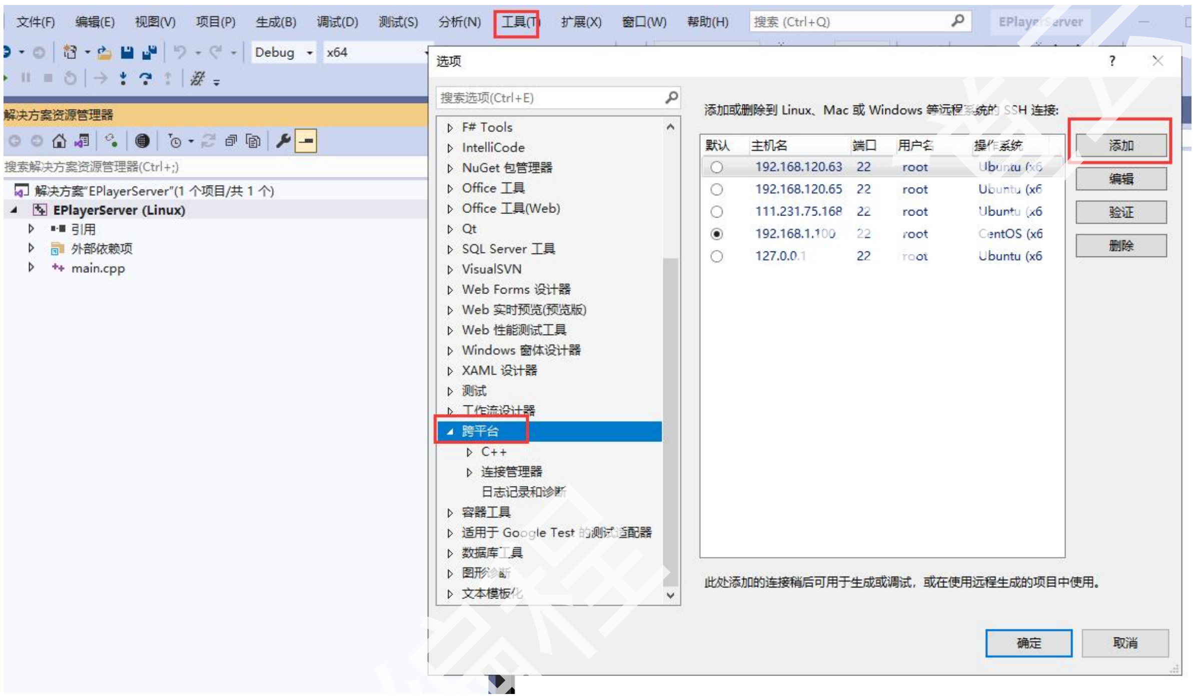
解决方案"EPlayerServer"(1 个项目/共 1 个)
EPlayerServer (Linux)
引用
外部依赖项
main.cpp

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**

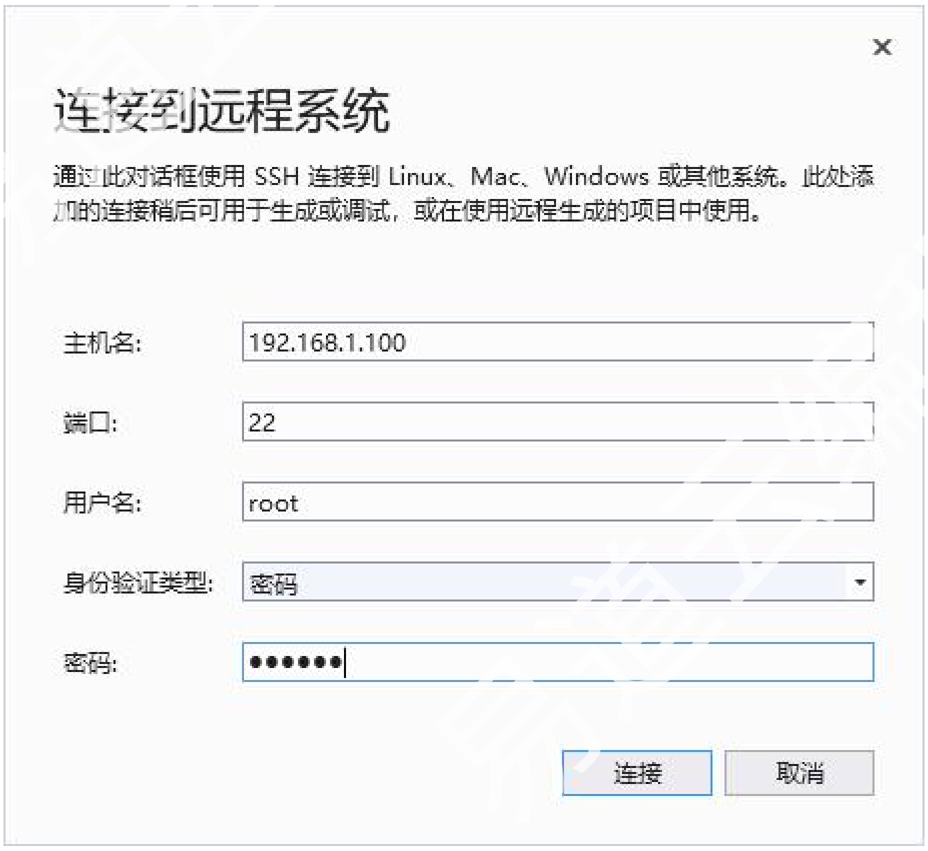


添加或删除到 Linux、Mac 或 Windows 等远程系统的 SSH 连接:

默认	主机名	端口	用户名	操作系统
○	192.168.120.63	22	root	Ubuntu (x6
○	192.168.120.65	22	root	Ubuntu (x6
○	111.231.75.168	22	root	Ubuntu (x6
●	192.168.1.100	22	root	CentOS (x6
○	127.0.0.1	22	root	Ubuntu (x6

此处添加的连接稍后可用于生成或调试，或在使用远程生成的项目中使用。

点击**添加**按钮，进入远程连接界面：



连接到远程系统

通过此对话框使用 SSH 连接到 Linux、Mac、Windows 或其他系统。此处添加的连接稍后可用于生成或调试，或在使用远程生成的项目中使用。

主机名: 192.168.1.100

端口: 22

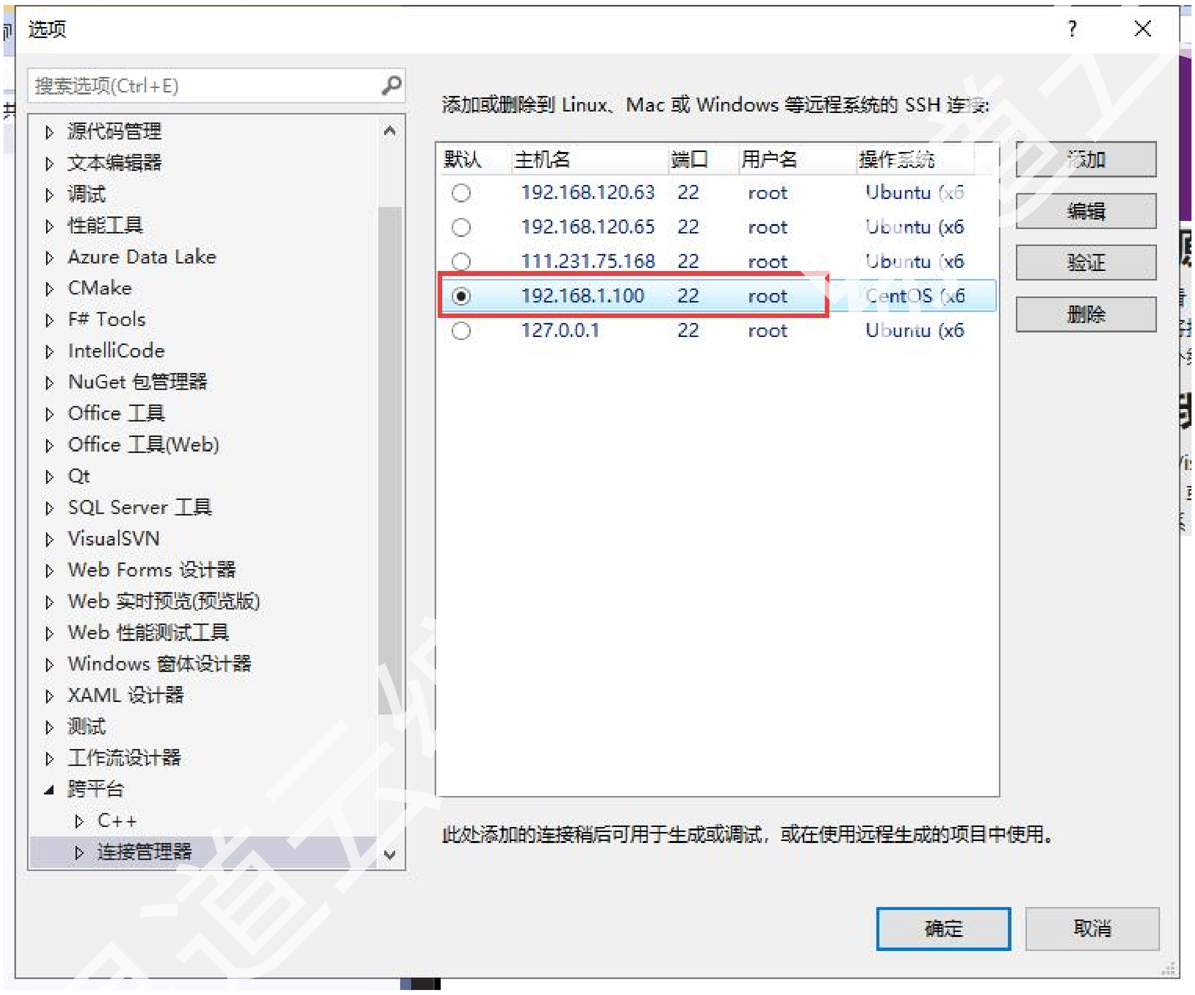
用户名: root

身份验证类型: 密码

密码: ●●●●●●

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：



添加或删除到 Linux、Mac 或 Windows 等远程系统的 SSH 连接:

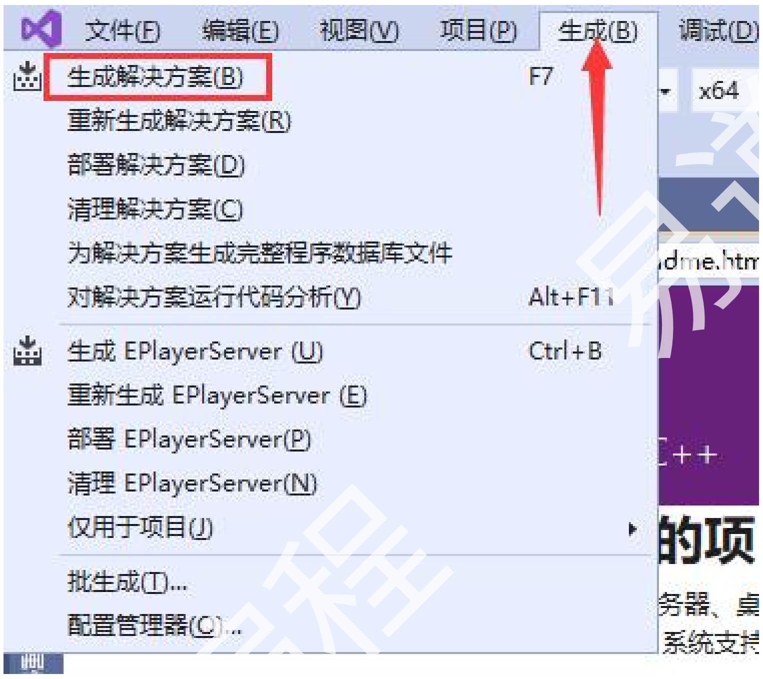
默认	主机名	端口	用户名	操作系统
○	192.168.120.63	22	root	Ubuntu (x6
○	192.168.120.65	22	root	Ubuntu (x6
○	111.231.75.168	22	root	Ubuntu (x6
●	192.168.1.100	22	root	CentOS (x6
○	127.0.0.1	22	root	Ubuntu (x6

此处添加的连接稍后可用于生成或调试，或在使用远程生成的项目中使用。

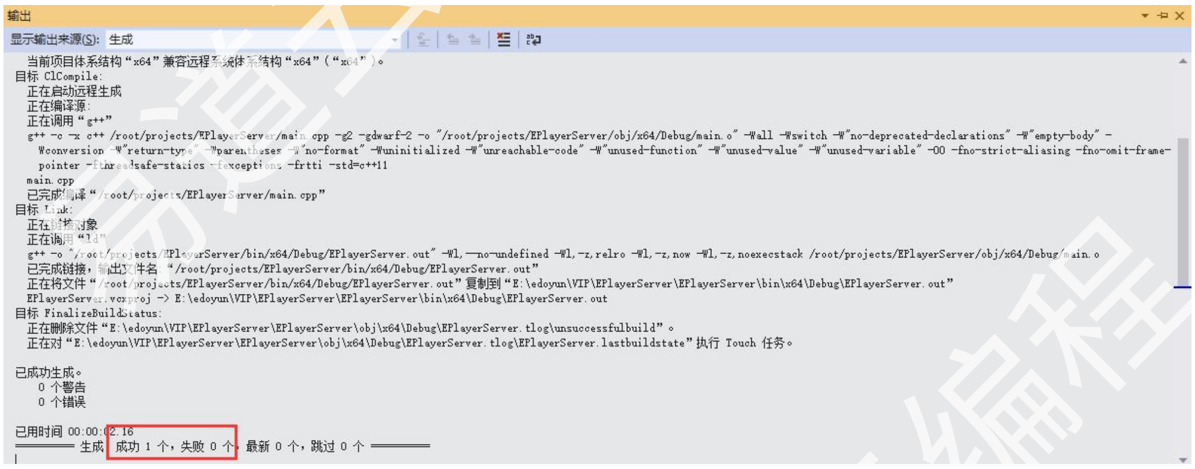
则表示成功。（注意，操作系统有可能识别不正确，但是对开发没有影响）

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**

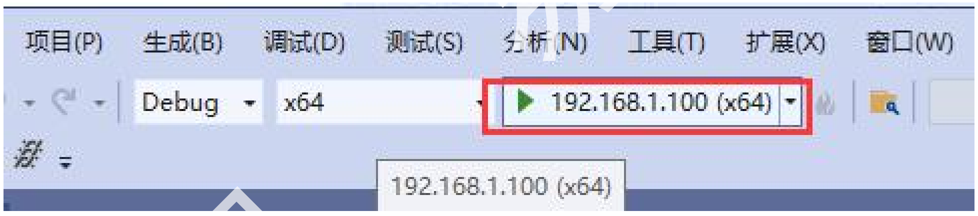


```
  当前项目体系结构“x64”兼容远程系统体系结构“x64”(“x64”)。
目标 ClCompile:
  正在启动远程生成
  正在编译源:
  正在调用“g++”
  g++ -c -x c++ /root/projects/EPlayerServer/main.cpp -g2 -gdwarf-2 -o "/root/projects/EPlayerServer/obj/x64/Debug/main.o" -Wall -Wswitch -W"no-deprecated-declarations" -W"empty-body" -
    Wconversion -W"return-type" -Wparentheses -W"no-format" -Wuninitialized -W"unreachable-code" -W"unused-function" -W"unused-value" -W"unused-variable" -O0 -fno-strict-aliasing -fno-omit-frame-
    pointer -fthreadsafe-statics -fexceptions -frtti -std=c++11
  main.cpp
  已完成编译“/root/projects/EPlayerServer/main.cpp”
目标 Link:
  正在链接对象
  正在调用“ld”
  g++ -o "/root/projects/EPlayerServer/bin/x64/Debug/EPlayerServer.out" -Wl,--no-undefined -Wl,-z,relro -Wl,-z,now -Wl,-z,noexecstack /root/projects/EPlayerServer/obj/x64/Debug/main.o
  已完成链接，输出文件名:“/root/projects/EPlayerServer/bin/x64/Debug/EPlayerServer.out”
  正在将文件“/root/projects/EPlayerServer/bin/x64/Debug/EPlayerServer.out”复制到“E:\edoyun\VIP\EPlayerServer\EPlayerServer\bin\x64\Debug\EPlayerServer.out”
  EPlayerServer.vcxproj -> E:\edoyun\VIP\EPlayerServer\EPlayerServer\bin\x64\Debug\EPlayerServer.out
目标 FinalizeBuildStatus:
  正在删除文件“E:\edoyun\VIP\EPlayerServer\EPlayerServer\obj\x64\Debug\EPlayerServer.tlog\unsuccessfulbuild”。
  正在对“E:\edoyun\VIP\EPlayerServer\EPlayerServer\obj\x64\Debug\EPlayerServer.tlog\EPlayerServer.lastbuildstate”执行 Touch 任务。

已成功生成。
    0 个警告
    0 个错误

已用时间 00:00:02.16
========== 生成: 成功 1 个，失败 0 个，最新 0 个，跳过 0 个 ==========
```

看到生成成功1个，则表示一切ok。

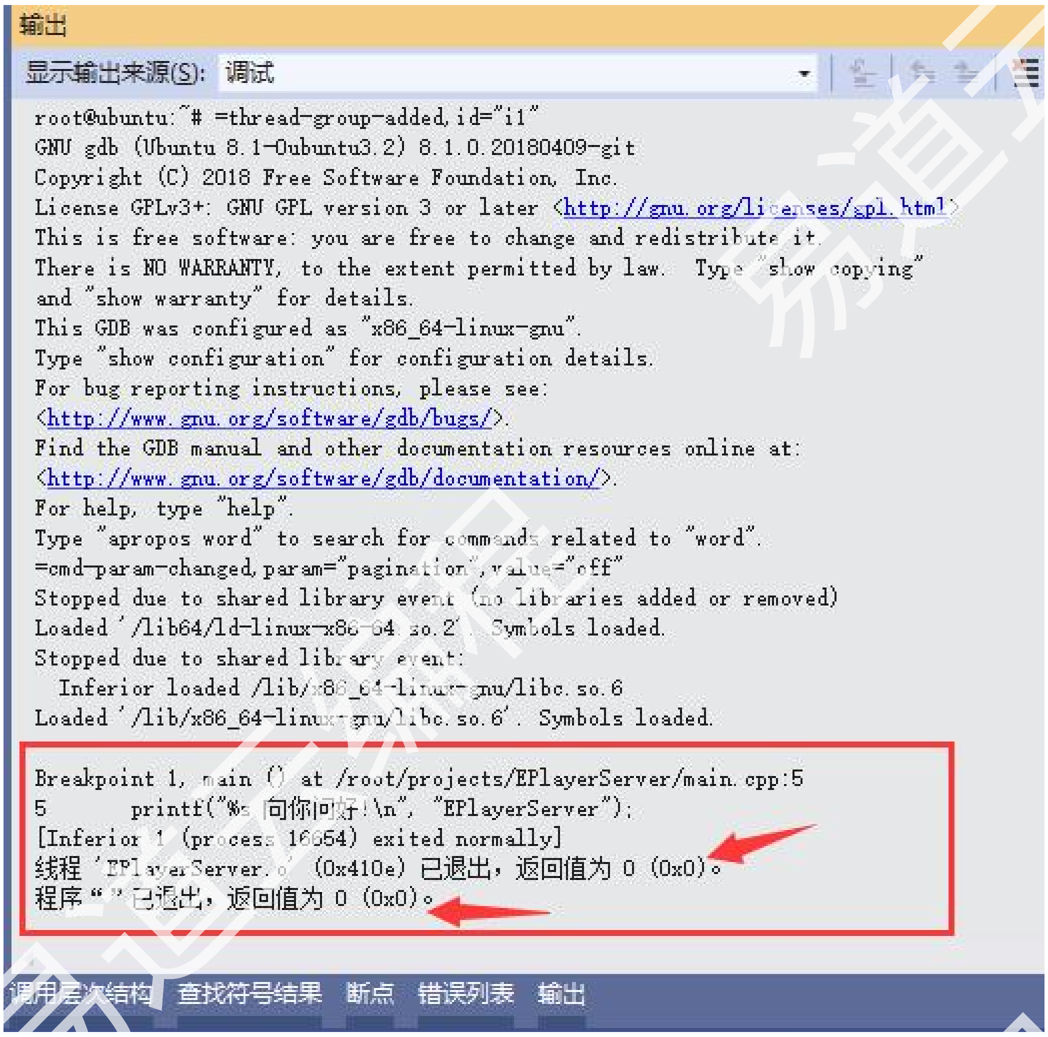
如果有错误，看看前面的操作是否存在问题，再运行一次。

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。



```
root@ubuntu:~# =thread-group-added,id="i1"
GNU gdb (Ubuntu 8.1-0ubuntu3.2) 8.1.0.20180409-git
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word".
=cmd-param-changed,param="pagination",value="off"
Stopped due to shared library event (no libraries added or removed)
Loaded '/lib64/ld-linux-x86-64.so.2'. Symbols loaded.
Stopped due to shared library event:
  Inferior loaded /lib/x86_64-linux-gnu/libc.so.6
Loaded '/lib/x86_64-linux-gnu/libc.so.6'. Symbols loaded.

Breakpoint 1, main () at /root/projects/EPlayerServer/main.cpp:5
5       printf("%s 向你问好!\n", "EPlayerServer");
[Inferior 1 (process 16654) exited normally]
线程 'EPlayerServer.o' (0x410e) 已退出，返回值为 0 (0x0)。
程序“”已退出，返回值为 0 (0x0)。
```

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

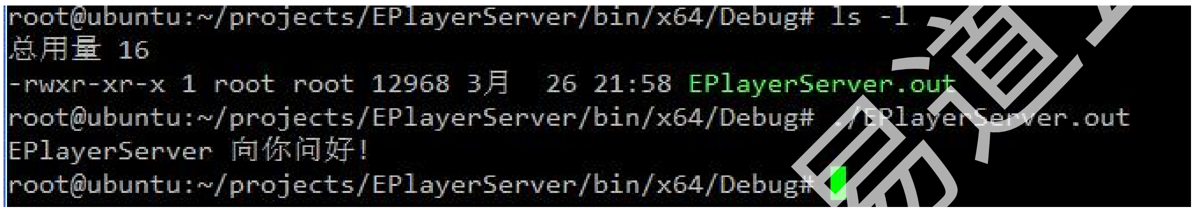
可以通过`cd /root/projects/EPlayerServer/bin/x64/Debug`

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：


```
root@ubuntu:~/projects/EPlayerServer/bin/x64/Debug# ls -l
总用量 16
-rwxr-xr-x 1 root root 12968 3月  26 21:58 EPlayerServer.out
root@ubuntu:~/projects/EPlayerServer/bin/x64/Debug# ./EPlayerServer.out
EPlayerServer 向你问好!
root@ubuntu:~/projects/EPlayerServer/bin/x64/Debug# 
```

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。

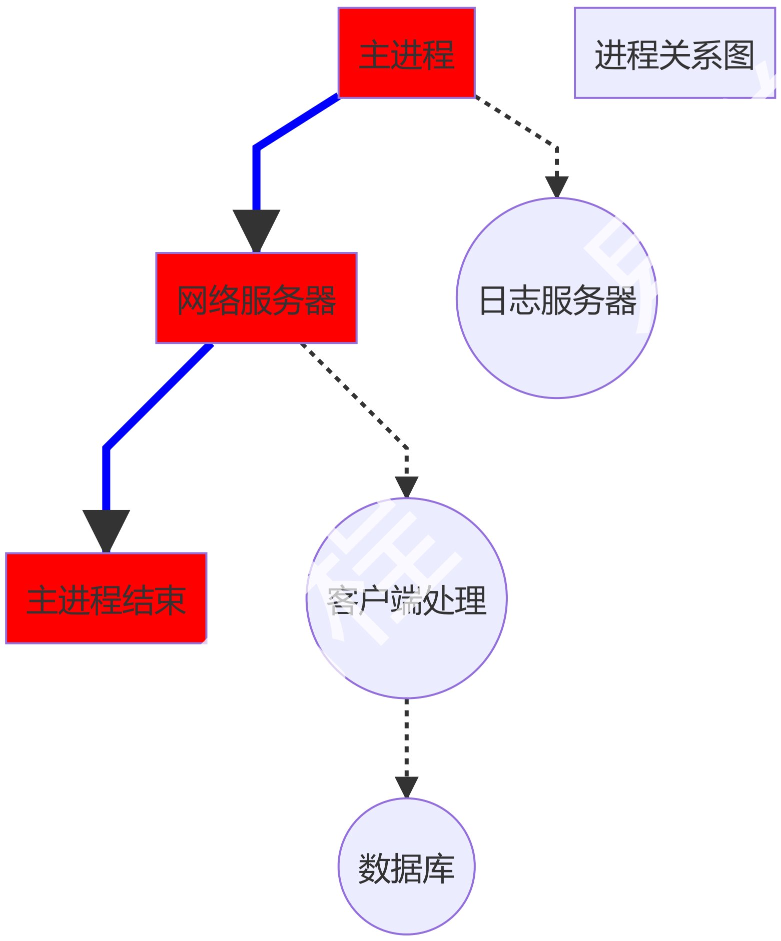
这也就意味着，如果要充分利用系统资源，最好的形式是多线程多进程模式。

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。

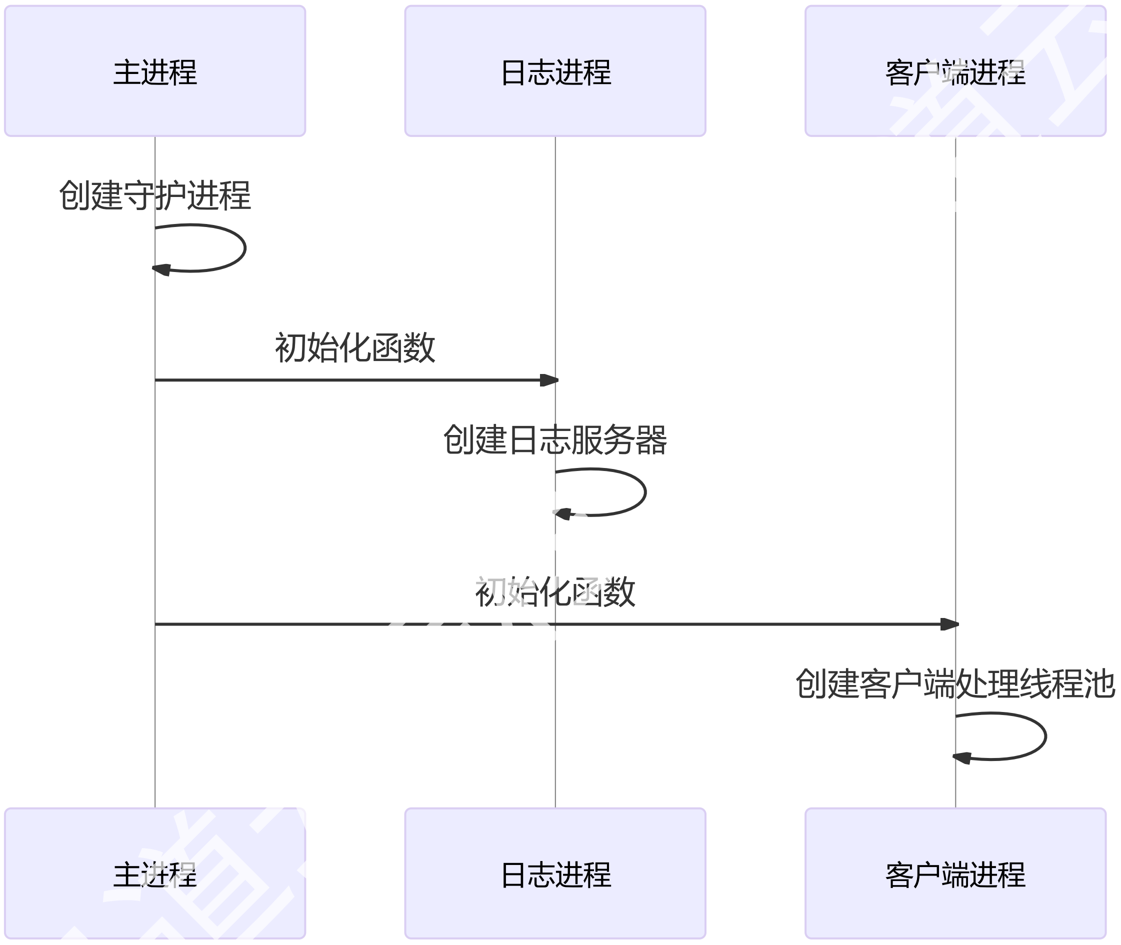
所以我们最终要生成的进程是日志进程和客户端通信处理进程。

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

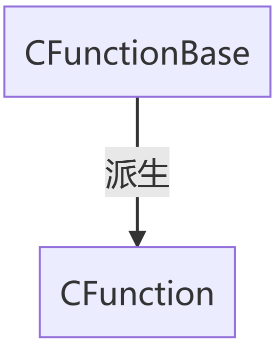
```
#include <unistd.h>
pid_t fork(void);
```

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;
```

```cpp
};

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;
    }

    int CreateSubProcess() {
        if (m_func == NULL)return -1;
        pid_t pid = fork();
        if (pid == -1)return -2;
        if (pid == 0) {
            //子进程
            return (*m_func)();
        }
        //主进程
        m_pid = pid;
        return 0;
    }

private:
    CFunctionBase* m_func;
    pid_t m_pid;
};


int CreateLogServer(CProcess* proc)
{
    return 0;
}

int CreateClientServer(CProcess* proc)
{
    return 0;
}

int main()
{
    CProcess proclog,proccliets;
    proclog.SetEntryFunction(CreateLogServer,
&proclog);
    int ret = proclog.CreateSubProcess();
    proccliets.SetEntryFunction(CreateClientServer, &proccliets);
    ret = proccliets.CreateSubProcess();
    return 0;
}
```

进程间文件描述符的实现

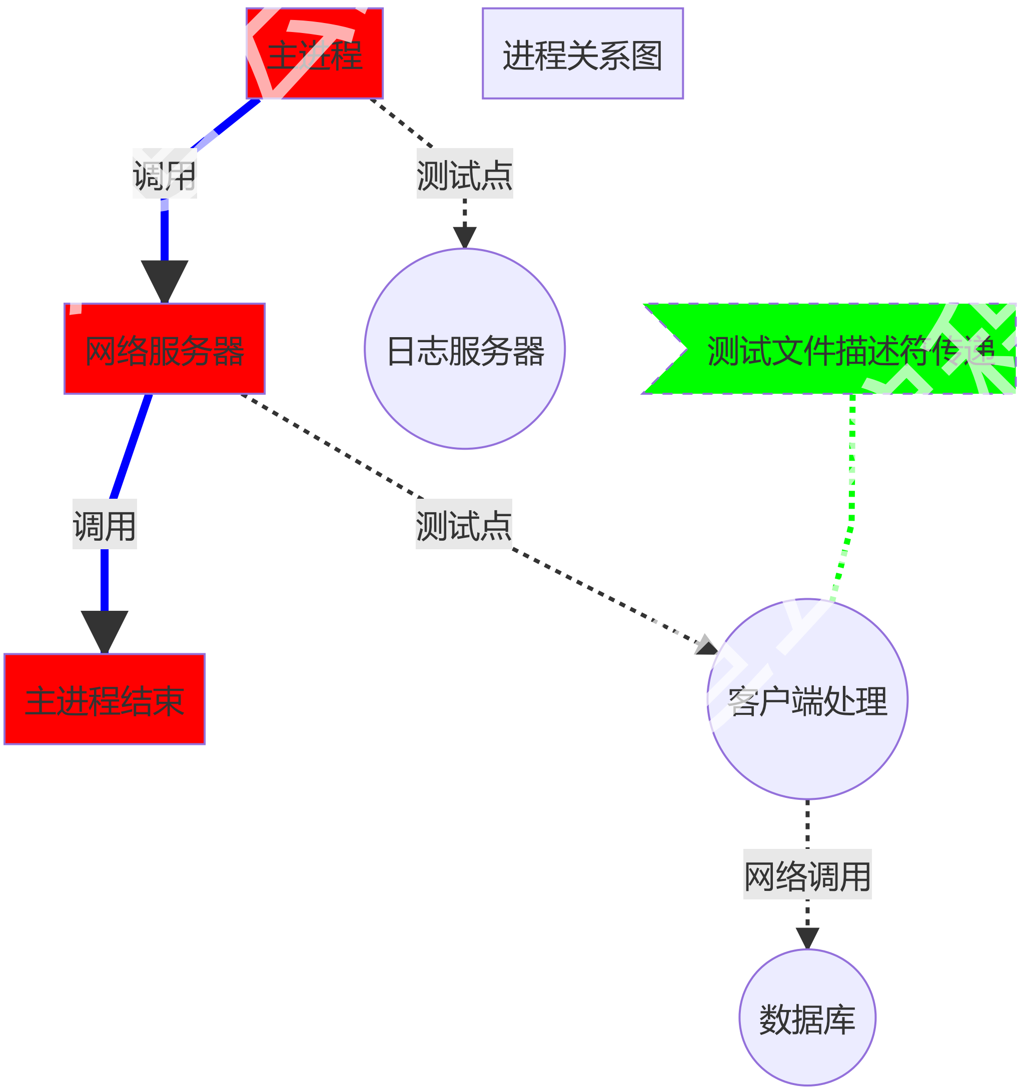
```
#include <unistd.h>
#include <sys/types.h>
#include <functional>
#include <memory.h>
#include <sys/socket.h>


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
{
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
            close(pipes[1]);//关闭掉写
            pipes[1] = 0;
            return (*m_func)();
        }
        //主进程
        close(pipes[0]);
        pipes[0] = 0;
        m_pid = pid;
        return 0;
    }

    int SendFD(int fd) {//主进程完成
        struct msghdr msg;
        iovec iov[2];
        iov[0].iov_base = (char*)"edoyun";
        iov[0].iov_len = 7;
        iov[1].iov_base = (char*)"jueding";
        iov[1].iov_len = 8;
        msg.msg_iov = iov;
        msg.msg_iovlen = 2;

        // 下面的数据，才是我们需要传递的。
        cmsghdr* cmsg = (cmsghdr*)calloc(1, CMSG_LEN(sizeof(int)));
        if (cmsg == NULL)return -1;
        cmsg->cmsg_len = CMSG_LEN(sizeof(int));
        cmsg->cmsg_level = SOL_SOCKET;
        cmsg->cmsg_type = SCM_RIGHTS;
        *(int*)CMSG_DATA(cmsg) = fd;
        msg.msg_control = cmsg;
        msg.msg_controllen = cmsg->cmsg_len;

        ssize_t ret = sendmsg(pipes[1], &msg, 0);
        free(cmsg);
        if (ret == -1) {
            return -2;
        }
        return 0;
    }

    int RecvFD(int& fd)
    {
        msghdr msg;
        iovec iov[2];
        char buf[][10] = { "","" };
        iov[0].iov_base = buf[0];
        iov[0].iov_len = sizeof(buf[0]);
        iov[1].iov_base = buf[1];
        iov[1].iov_len = sizeof(buf[1]);
        msg.msg_iov = iov;
        msg.msg_iovlen = 2;

        cmsghdr* cmsg = (cmsghdr*)calloc(1, 
CMSG_LEN(sizeof(int)));
        if (cmsg == NULL)return -1;
        cmsg->cmsg_len = CMSG_LEN(sizeof(int));
        cmsg->cmsg_level = SOL_SOCKET;
        cmsg->cmsg_type = SCM_RIGHTS;
        msg.msg_control = cmsg;
        msg.msg_controllen = 
CMSG_LEN(sizeof(int));
        ssize_t ret = recvmsg(pipes[0], &msg, 
0);
        if (ret == -1) {
            free(cmsg);
            return -2;
        }
        fd = *(int*)CMSG_DATA(cmsg);
        return 0;
    }


private:
    CFunctionBase* m_func;
    pid_t m_pid;
    int pipes[2];
};
```

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：

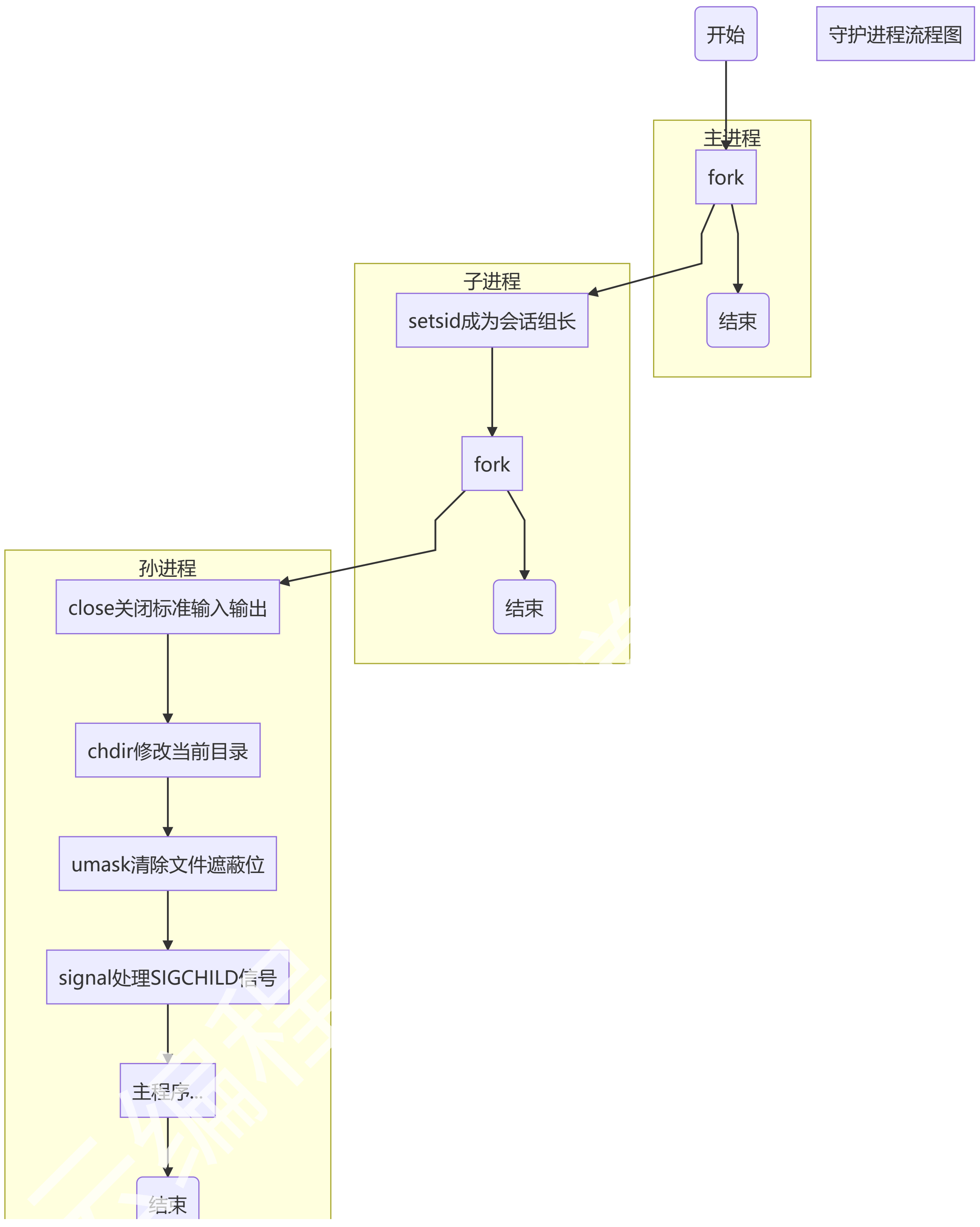
```cpp
#include <cstdio>

#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
{
```

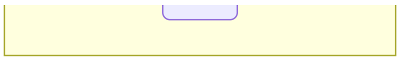
```cpp
public:
    CProcess() {
        m_func = NULL;
        memset(pipes, 0, sizeof(pipes));
    }
    ~CProcess() {
        if (m_func != NULL) {
            delete m_func;
            m_func = NULL;
        }
    }

    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
            close(pipes[1]);//关闭掉写
            pipes[1] = 0;
            ret = (*m_func)();
            exit(0);
        }
        //主进程
        close(pipes[0]);
        pipes[0] = 0;
        m_pid = pid;
        return 0;
    }

    int SendFD(int fd) {//主进程完成
        struct msghdr msg;
        iovec iov[2];
        char buf[2][10] = { "edoyun","jueding" };
        iov[0].iov_base = buf[0];
        iov[0].iov_len = sizeof(buf[0]);
        iov[1].iov_base = buf[1];
        iov[1].iov_len = sizeof(buf[1]);
        msg.msg_iov = iov;
        msg.msg_iovlen = 2;

        // 下面的数据，才是我们需要传递的。
        cmsghdr* cmsg = (cmsghdr*)calloc(1, CMSG_LEN(sizeof(int)));
        if (cmsg == NULL)return -1;
        cmsg->cmsg_len = CMSG_LEN(sizeof(int));
        cmsg->cmsg_level = SOL_SOCKET;
        cmsg->cmsg_type = SCM_RIGHTS;
        *(int*)CMSG_DATA(cmsg) = fd;
        msg.msg_control = cmsg;
        msg.msg_controllen = cmsg->cmsg_len;

        ssize_t ret = sendmsg(pipes[1], &msg, 0);
        free(cmsg);
        if (ret == -1) {
            return -2;
        }
        return 0;
    }

    int RecvFD(int& fd)
    {
        msghdr msg;
        iovec iov[2];
        char buf[][10] = { "","" };
        iov[0].iov_base = buf[0];
        iov[0].iov_len = sizeof(buf[0]);
        iov[1].iov_base = buf[1];
        iov[1].iov_len = sizeof(buf[1]);
        msg.msg_iov = iov;
        msg.msg_iovlen = 2;

        cmsghdr* cmsg = (cmsghdr*)calloc(1, CMSG_LEN(sizeof(int)));
        if (cmsg == NULL)return -1;
        cmsg->cmsg_len = CMSG_LEN(sizeof(int));
        cmsg->cmsg_level = SOL_SOCKET;
        cmsg->cmsg_type = SCM_RIGHTS;
        msg.msg_control = cmsg;
        msg.msg_controllen = CMSG_LEN(sizeof(int));
        ssize_t ret = recvmsg(pipes[0], &msg, 0);
        if (ret == -1) {
            free(cmsg);
            return -2;
        }
        fd = *(int*)CMSG_DATA(cmsg);
        free(cmsg);
        return 0;
    }


private:
    CFunctionBase* m_func;
    pid_t m_pid;
    int pipes[2];
};


int CreateLogServer(CProcess* proc)
{
    printf("%s(%d):<%s> pid=%d\n", __FILE__,
__LINE__, __FUNCTION__, getpid());
    return 0;
}

int CreateClientServer(CProcess* proc)
{
    printf("%s(%d):<%s> pid=%d\n", __FILE__,
__LINE__, __FUNCTION__, getpid());
    int fd = -1;
    int ret = proc->RecvFD(fd);
    printf("%s(%d):<%s> ret=%d\n", __FILE__,
__LINE__, __FUNCTION__, ret);
    printf("%s(%d):<%s> fd=%d\n", __FILE__,
__LINE__, __FUNCTION__, fd);
    sleep(1);
    char buf[10] = "";
    lseek(fd, 0, SEEK_SET);
    read(fd, buf, sizeof(buf));
    printf("%s(%d):<%s> buf=%s\n", __FILE__,
__LINE__, __FUNCTION__, buf);
    close(fd);
    return 0;
}

int main()
{
    CProcess proclog, procclients;
    printf("%s(%d):<%s> pid=%d\n", __FILE__,
__LINE__, __FUNCTION__, getpid());
```

```
    proclog.SetEntryFunction(CreateLogServer, &proclog);
    int ret = proclog.CreateSubProcess();
    if (ret != 0) {
        printf("%s(%d):<%s> pid=%d\n", __FILE__, __LINE__, __FUNCTION__, getpid());
        return -1;
    }
    printf("%s(%d):<%s> pid=%d\n", __FILE__, __LINE__, __FUNCTION__, getpid());
    procclients.SetEntryFunction(CreateClientServer, &procclients);
    ret = procclients.CreateSubProcess();
    if (ret != 0) {
        printf("%s(%d):<%s> pid=%d\n", __FILE__, __LINE__, __FUNCTION__, getpid());
        return -2;
    }
    printf("%s(%d):<%s> pid=%d\n", __FILE__, __LINE__, __FUNCTION__, getpid());
    usleep(100 * 000);
    int fd = open("./test.txt", O_RDWR | O_CREAT | O_APPEND);
    printf("%s(%d):<%s> fd=%d\n", __FILE__, __LINE__, __FUNCTION__, fd);
    if (fd == -1)return -3;
    ret = procclients.SendFD(fd);
    printf("%s(%d):<%s> ret=%d\n", __FILE__, __LINE__, __FUNCTION__, ret);
    if (ret != 0)printf("errno:%d msg:%s\n", errno, strerror(errno));
    write(fd, "edoyun", 6);
    close(fd);
    return 0;
}
```

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。

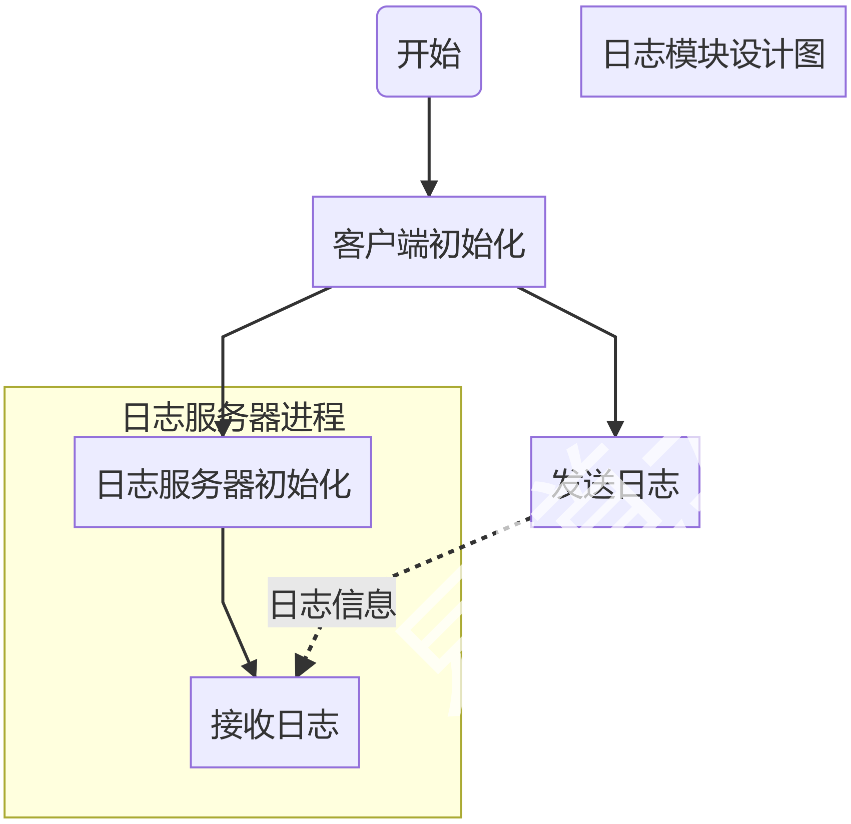
进程间通信，最方便最快速的就是**本地套接字**通信了。

- 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

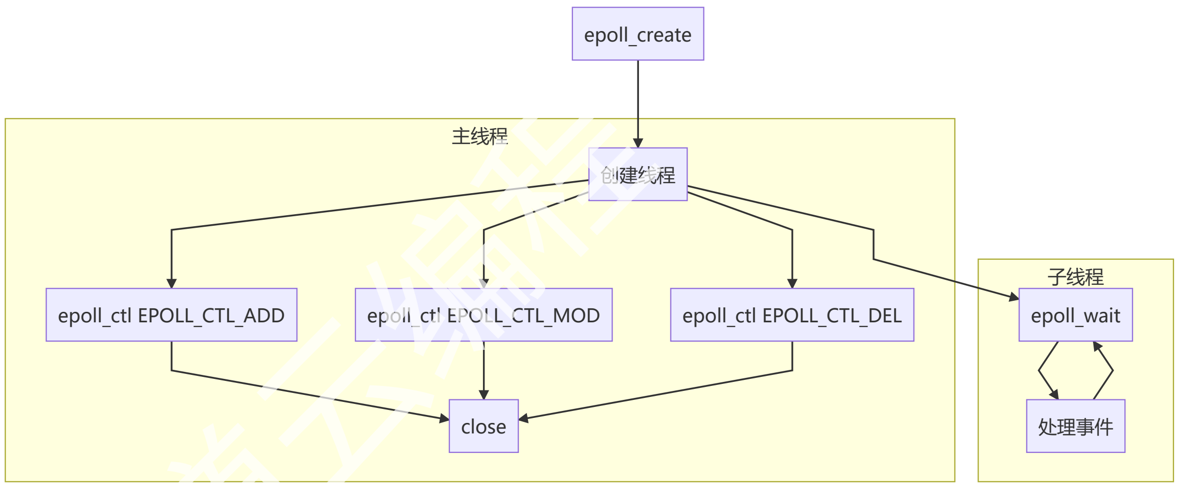
- 无需IP和端口，不影响服务器对外的资源
- 信息无需加锁，可以多线程并发写
- 数据传输量巨大，传输速率高（纯内存读写）

日志模块的设计图



Epoll的封装

epoll简单模型



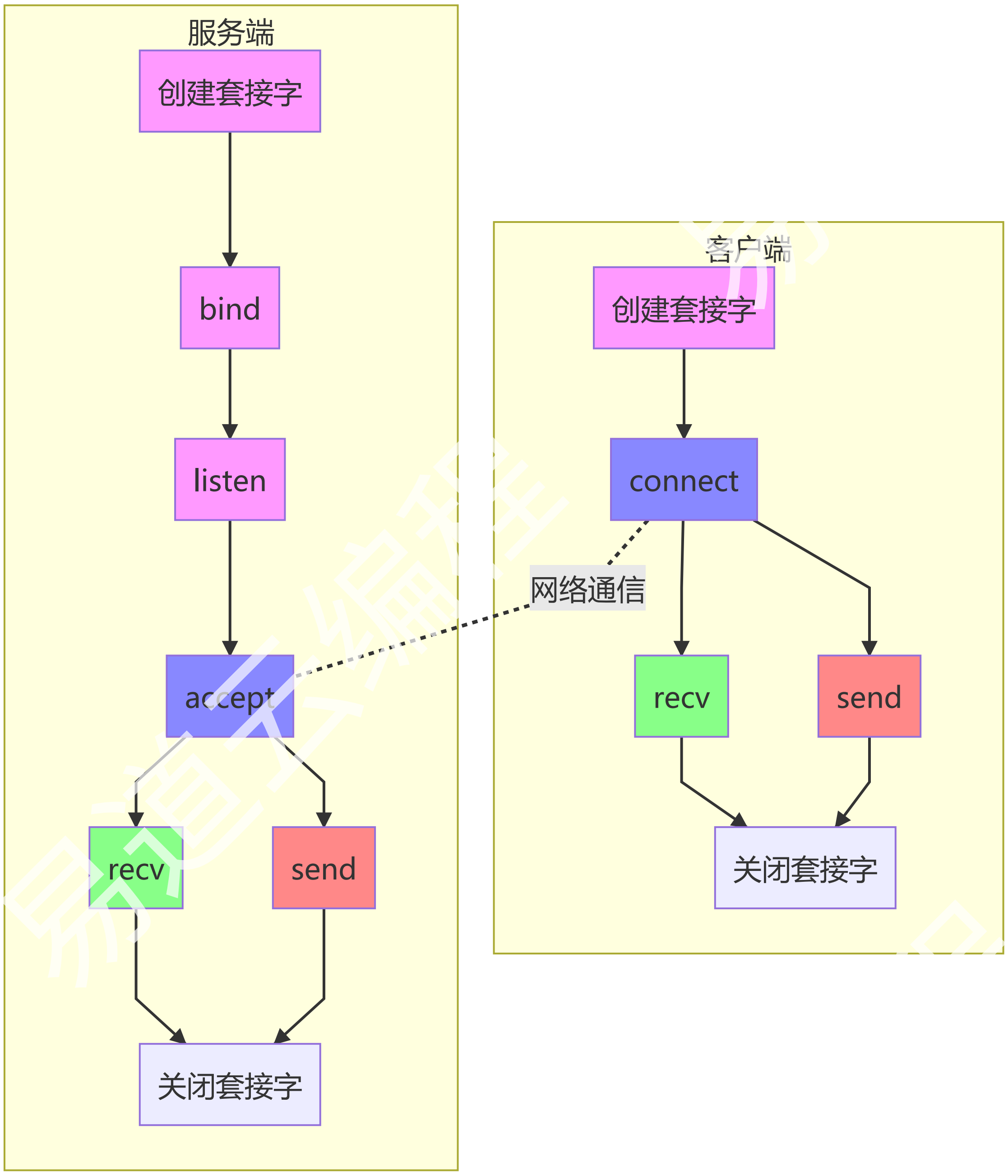
接口：

```
#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; }
    operator const epoll_data_t* ()const { return &m_data; }
private:
    epoll_data_t m_data;
};

using EPEvents = std::vector<epoll_event>;

class CEpoll
{
public:
    CEpoll() {
        m_epoll = -1;
    }
    ~CEpoll() {
        Close();
    }
public:
    CEpoll(const CEpoll&) = delete;
    CEpoll& operator=(const CEpoll&) = delete;
public:
    operator int()const { return m_epoll; }
public:
    int Create(unsigned count) {
        if (m_epoll != -1)return -1;
        m_epoll = epoll_create(count);
        if (m_epoll == -1)return -2;
        return 0;
    }
    //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
    ssize_t WaitEvents(EPEvents& events, int timeout = 10) {
        if (m_epoll == -1)return -1;
        EPEvents evs(EVENT_SIZE);
        int ret = epoll_wait(m_epoll, evs.data(), (int)evs.size(), timeout);
        if (ret == -1) {
            if ((errno == EINTR) || (errno == EAGAIN)) {
                return 0;
            }
            return -2;
        }
        if (ret > (int)events.size()) {
            events.resize(ret);
        }
        memcpy(events.data(), evs.data(), sizeof(epoll_event) * ret);
        return ret;
    }
    int Add(int fd, const EpollData& data = EpollData((void*)0), uint32_t events = EPOLLIN)
    {
        if (m_epoll == -1)return -1;
        epoll_event ev = { events,data };
        int ret = epoll_ctl(m_epoll, EPOLL_CTL_ADD, fd, &ev);
        if (ret == -1)return -2;
        return 0;
    }
    int Modify(int fd, uint32_t events, const EpollData& data = EpollData((void*)0))
    {
        if (m_epoll == -1)return -1;
        epoll_event ev = { events,data };
        int ret = epoll_ctl(m_epoll, EPOLL_CTL_MOD, fd, &ev);
        if (ret == -1)return -2;
        return 0;
    }
    int Del(int fd)
    {
        if (m_epoll == -1)return -1;
        int ret = epoll_ctl(m_epoll,
EPOLL_CTL_DEL, fd, NULL);
        if (ret == -1)return -2;
        return 0;
    }
    void Close() {
        if (m_epoll != -1) {
            int fd = m_epoll;
            m_epoll = -1;
            close(fd);
        }
    }

private:
    int m_epoll;
};
```

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);
    }
    CSockParam(const Buffer& path, int attr) {
        ip = path;
        addr_un.sun_family = AF_UNIX;
        strcpy(addr_un.sun_path, path);
        this->attr = attr;
    }
    ~CSockParam() {}
    CSockParam(const CSockParam& param) {
        ip = param.ip;
        port = param.port;
        attr = param.attr;
        memcpy(&addr_in, &param.addr_in, sizeof(addr_in));
        memcpy(&addr_un, &param.addr_un, sizeof(addr_un));
    }
public:
    CSockParam& operator=(const CSockParam& param) {
        if (this != &param) {
            ip = param.ip;
            port = param.port;
            attr = param.attr;
            memcpy(&addr_in, &param.addr_in, sizeof(addr_in));
            memcpy(&addr_un, &param.addr_un, sizeof(addr_un));
        }
        return *this;
    }
    sockaddr* addrin() { return (sockaddr*)&addr_in; }
    sockaddr* addrun() { return (sockaddr*)&addr_un; }
public:
    //地址
    sockaddr_in addr_in;
    sockaddr_un addr_un;
    //ip
    Buffer ip;
    //端口
    short port;
    //参考SockAttr
    int attr;
};

class CSocketBase
{
public:
    CSocketBase() {
        m_socket = -1;
        m_status = 0;//初始化未完成
    }
    //传递析构操作
    virtual ~CSocketBase() {
        Close();
    }
public:
    //初始化 服务器 套接字创建、bind、listen  客户端套接字创建
    virtual int Init(const CSockParam& param) = 0;
    //连接 服务器 accept 客户端 connect  对于udp这里可以忽略
    virtual int Link(CSocketBase** pClient = NULL) = 0;
    //发送数据
    virtual int Send(const Buffer& data) = 0;
    //接收数据
    virtual int Recv(Buffer& data) = 0;
    //关闭连接
    virtual int Close() {
        m_status = 3;
        if (m_socket != -1) {
            int fd = m_socket;
            m_socket = -1;
            close(fd);
        }
    };
protected:
    //套接字描述符，默认是-1
    int m_socket;
    //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
    int m_status;
    //初始化参数
    CSockParam m_param;
};

class CLocalSocket
    :public CSocketBase
{
public:
    CLocalSocket() :CSocketBase() {}
    CLocalSocket(int sock) :CSocketBase() {
        m_socket = sock;
    }
    //传递析构操作
    virtual ~CLocalSocket() {
        Close();
    }
public:
    //初始化 服务器 套接字创建、bind、listen  客户端 套接字创建
    virtual int Init(const CSockParam& param) {
        if (m_status != 0)return -1;
        m_param = param;
        int type = (m_param.attr & SOCK_ISUDP) ? SOCK_DGRAM : SOCK_STREAM;
        if (m_socket == -1)
```

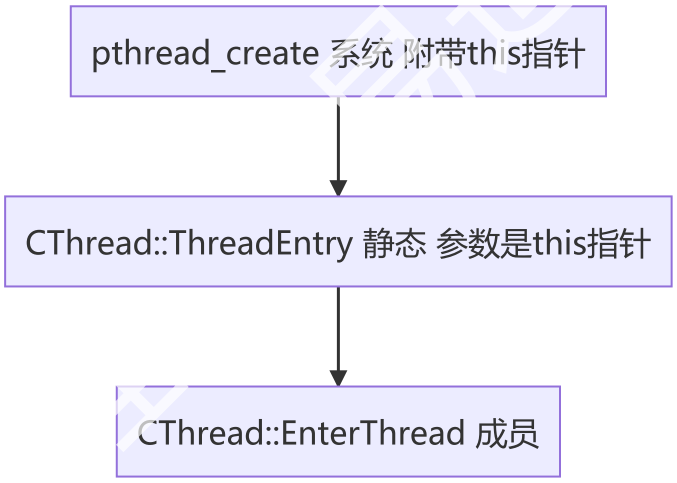
```
            m_socket = socket(PF_LOCAL, type, 0);
        if (m_socket == -1)return -2;
        int ret = 0;
        if (m_param.attr & SOCK_ISSERVER) {
            ret = bind(m_socket, m_param.addrun(), sizeof(sockaddr_un));
            if (ret == -1) return -3;
            ret = listen(m_socket, 32);
            if (ret == -1)return -4;
        }
        if (m_param.attr & SOCK_ISNONBLOCK) {
            int option = fcntl(m_socket, F_GETFL);
            if (option == -1)return -5;
            option |= O_NONBLOCK;
            ret = fcntl(m_socket, F_SETFL, option);
            if (ret == -1)return -6;
        }
        m_status = 1;
        return 0;
    }
    //连接 服务器 accept 客户端 connect  对于udp这里可以忽略
    virtual int Link(CSocketBase** pClient = NULL) {
        if (m_status <= 0 || (m_socket == -1))return -1;
        int ret = 0;
        if (m_param.attr & SOCK_ISSERVER) {
            if (pClient == NULL)return -2;
            CSockParam param;
            socklen_t len = sizeof(sockaddr_un);
            int fd = accept(m_socket, param.addrun(), &len);
            if (fd == -1)return -3;
```

```
                *pClient = new CLocalSocket(fd);
                if (*pClient == NULL)return -4;
                ret = (*pClient)->Init(param);
                if (ret != 0) {
                    delete (*pClient);
                    *pClient = NULL;
                    return -5;
                }
            }
            else {
                ret = connect(m_socket,
m_param.addrun(), sizeof(sockaddr_un));
                if (ret != 0)return -6;
            }
            m_status = 2;
            return 0;
        }
        //发送数据
        virtual int Send(const Buffer& data) {
            if (m_status < 2 || (m_socket ==
-1))return -1;
            ssize_t index = 0;
            while (index < (ssize_t)data.size()) {
                ssize_t len = write(m_socket,
(char*)data + index, data.size() - index);
                if (len == 0)return -2;
                if (len < 0)return -3;
                index += len;
            }
            return 0;
        }
        //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
        virtual int Recv(Buffer& data) {
            if (m_status < 2 || (m_socket ==
-1))return -1;
```

```
        ssize_t len = read(m_socket, data, data.size());
        if (len > 0) {
            data.resize(len);
            return (int)len;//收到数据
        }
        if (len < 0) {
            if (errno == EINTR || (errno == EAGAIN)) {
                data.clear();
                return 0;//没有数据收到
            }
            return -2;//发送错误
        }
        return -3;//套接字被关闭
    }
    //关闭连接
    virtual int Close() {
        return CSocketBase::Close();
    }
};
```

线程的封装

静态函数到非静态函数的转换


pthread_create 系统 附带this指针
CThread::ThreadEntry 静态 参数是this指针
CThread::EnterThread 成员

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_>
```

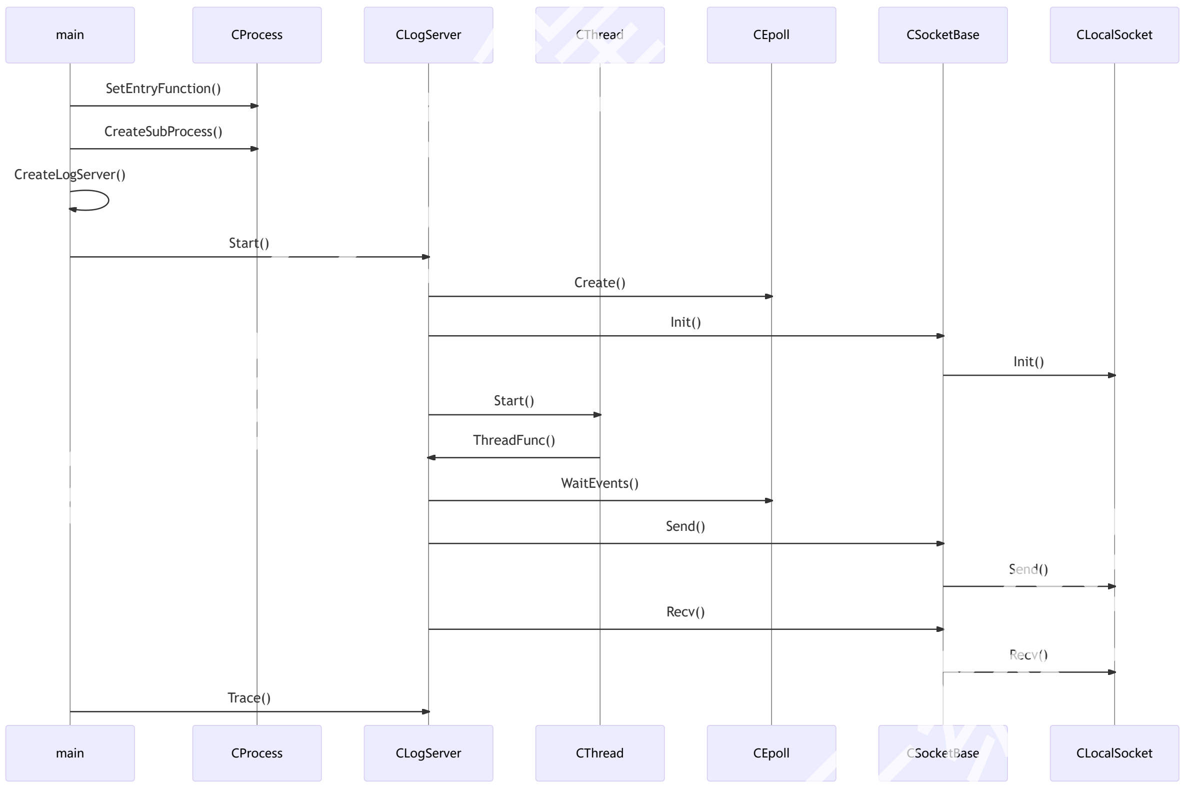
```cpp
    int SetThreadFunc(_FUNCTION_ func, _ARGS_... args)
    {
        m_function = new CFunction<_FUNCTION_, _ARGS_...>(func, args...);
        if (m_function == NULL)return -1;
        return 0;
    }
    int Start() {
        pthread_attr_t attr;
        int ret = 0;
        ret = pthread_attr_init(&attr);
        if (ret != 0)return -1;
        ret = pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);
        if (ret != 0)return -2;
        ret = pthread_attr_setscope(&attr, PTHREAD_SCOPE_PROCESS);
        if (ret != 0)return -3;
        ret = pthread_create(&m_thread, &attr, &CThread::ThreadEntry, this);
        if (ret != 0)return -4;
        m_mapThread[m_thread] = this;
        ret = pthread_attr_destroy(&attr);
        if (ret != 0)return -5;
        return 0;
    }
    int Pause() {
        if (m_thread != 0)return -1;
        if (m_bpaused) {
            m_bpaused = false;
            return 0;
        }
        m_bpaused = true;
        int ret = pthread_kill(m_thread, SIGUSR1);
        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;
        pthread_t thread = pthread_self();//不是冗余，有可能被stop函数把m_thread给清零了
        auto it = m_mapThread.find(thread);
        if (it != m_mapThread.end())
            m_mapThread[thread] = NULL;
        pthread_detach(thread);
        pthread_exit(NULL);
    }

    static void Sigaction(int signo, siginfo_t* info, void* context)
    {
        if (signo == SIGUSR1) {
            pthread_t thread = pthread_self();
            auto it = m_mapThread.find(thread);
            if (it != m_mapThread.end()) {
                if (it->second) {
                    while (it->second->m_bpaused) {
                        if (it->second->m_thread == 0) {
                            pthread_exit(NULL);
                        }
                        usleep(1000);//1ms
                    }
                }
            }
        }
        else if (signo == SIGUSR2) {//线程退出
            pthread_exit(NULL);
        }
    }

    void EnterThread() {//__thiscall
        if (m_function != NULL) {
            int ret = (*m_function)();
            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>
    LogInfo& operator<<(const T& data) {
        std::stringstream stream;
        stream << data;
        m_buf += stream.str();
        return *this;
    }
private:
    bool bAuto;//默认是false 流式日志，则为true
    Buffer m_buf;
};

class CLoggerServer
{
public:
    CLoggerServer() :
        m_thread(&CLoggerServer::ThreadFunc,
this)
    {
        m_server = NULL;
        m_path = "./log/" + GetTimeStr() +
".log";
        printf("%s(%d):[%s]path=%s\n", __FILE__,
__LINE__, __FUNCTION__, (char*)m_path);
    }
    ~CLoggerServer() {
        Close();
    }
public:
    CLoggerServer(const CLoggerServer&) =
delete;
    CLoggerServer& operator=(const
CLoggerServer&) = delete;
public:
    //日志服务器的启动
    int Start() {
        if (m_server != NULL)return -1;
        if (access("log", W_OK | R_OK) != 0) {
            mkdir("log", S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH);
        }
        m_file = fopen(m_path, "w+");
        if (m_file == NULL)return -2;
        int ret = m_epoll.Create(1);
        if (ret != 0)return -3;
        m_server = new CLocalSocket();
        if (m_server == NULL) {
            Close();
            return -4;
        }
        ret = m_server->Init(CSockParam("./log/server.sock", (int)SOCK_ISSERVER));
        if (ret != 0) {
            Close();
            return -5;
        }
        ret = m_thread.Start();
        if (ret != 0) {
            Close();
            return -6;
        }
        return 0;
    }
    int ThreadFunc() {
        EPEvents events;
        std::map<int, CSocketBase*> mapClients;
        while (m_thread.isValid() && (m_epoll != -1) && (m_server != NULL)) {
            ssize_t ret = m_epoll.WaitEvents(events, 1);
            if (ret < 0)break;
            if (ret > 0) {
                ssize_t i = 0;
                for (; i < ret; i++) {
                    if (events[i].events & EPOLLERR) {
                        break;
                    }
                    else if (events[i].events & EPOLLIN) {
                        if (events[i].data.ptr == m_server) {
                            CSocketBase* pClient = NULL;
                            int r = m_server->Link(&pClient);
                            if (r < 0) continue;
                            r = m_epoll.Add(*pClient, EpollData((void*)pClient), EPOLLIN | EPOLLERR);
                            if (r < 0) {
                                delete pClient;
                                continue;
                            }
                            auto it = mapClients.find(*pClient);
                            if (it->second != NULL) {
                                delete it->second;
                            }
                            mapClients[*pClient] = pClient;
                        }
                        else {
                            CSocketBase* pClient = (CSocketBase*)events[i].data.ptr;
                            if (pClient != NULL) {
                                Buffer data(1024 * 1024);
                                int r = pClient->Recv(data);
                                if (r <= 0) {
                                    delete pClient;
                                    mapClients[*pClient] = NULL;
                                }
                                else {
                                    WriteLog(data);
                                }
                            }
                        }
                    }
                }
                if (i != ret) {
                    break;
                }
            }
        }
        for (auto it = mapClients.begin(); it != mapClients.end(); it++) {
            if (it->second) {
                delete it->second;
            }
        }
        mapClients.clear();
        return 0;
    }
    int Close() {
        if (m_server != NULL) {
            CSocketBase* p = m_server;
            m_server = NULL;
            delete p;
        }
        m_epoll.Close();
```

```cpp
        m_thread.Stop();
        return 0;
    }
    //给其他非日志进程的进程和线程使用的
    static void Trace(const LogInfo& info) {
        static thread_local CLocalSocket client;
        if (client == -1) {
            int ret = 0;
            ret = client.Init(CSockParam("./log/server.sock", 0));
            if (ret != 0) {
#ifdef _DEBUG
                printf("%s(%d):[%s]ret=%d\n", __FILE__, __LINE__, __FUNCTION__, ret);
#endif
                return;
            }
        }
        client.Send(info);
    }
    static Buffer GetTimeStr() {
        Buffer result(128);
        timeb tmb;
        ftime(&tmb);
        tm* pTm = localtime(&tmb.time);
        int nSize = snprintf(result, result.size(),
            "%04d-%02d-%02d %02d-%02d-%02d %03d",
            pTm->tm_year + 1900, pTm->tm_mon + 1, pTm->tm_mday,
            pTm->tm_hour, pTm->tm_min, pTm->tm_sec,
            tmb.millitm
        );
        result.resize(nSize);
        return result;
    }
private:
    void WriteLog(const Buffer& data) {
        if (m_file != NULL) {
            FILE* pFile = m_file;
            fwrite((char*)data, 1, data.size(), 
pFile);
            fflush(pFile);
#ifdef _DEBUG
            printf("%s", (char*)data);
#endif
        }
    }
private:
    CThread m_thread;
    CEpoll m_epoll;
    CSocketBase* m_server;
    Buffer m_path;
    FILE* m_file;
};

#ifndef TRACE
#define TRACEI(...) 
CLoggerServer::Trace(LogInfo(__FILE__, __LINE__, 
__FUNCTION__, getpid(), pthread_self(), 
LOG_INFO, __VA_AGRS__))
#define TRACED(...) 
CLoggerServer::Trace(LogInfo(__FILE__, __LINE__, 
__FUNCTION__, getpid(), pthread_self(), 
LOG_DEBUG, __VA_AGRS__))
#define TRACEW(...) 
CLoggerServer::Trace(LogInfo(__FILE__, __LINE__, 
__FUNCTION__, getpid(), pthread_self(), 
LOG_WARNING, __VA_AGRS__))
```

```
#define TRACEE(...) CLoggerServer::Trace(LogInfo(__FILE__, __LINE__, __FUNCTION__, getpid(), pthread_self(), LOG_ERROR, __VA_AGRS__))
#define TRACEF(...) CLoggerServer::Trace(LogInfo(__FILE__, __LINE__, __FUNCTION__, getpid(), pthread_self(), LOG_FATAL, __VA_AGRS__))

//LOGI<<"hello"<<"how are you";
#define LOGI LogInfo(__FILE__, __LINE__, __FUNCTION__, getpid(), pthread_self(), LOG_INFO)
#define LOGD LogInfo(__FILE__, __LINE__, __FUNCTION__, getpid(), pthread_self(), LOG_DEBUG)
#define LOGW LogInfo(__FILE__, __LINE__, __FUNCTION__, getpid(), pthread_self(), LOG_WARNING)
#define LOGE LogInfo(__FILE__, __LINE__, __FUNCTION__, getpid(), pthread_self(), LOG_ERROR)
#define LOGF LogInfo(__FILE__, __LINE__, __FUNCTION__, getpid(), pthread_self(), LOG_FATAL)

//内存导出
//00 01 02 65……  ; ...a……
//
#define DUMPI(data, size) LogInfo(__FILE__, __LINE__, __FUNCTION__, getpid(), pthread_self(), LOG_INFO, data, size)
#define DUMPD(data, size) LogInfo(__FILE__, __LINE__, __FUNCTION__, getpid(), pthread_self(), LOG_DEBUG, data, size)
```

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

```

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

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

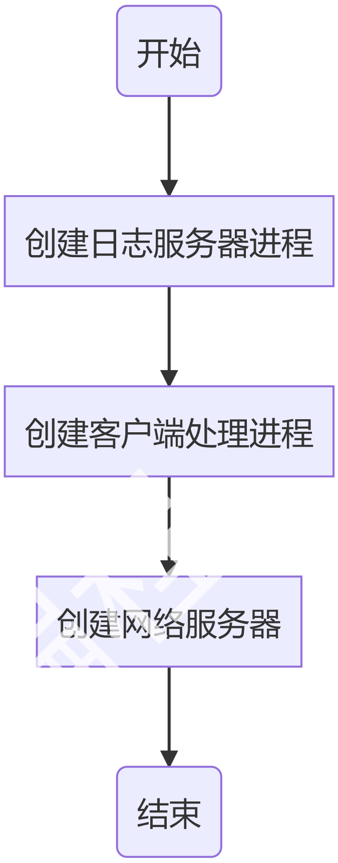
```cpp

int main()
{
    //CProcess::SwitchDeamon();
    CProcess proclog, procclients;
    printf("%s(%d):<%s> pid=%d\n", __FILE__, __LINE__, __FUNCTION__, getpid());
    proclog.SetEntryFunction(CreateLogServer, &proclog);
    int ret = proclog.CreateSubProcess();
    if (ret != 0) {
        printf("%s(%d):<%s> pid=%d\n", __FILE__, __LINE__, __FUNCTION__, getpid());
        return -1;
    }
    LogTest();
    printf("%s(%d):<%s> pid=%d\n", __FILE__, __LINE__, __FUNCTION__, getpid());
    CThread thread(LogTest);
    thread.Start();
    procclients.SetEntryFunction(CreateClientServer, &procclients);
    ret = procclients.CreateSubProcess();
    if (ret != 0) {
        printf("%s(%d):<%s> pid=%d\n", __FILE__, __LINE__, __FUNCTION__, getpid());
        return -2;
    }
    printf("%s(%d):<%s> pid=%d\n", __FILE__, __LINE__, __FUNCTION__, getpid());
    usleep(100 * 000);
    int fd = open("./test.txt", O_RDWR | O_CREAT | O_APPEND);
    printf("%s(%d):<%s> fd=%d\n", __FILE__, __LINE__, __FUNCTION__, fd);
    if (fd == -1)return -3;
    ret = procclients.SendFD(fd);
    printf("%s(%d):<%s> ret=%d\n", __FILE__,
__LINE__, __FUNCTION__, ret);
    if (ret != 0)printf("errno:%d msg:%s\n",
errno, strerror(errno));
    write(fd, "edoyun", 6);
    close(fd);
    proclog.SendFD(-1);
    (void)getchar();
    return 0;
}
```

主模块的设计

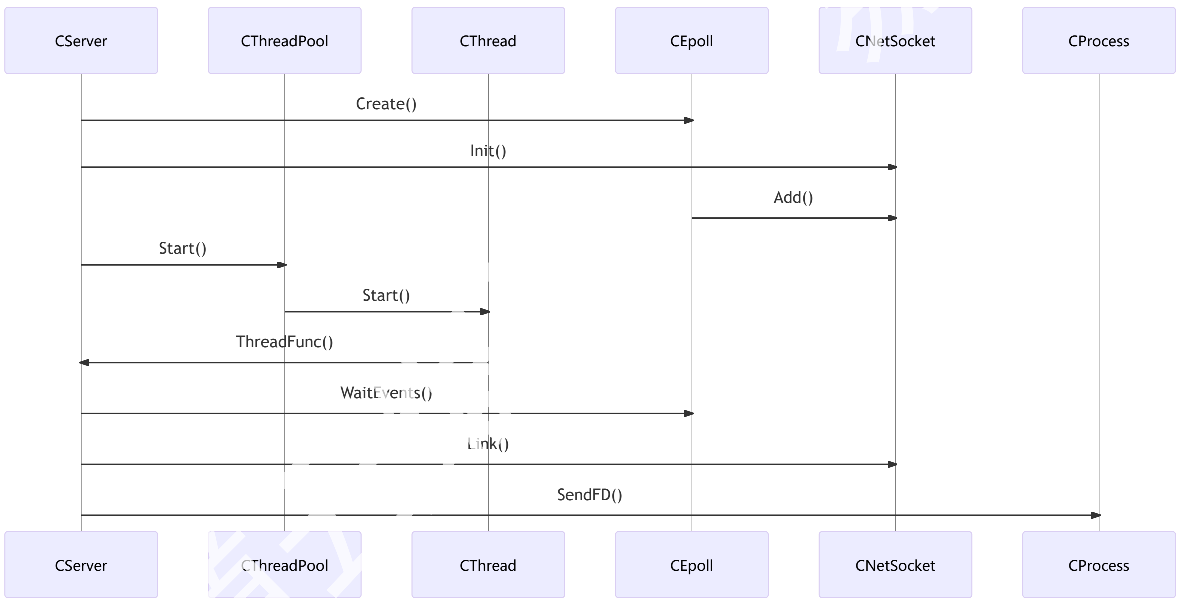
主模块主要就是**客户端的接入**，然后分发客户端到客户端处理进程去处理

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：

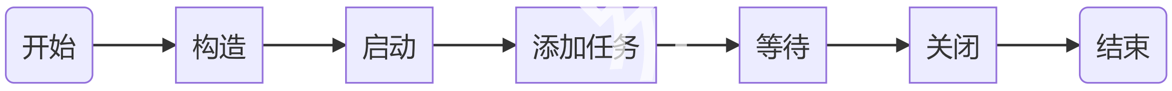


网络服务器逻辑则要复杂一点



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"
```

```
#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;
};
```

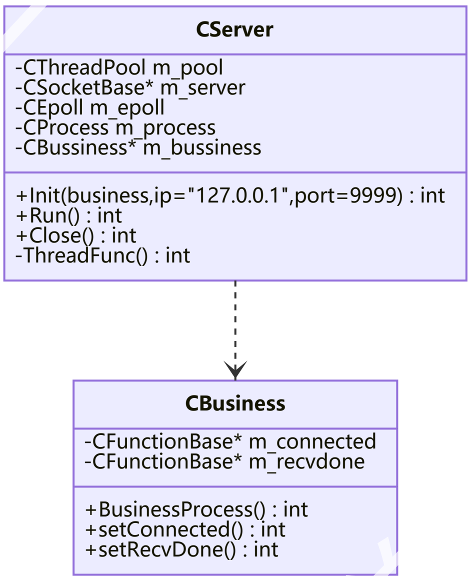
线程池的实现

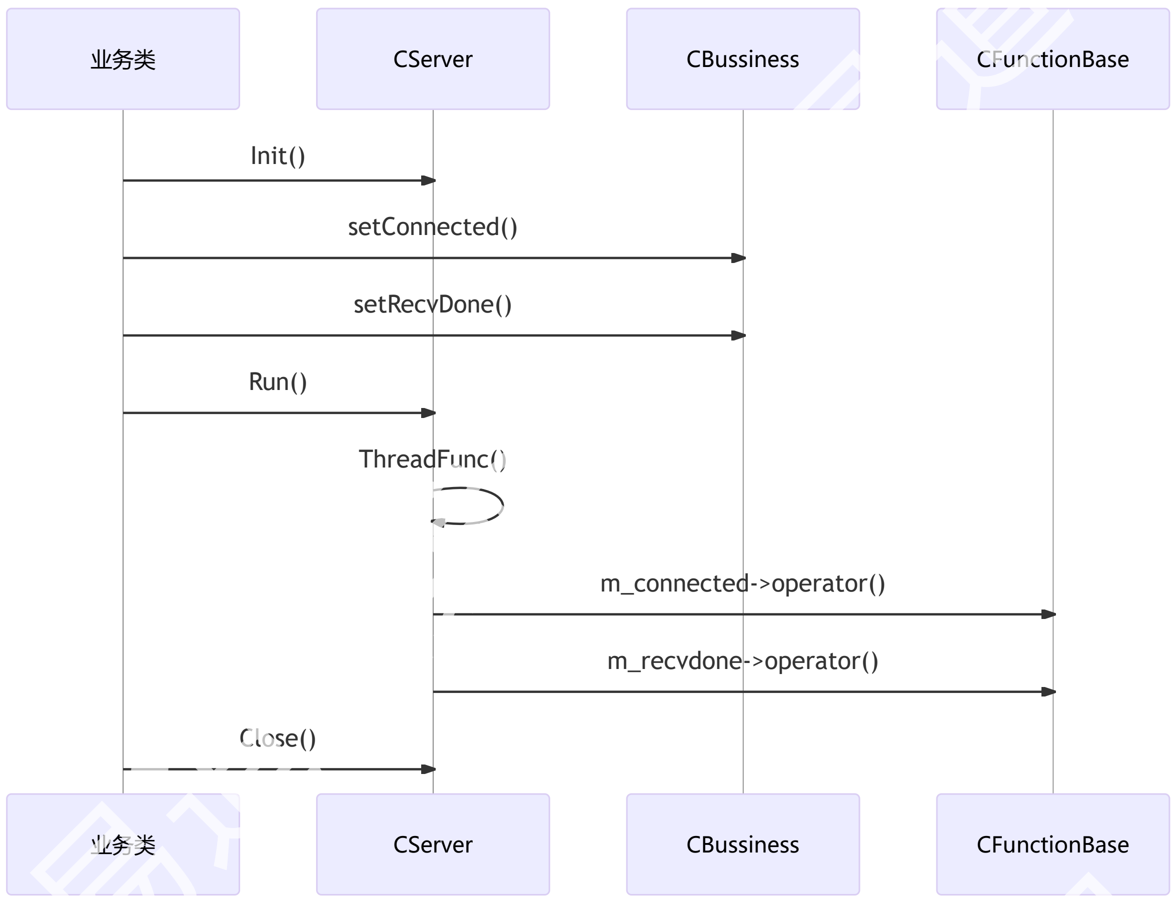
```
#pragma once
#include "Epoll.h"
#include "Thread.h"
#include "Function.h"
#include "Socket.h"
```

```

class CThreadPool
{
public:
    CThreadPool() {
        m_server = NULL;
        timespec tp = { 0,0 };
        clock_gettime(CLOCK_REALTIME, &tp);
        char* buf = NULL;
        asprintf(&buf, "%d.%d.sock", tp.tv_sec % 100000, tp.tv_nsec % 1000000);
        if (buf != NULL) {
            m_path = buf;
            free(buf);
        }//有问题的话，在start接口里面判断m_path来解决问题。
        usleep(1);
    }
    ~CThreadPool() {
        Close();
    }
    CThreadPool(const CThreadPool&) = delete;
    CThreadPool& operator=(const CThreadPool&) = delete;
public:
    int Start(unsigned count) {
        int ret = 0;
        if (m_server != NULL)return -1;//已经初始化了
        if (m_path.size() == 0)return -2;//构造函数失败！！！
        m_server = new CLocalSocket();
        if (m_server == NULL)return -3;
        ret = m_server->Init(CSockParam(m_path, SOCK_ISSERVER));
        if (ret != 0)return -4;
        ret = m_epoll.Create(count);
        if (ret != 0)return -5;
        ret = m_epoll.Add(*m_server, EpollData((void*)m_server));
        if (ret != 0)return -6;
        m_threads.resize(count);
        for (unsigned i = 0; i < count; i++) {
            m_threads[i] = new CThread(&CThreadPool::TaskDispatch, this);
            if (m_threads[i] == NULL)return -7;
            ret = m_threads[i]->Start();
            if (ret != 0)return -8;
        }
        return 0;
    }
    void Close() {
        m_epoll.Close();
        if (m_server) {
            CSocketBase* p = m_server;
            m_server = NULL;
            delete p;
        }
        for (auto thread : m_threads)
        {
            if (thread)delete thread;
        }
        m_threads.clear();
        unlink(m_path);
    }
    template<typename _FUNCTION_, typename... _ARGS_>
    int AddTask(_FUNCTION_ func, _ARGS_... args) {
        static thread_local CLocalSocket client;
        int ret = 0;
        if (client == -1) {
            ret = client.Init(CSockParam(m_path, 0));
            if (ret != 0)return -1;
            ret = client.Link();
            if (ret != 0)return -2;
        }
        CFunctionBase* base = new CFunction<_FUNCTION_, _ARGS_...>(func, args...);
        if (base == NULL)return -3;
        Buffer data(sizeof(base));
        memcpy(data, &base, sizeof(base));
        ret = client.Send(data);
        if (ret != 0) {
            delete base;
            return -4;
        }
        return 0;
    }
private:
    int TaskDispatch() {
        while (m_epoll != -1) {
            EPEvents events;
            int ret = 0;
            ssize_t esize = m_epoll.WaitEvents(events);
            if (esize > 0) {
                for (ssize_t i = 0; i < esize; i++) {
                    if (events[i].events & EPOLLIN) {
                        CSocketBase* pClient = NULL;
                        if (events[i].data.ptr == m_server) {//客户端请求连接

                            ret = m_server->Link(&pClient);
                            if (ret != 0)continue;
                            ret = m_epoll.Add(*pClient, EpollData((void*)pClient));
                            if (ret != 0) {
                                delete pClient;
                                continue;
                            }
                        }
                        else {//客户端的数据来了
                            pClient = (CSocketBase*)events[i].data.ptr;
                            if (pClient) {
                                CFunctionBase* base = NULL;
                                Buffer data(sizeof(base));
                                ret = pClient->Recv(data);
                                if (ret <= 0) {
                                    m_epoll.Del(*pClient);
                                    delete pClient;
                                    continue;
                                }
                                memcpy(&base, (char*)data, sizeof(base));
                                if (base != NULL) {
                                    (*base)();
                                    delete base;
                                }
                            }
                        }
                    }
                }
            }
```

```cpp
        }
        return 0;
    }
private:
    CEpoll m_epoll;
    std::vector<CThread*> m_threads;
    CSocketBase* m_server;
    Buffer m_path;
};
```

主模块的实现





CServer.h

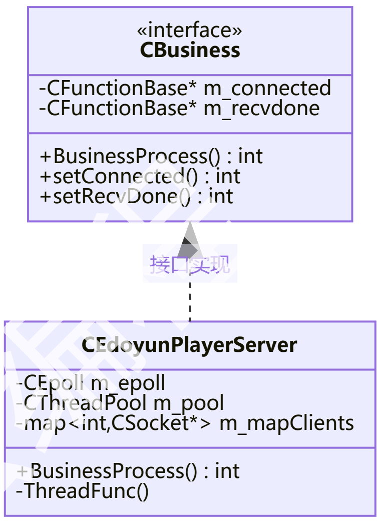
```cpp
#pragma once
#include "Socket.h"
#include "Epoll.h"
#include "ThreadPool.h"
#include "Process.h"
class CBusiness
{
public:
    virtual int BusinessProcess() = 0;
    template<typename _FUNCTION_, typename...
_ARGS_>
    int setConnectedCallback(_FUNCTION_ func,
_ARGS_... args) {
        m_connectedcallback = new CFunction<_FUNCTION_, _ARGS_...>(func, args...);
        if (m_connectedcallback == NULL)return -1;
        return 0;
    }
    template<typename _FUNCTION_, typename... _ARGS_>
    int setRecvCallback(_FUNCTION_ func, _ARGS_... args) {
        m_recvcallback = new CFunction<_FUNCTION_, _ARGS_...>(func, args...);
        if (m_recvcallback == NULL)return -1;
        return 0;
    }
private:
    CFunctionBase* m_connectedcallback;
    CFunctionBase* m_recvcallback;
};

class CServer
{
public:
    CServer();
    ~CServer() { Close(); }
    CServer(const CServer&) = delete;
    CServer& operator=(const CServer&) = delete;
public:
    int Init(CBusiness* business, const Buffer& ip = "127.0.0.1", short port = 9999);
    int Run();
    int Close();
private:
    int ThreadFunc();
private:
    CThreadPool m_pool;
    CSocketBase* m_server;
    CEpoll m_epoll;
    CProcess m_process;
    CBusiness* m_business;//业务模块 需要我们手动
delete
};


```

CServer.cpp

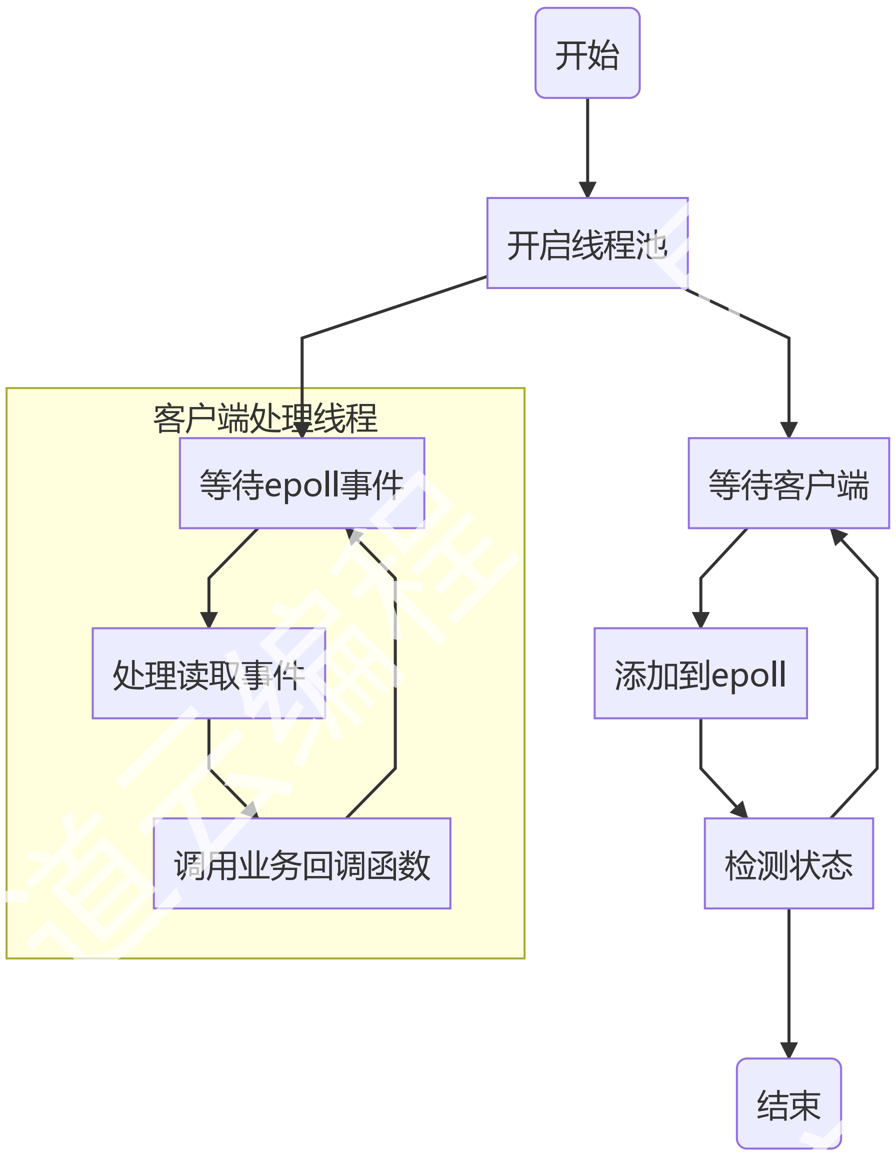
```
#include "CServer.h"
#include "Logger.h"

CServer::CServer()
{
    m_server = NULL;
    m_business = NULL;
}

int CServer::Init(CBusiness* business, const
Buffer& ip, short port)
{
    int ret = 0;
    if (business == NULL)return -1;
    m_business = business;
    ret =
m_process.SetEntryFunction(&CBusiness::BusinessPr
ocess, m_business);
    if (ret != 0)return -2;
    ret = m_process.CreateSubProcess();
    if (ret != 0)return -3;
    ret = m_pool.Start(2);
    if (ret != 0)return -4;
    ret = m_epoll.Create(2);
    if (ret != 0)return -5;
    m_server = new CSocket();
    if (m_server == NULL)return -6;
    ret = m_server->Init(CSockParam(ip, port, SOCK_ISSERVER | SOCK_ISIP));
    if (ret != 0)return -7;
    ret = m_epoll.Add(*m_server, EpollData((void*)m_server));
    if (ret != 0)return -8;
    for (size_t i = 0; i < m_pool.Size(); i++) {
        ret = m_pool.AddTask(&CServer::ThreadFunc, this);
        if (ret != 0)return -9;
    }
    return 0;
}

int CServer::Run()
{
    while (m_server != NULL) {
        usleep(10);
    }
    return 0;
}

int CServer::Close()
{
    if (m_server) {
        CSocketBase* sock = m_server;
        m_server = NULL;
        m_epoll.Del(*sock);
        delete sock;
    }
    m_epoll.Close();
    m_process.SendFD(-1);
    m_pool.Close();
    return 0;
}

int CServer::ThreadFunc()
```

```
{
    int ret = 0;
    EPEvents events;
    while ((m_epoll != -1) && (m_server != NULL)) {
        ssize_t size = m_epoll.WaitEvents(events);
        if (size < 0)break;
        if (size > 0) {
            for (ssize_t i = 0; i < size; i++)
            {
                if (events[i].events & EPOLLERR) {
                    break;
                }
                else if (events[i].events & EPOLLIN) {
                    if (m_server) {
                        CSocketBase* pClient = NULL;
                        ret = m_server->Link(&pClient);
                        if (ret != 0)continue;
                        ret = m_process.SendFD(*pClient);
                        delete pClient;
                        if (ret != 0) {
                            TRACEE("send client %d failed!", (int)*pClient);
                            continue;
                        }
                    }
                }
            }
        }
    }
    return 0;
}

```

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
*/
```

```cpp
#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::ConnectedDone, 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) {
            ssize_t size = m_epoll.WaitEvents(events);
            if (size < 0)break;
            if (size > 0) {
                for (ssize_t i = 0; i < size; i++)
                {
                    if (events[i].events & EPOLLERR) {
                        break;
                    }
                    else if (events[i].events & EPOLLIN) {
                        CSocketBase* pClient = (CSocketBase*)events[i].data.ptr;
                        if (pClient) {
                            Buffer data;
                            ret = pClient->Recv(data);
                            WARN_CONTINUE(ret);
                            if (m_recvcallback) {
                                (*m_recvcallback)(pClient, data);
                            }
                        }
                    }
                }
            }
        }
        return 0;
    }
private:
    CEpoll m_epoll;
    std::map<int, CSocketBase*> m_mapClients;
    CThreadPool m_pool;
    unsigned m_count;
};
```

Function.h

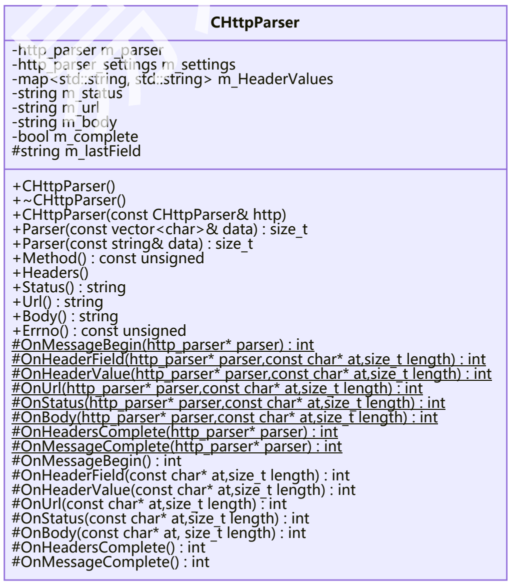
```
#pragma once
#include <unistd.h>
#include <sys/types.h>
#include <functional>

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;
};
```

```

template<typename _FUNCTION_, typename... _ARGS_>
class CConnectedFunction :public CFunctionBase
{
public:
    CConnectedFunction(_FUNCTION_ func, _ARGS_... args)
        :m_binder(std::forward<_FUNCTION_>(func), std::forward<_ARGS_>(args)...)
    {}
    virtual ~CConnectedFunction() {}

    virtual int operator()(CSocketBase* pClient) {
        return m_binder(pClient);
    }

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

template<typename _FUNCTION_, typename... _ARGS_>
class CRecvFunction :public CFunctionBase
{
public:
    CRecvFunction(_FUNCTION_ func, _ARGS_... args)
        :m_binder(std::forward<_FUNCTION_>(func), std::forward<_ARGS_>(args)...)
    {}
    virtual ~CRecvFunction() {}

    virtual int operator()(CSocketBase* pClient, const Buffer& data) {
        return m_binder(pClient, data);
    }

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

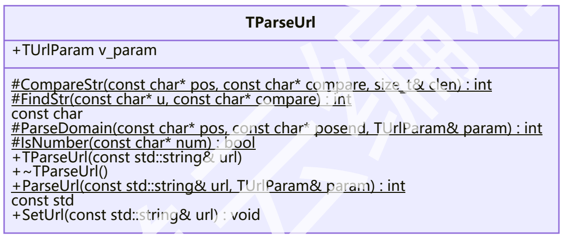
HTTP模块的设计

封装的作用

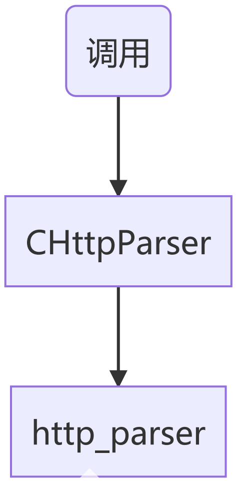
- 降低使用成本
- 对外屏蔽细节（**低耦合**）
- 增加可以移植性
- 与更多同类数据关联（**高内聚**）



```
CHttpParser
-http_parser m_parser
-http_parser_settings m_settings
-map<std::string, std::string> m_HeaderValues
-string m_status
-string m_url
-string m_body
-bool m_complete
#string m_lastField
+CHttpParser()
+~CHttpParser()
+CHttpParser(const CHttpParser& http)
+Parser(const vector<char>& data) : size_t
+Parser(const string& data) : size_t
+Method() : const unsigned
+Headers()
+Status() : string
+Url() : string
+Body() : string
+Errno() : const unsigned
#OnMessageBegin(http_parser* parser) : int
#OnHeaderField(http_parser* parser,const char* at,size_t length) : int
#OnHeaderValue(http_parser* parser,const char* at,size_t length) : int
#OnUrl(http_parser* parser,const char* at,size_t length) : int
#OnStatus(http_parser* parser,const char* at,size_t length) : int
#OnBody(http_parser* parser,const char* at,size_t length) : int
#OnHeadersComplete(http_parser* parser) : int
#OnMessageComplete(http_parser* parser) : int
#OnMessageBegin() : int
#OnHeaderField(const char* at,size_t length) : int
#OnHeaderValue(const char* at,size_t length) : int
#OnUrl(const char* at,size_t length) : int
#OnStatus(const char* at,size_t length) : int
#OnBody(const char* at, size_t length) : int
#OnHeadersComplete() : int
#OnMessageComplete() : int
```



```
TParseUrl
+TUrlParam v_param
#CompareStr(const char* pos, const char* compare, size_t& clen) : int
#FindStr(const char* u, const char* compare) : int
const char
#ParseDomain(const char* pos, const char* posend, TUrlParam& param) : int
#IsNumber(const char* num) : bool
+TParseUrl(const std::string& url)
+~TParseUrl()
+ParseUrl(const std::string& url, TUrlParam& param) : int
const std
+SetUrl(const std::string& url) : void
```



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);
```

```
    static int OnMessageComplete(http_parser* parser);
    int OnMessageBegin();
    int OnUrl(const char* at, size_t length);
    int OnStatus(const char* at, size_t length);
    int OnHeaderField(const char* at, size_t length);
    int OnHeaderValue(const char* at, size_t length);
    int OnHeadersComplete();
    int OnBody(const char* at, size_t length);
    int OnMessageComplete();
};

class UrlParser
{
public:
    UrlParser(const Buffer& url);
    ~UrlParser() {}
    int Parser();
    Buffer operator[](const Buffer& name)const;
    Buffer Protocol()const { return m_protocol; }
    Buffer Host()const { return m_host; }
    //默认返回80
    int Port()const { return m_port; }
    void SetUrl(const Buffer& url);
private:
    Buffer m_url;
    Buffer m_protocol;
    Buffer m_host;
    Buffer m_uri;
    int m_port;
    std::map<Buffer, Buffer> m_values;
};
```

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;
    m_url = http.m_url;
    m_body = http.m_body;
    m_complete = http.m_complete;
    m_lastField = http.m_lastField;
}

CHttpParser& CHttpParser::operator=(const CHttpParser& http)
{
    if (this != &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;
        m_url = http.m_url;
        m_body = http.m_body;
        m_complete = http.m_complete;
        m_lastField = http.m_lastField;
    }
    return *this;
}

size_t CHttpParser::Parser(const Buffer& data)
{
    m_complete = false;
    size_t ret = http_parser_execute(
        &m_parser, &m_settings, data, data.size());
    if (m_complete == false) {
        m_parser.http_errno = 0x7F;
        return 0;
    }
    return ret;
}

```

```
int CHttpParser::OnMessageBegin(http_parser* parser)
{
    return ((CHttpParser*)parser->data)->OnMessageBegin();
}

int CHttpParser::OnUrl(http_parser* parser, const char* at, size_t length)
{
    return ((CHttpParser*)parser->data)->OnUrl(at, length);
}

int CHttpParser::OnStatus(http_parser* parser, const char* at, size_t length)
{
    return ((CHttpParser*)parser->data)->OnStatus(at, length);
}

int CHttpParser::OnHeaderField(http_parser* parser, const char* at, size_t length)
{
    return ((CHttpParser*)parser->data)->OnHeaderField(at, length);
}

int CHttpParser::OnHeaderValue(http_parser* parser, const char* at, size_t length)
{
    return ((CHttpParser*)parser->data)->OnHeaderValue(at, length);
}

int CHttpParser::OnHeadersComplete(http_parser* parser)
```

```
{
    return ((CHttpParser*)parser->data)->OnHeadersComplete();
}

int CHttpParser::OnBody(http_parser* parser, const char* at, size_t length)
{
    return ((CHttpParser*)parser->data)->OnBody(at, length);
}

int CHttpParser::OnMessageComplete(http_parser* parser)
{
    return ((CHttpParser*)parser->data)->OnMessageComplete();
}

int CHttpParser::OnMessageBegin()
{
    return 0;
}

int CHttpParser::OnUrl(const char* at, size_t length)
{
    m_url = Buffer(at, length);
    return 0;
}

int CHttpParser::OnStatus(const char* at, size_t length)
{
    m_status = Buffer(at, length);
    return 0;
}
```

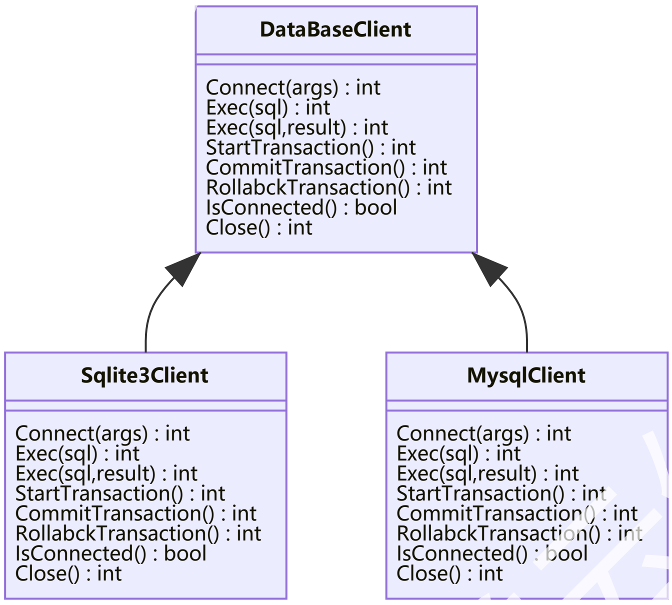
```cpp

int CHttpParser::OnHeaderField(const char* at, size_t length)
{
    m_lastField = Buffer(at, length);
    return 0;
}

int CHttpParser::OnHeaderValue(const char* at, size_t length)
{
    m_HeaderValues[m_lastField] = Buffer(at, length);
    return 0;
}

int CHttpParser::OnHeadersComplete()
{
    return 0;
}

int CHttpParser::OnBody(const char* at, size_t length)
{
    m_body = Buffer(at, length);
    return 0;
}

int CHttpParser::OnMessageComplete()
{
    m_complete = true;
    return 0;
}

UrlParser::UrlParser(const Buffer& url)
{
    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;
```

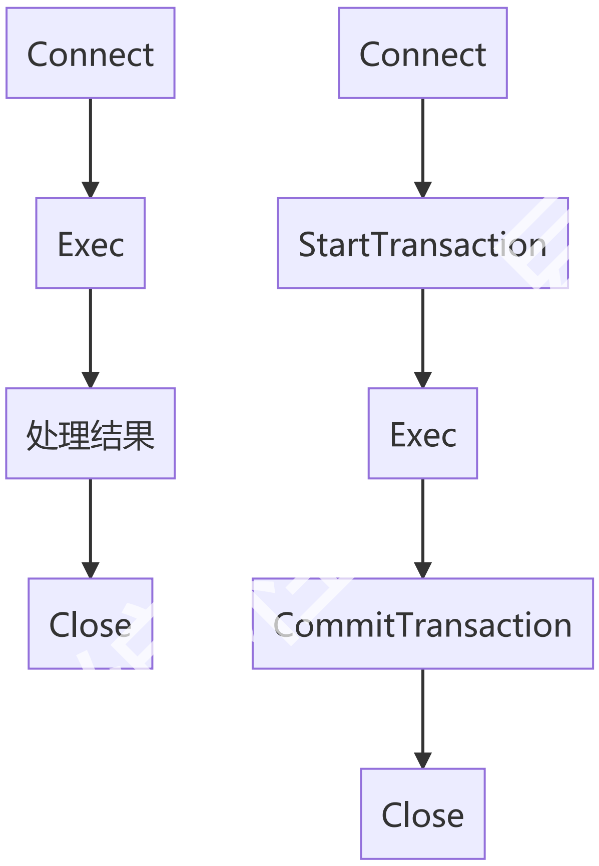
```cpp
    }
    else {
        m_uri = Buffer(pos, target);
        //解析key和value
        pos = target + 1;
        const char* t = NULL;
        do {
            target = strchr(pos, '&');
            if (target == NULL) {
                t = strchr(pos, '=');
                if (t == NULL)return -4;
                m_values[Buffer(pos, t)] = Buffer(t + 1);
            }
            else {
                Buffer kv(pos, target);
                t = strchr(kv, '=');
                if (t == NULL)return -5;
                m_values[Buffer(kv, t)] = Buffer(t + 1, kv + kv.size());
                pos = target + 1;
            }
        } while (target != NULL);
    }

    return 0;
}

Buffer UrlParser::operator[](const Buffer& name) const
{
    auto it = m_values.find(name);
    if (it == m_values.end())return Buffer();
    return it->second;
}

void UrlParser::SetUrl(const Buffer& url)
```

```
{
    m_url = url;
    m_protocol = "";
    m_host = "";
    m_port = 80;
    m_values.clear();
}

```

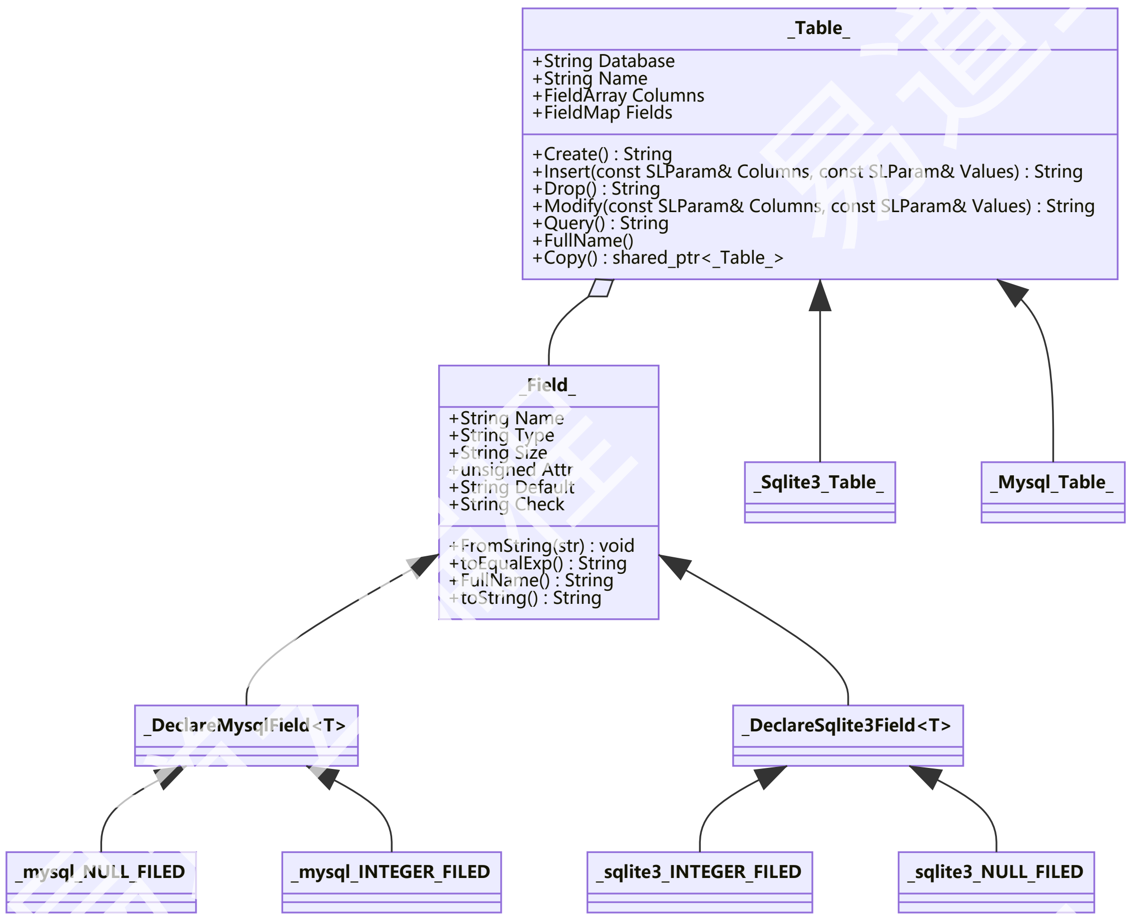
数据库模块的设计



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; }

#define DECLARE_TABLE_CLASS_EDN() }};

DECLARE_TABLE_CLASS(edoyunLogin_user_mysql, _Mysql_Table_)
DECLARE_ITEM(user_id, NOT_NULL | PRIMARY_KEY | AUTOINCREMENT, INTEGER_FILED, "", "", "")
DECLARE_ITEM(user_qq, NOT_NULL, VARCHAR_FILED, "(15)", "", "")  //QQ号
DECLARE_ITEM(user_phone, DEFAULT, VARCHAR_FILED, "(11)", "'18888888888'", "")  //手机
DECLARE_ITEM(user_name, NOT_NULL, TEXT_FILED, "", "", "")     //姓名
DECLARE_ITEM(user_nick, NOT_NULL, TEXT_FILED, "", "", "")     //昵称
DECLARE_ITEM(user_wechat, DEFAULT, TEXT_FILED, "", "NULL", "")
DECLARE_ITEM(user_wechat_id, DEFAULT, TEXT_FILED, "", "NULL", "")
DECLARE_ITEM(user_address, DEFAULT, TEXT_FILED, "", "", "")
DECLARE_ITEM(user_province, DEFAULT, TEXT_FILED, "", "", "")
DECLARE_ITEM(user_country, DEFAULT, TEXT_FILED, "", "", "")
DECLARE_ITEM(user_age, DEFAULT | CHECK, INTEGER_FILED, "", "18", "")
DECLARE_ITEM(user_male, DEFAULT, BOOL_FILED, "", "1", "")
DECLARE_ITEM(user_flags, DEFAULT, TEXT_FILED, "", "0", "")
DECLARE_ITEM(user_experience, DEFAULT, REAL_FILED, "", "0.0", "")
DECLARE_ITEM(user_level, DEFAULT | CHECK, INTEGER_FILED, "", "0", "")
DECLARE_ITEM(user_class_priority, DEFAULT, TEXT_FILED, "", "", "")
DECLARE_ITEM(user_time_per_viewer, DEFAULT, REAL_FILED, "", "", "")
DECLARE_ITEM(user_career, NONE, TEXT_FILED, "", "", "")
DECLARE_ITEM(user_password, NOT_NULL, TEXT_FILED, "", "", "")
DECLARE_ITEM(user_birthday, NONE, DATETIME_FILED, "", "", "")
DECLARE_ITEM(user_describe, NONE, TEXT_FILED, "", "", "")
DECLARE_ITEM(user_education, NONE, TEXT_FILED, "", "", "")
DECLARE_ITEM(user_register_time, DEFAULT, DATETIME_FILED, "", "LOCALTIME()", "")
DECLARE_TABLE_CLASS_END()

DECLARE_TABLE_CLASS(edoyunLogin_user_test, _Sqlite3_Table_)
DECLARE_ITEM(user_id, NOT_NULL | PRIMARY_KEY | AUTOINCREMENT, INTEGER_FILED, "", "", "")
DECLARE_ITEM(user_qq, NOT_NULL, VARCHAR_FILED, "(15)", "", "")  //QQ号
DECLARE_ITEM(user_phone, DEFAULT, TEXT_FILED, "", "18888888888", "")  //手机
DECLARE_ITEM(user_name, NOT_NULL, TEXT_FILED, "", "", "")    //姓名
DECLARE_ITEM(user_nick, NOT_NULL, TEXT_FILED, "", "", "")    //昵称
DECLARE_ITEM(user_wechat, DEFAULT, TEXT_FILED, "", "none", "")
DECLARE_ITEM(user_wechat_id, DEFAULT, TEXT_FILED, "", "none", "")
DECLARE_ITEM(user_address, DEFAULT, TEXT_FILED, "", "\"长安大街1号\"", "")
DECLARE_ITEM(user_province, DEFAULT, TEXT_FILED, "", "\"北京\"", "")
DECLARE_ITEM(user_country, DEFAULT, TEXT_FILED, "", "\"中国\"", "")
DECLARE_ITEM(user_age, DEFAULT | CHECK, INTEGER_FILED, "", "18", "\"user_age\" >= 0")
DECLARE_ITEM(user_male, DEFAULT, BOOL_FILED, "", "1", "")
DECLARE_ITEM(user_flags, DEFAULT, TEXT_FILED, "", "0", "")
DECLARE_ITEM(user_experience, DEFAULT, REAL_FILED, "", "0.0", "")
DECLARE_ITEM(user_level, DEFAULT | CHECK, INTEGER_FILED, "", "0", "\"user_level\" >= 0")
DECLARE_ITEM(user_class_priority, DEFAULT, TEXT_FILED, "", "", "")
DECLARE_ITEM(user_time_per_viewer, DEFAULT, REAL_FILED, "", "", "")
DECLARE_ITEM(user_career, NONE, TEXT_FILED, "", "", "")
DECLARE_ITEM(user_password, NOT_NULL, TEXT_FILED, "", "", "")
DECLARE_ITEM(user_birthday, NONE, DATETIME_FILED, "", "", "")
DECLARE_ITEM(user_describe, NONE, TEXT_FILED, "", "", "")
DECLARE_ITEM(user_education, NONE, TEXT_FILED, "", "", "")
DECLARE_ITEM(user_register_time, DEFAULT, DATETIME_FILED, "", "(datetime('now', 'localtime'))", "")
DECLARE_TABLE_CLASS_END()

```

sqlite3数据库的实现

DataBaseHelper.h

```
#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;
    //关闭连接
    virtual int Close() = 0;
    //是否连接
    virtual bool IsConnected() = 0;
};

//表和列的基类的实现
class _Field_;
using PField = std::shared_ptr<_Field_>;
using FieldArray = std::vector<PField>;
using FieldMap = std::map<Buffer, PField>;


class _Table_ {
public:
    _Table_() {}
    virtual ~_Table_() {}
    //返回创建的SQL语句
    virtual Buffer Create() = 0;
    //删除表
    virtual Buffer Drop() = 0;
    //增删改查
    //TODO:参数进行优化
    virtual Buffer Insert(const _Table_& values) = 0;
    virtual Buffer Delete(const _Table_& values) = 0;
    //TODO:参数进行优化
    virtual Buffer Modify(const _Table_& values) = 0;
    virtual Buffer Query() = 0;
    //创建一个基于表的对象
    virtual PTable Copy()const = 0;
    virtual void ClearFieldUsed() = 0;
public:
    //获取表的全名
    virtual operator const Buffer() const = 0;
```

```cpp
public:
    //表所属的DB的名称
    Buffer Database;
    Buffer Name;
    FieldArray FieldDefine;//列的定义（存储查询结
果）
    FieldMap Fields;//列的定义映射表
};

enum {
    SQL_INSERT = 1,//插入的列
    SQL_MODIFY = 2,//修改的列
    SQL_CONDITION = 4//查询条件列
};

enum {
    NOT_NULL = 1,
    DEFAULT = 2,
    UNIQUE = 4,
    PRIMARY_KEY = 8,
    CHECK = 16,
    AUTOINCREMENT = 32
};

using SqlType = enum {
    TYPE_NULL = 0,
    TYPE_BOOL = 1,
    TYPE_INT = 2,
    TYPE_DATETIME = 4,
    TYPE_REAL = 8,
    TYPE_VARCHAR = 16,
    TYPE_TEXT = 32,
    TYPE_BLOB = 64
};

class _Field_
{
```

```
public:
    _Field_() {}
    _Field_(const _Field_& field) {
        Name = field.Name;
        Type = field.Type;
        Attr = field.Attr;
        Default = field.Default;
        Check = field.Check;
    }
    virtual _Field_& operator=(const _Field_&
field) {
        if (this != &field) {
            Name = field.Name;
            Type = field.Type;
            Attr = field.Attr;
            Default = field.Default;
            Check = field.Check;
        }
        return *this;
    }
    virtual ~_Field_() {}
public:
    virtual Buffer Create() = 0;
    virtual void LoadFromStr(const Buffer& str)
= 0;
    //where 语句使用的
    virtual Buffer toEqualExp() const = 0;
    virtual Buffer toSqlStr() const = 0;
    //列的全名
    virtual operator const Buffer() const = 0;
public:
    Buffer Name;
    Buffer Type;
    Buffer Size;
    unsigned Attr;
    Buffer Default;
    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);
    //带结果的执行
    virtual int Exec(const Buffer& sql, Result& result, const _Table_& table);
    //开启事务
    virtual int StartTransaction();
    //提交事务
    virtual int CommitTransaction();
    //回滚事务
    virtual int RollbackTransaction();
    //关闭连接
    virtual int Close();
    //是否连接 true表示连接中 false表示未连接
    virtual bool IsConnected();
private:
    static int ExecCallback(void* arg, int count, char** names, char** values);
    int ExecCallback(Result& result, const _Table_& table, int count, char** names, char** values);
private:
    sqlite3_stmt* m_stmt;
    sqlite3* m_db;
private:
    class ExecParam {
    public:
        ExecParam(CSqlite3Client* obj, Result& result, const _Table_& table)
            :obj(obj), result(result), table(table)
        {}
        CSqlite3Client* obj;
        Result& result;
        const _Table_& table;
    };
};

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,
        unsigned attr,
        const Buffer& type,
        const Buffer& size,
        const Buffer& default_,
        const Buffer& check
    );
    _sqlite3_field_(const _sqlite3_field_& field);
    virtual ~_sqlite3_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: \
virtual PTable Copy() const {return PTable(new name(*this));} \
name():base(){Name=#name;

#define DECLARE_FIELD(ntype,name,attr,type,size,default_,check) \
{PField field(new _sqlite3_field_(ntype, #name, attr, type, size, default_, check));FieldDefine.push_back(field);Fields[#name] = field; }

#define DECLARE_TABLE_CLASS_EDN() }};
```

Sqlite3Client.cpp

```
#include "Sqlite3Client.h"
#include "Logger.h"

int CSqlite3Client::Connect(const KeyValue& args)
{
    auto it = args.find("host");
    if (it == args.end())return -1;
    if (m_db != NULL)return -2;
    int ret = sqlite3_open(it->second, &m_db);
    if (ret != 0) {
        TRACEE("connect failed:%d [%s]", ret, sqlite3_errmsg(m_db));
        return -3;
    }
    return 0;
}

int CSqlite3Client::Exec(const Buffer& sql)
{
    printf("sql={%s}\n", (char*)sql);
    if (m_db == NULL)return -1;
    int ret = sqlite3_exec(m_db, sql, NULL, this, NULL);
    if (ret != SQLITE_OK) {
        printf("sql={%s}\n", (char*)sql);
        printf("Exec failed:%d [%s]\n", ret, sqlite3_errmsg(m_db));
        return -2;
    }
    return 0;
}

int CSqlite3Client::Exec(const Buffer& sql, Result& result, const _Table_& table)
{
    char* errmsg = NULL;
    if (m_db == NULL)return -1;
    printf("sql={%s}\n", (char*)sql);
    ExecParam param(this, result, table);
    int ret = sqlite3_exec(m_db, sql,
        &CSqlite3Client::ExecCallback, (void*)&param, &errmsg);
    if (ret != SQLITE_OK) {
        printf("sql={%s}\n", sql);
        printf("Exec failed:%d [%s]\n", ret, errmsg);
        if (errmsg)sqlite3_free(errmsg);
        return -2;
    }
    if (errmsg)sqlite3_free(errmsg);
    return 0;
}

int CSqlite3Client::StartTransaction()
{
    if (m_db == NULL)return -1;
    int ret = sqlite3_exec(m_db, "BEGIN TRANSACTION", 0, 0, NULL);
    if (ret != SQLITE_OK) {
        TRACEE("sql={BEGIN TRANSACTION}");
        TRACEE("BEGIN failed:%d [%s]", ret, sqlite3_errmsg(m_db));
        return -2;
    }
    return 0;
}

int CSqlite3Client::CommitTransaction()
{
    if (m_db == NULL)return -1;
    int ret = sqlite3_exec(m_db, "COMMIT 
TRANSACTION", 0, 0, NULL);
    if (ret != SQLITE_OK) {
        TRACEE("sql={COMMIT TRANSACTION}");
        TRACEE("COMMIT failed:%d [%s]", ret, 
sqlite3_errmsg(m_db));
        return -2;
    }
    return 0;
}

int CSqlite3Client::RollbackTransaction()
{
    if (m_db == NULL)return -1;
    int ret = sqlite3_exec(m_db, "ROLLBACK 
TRANSACTION", 0, 0, NULL);
    if (ret != SQLITE_OK) {
        TRACEE("sql={ROLLBACK TRANSACTION}");
        TRACEE("ROLLBACK failed:%d [%s]", ret, 
sqlite3_errmsg(m_db));
        return -2;
    }
    return 0;
}

int CSqlite3Client::Close()
{
    if (m_db == NULL)return -1;
    int ret = sqlite3_close(m_db);
    if (ret != SQLITE_OK) {
        TRACEE("Close failed:%d [%s]", ret, sqlite3_errmsg(m_db));
        return -2;
    }
    m_db = NULL;
    return 0;
}

bool CSqlite3Client::IsConnected()
{
    return m_db != NULL;
}

int CSqlite3Client::ExecCallback(void* arg, int count, char** values, char** names)
{
    ExecParam* param = (ExecParam*)arg;
    return param->obj->ExecCallback(param->result, param->table, count, names, values);
}

int CSqlite3Client::ExecCallback(Result& result, const _Table_& table, int count, char** names, char** values)
{
    PTable pTable = table.Copy();
    if (pTable == nullptr) {
        printf("table %s error!\n", (const char*)(Buffer)table);
        return -1;
    }
    for (int i = 0; i < count; i++) {
        Buffer name = names[i];
        auto it = pTable->Fields.find(name);
        if (it == pTable->Fields.end()) {
            printf("table %s error!\n", (const char*)(Buffer)table);
```

```
                return -2;
            }
            if (values[i] != NULL)
                it->second->LoadFromStr(values[i]);
        }
        result.push_back(pTable);
        return 0;
}

_sqlite3_table_::_sqlite3_table_(const
_sqlite3_table_& table)
{
    Database = table.Database;
    Name = table.Name;
    for (size_t i = 0; i <
table.FieldDefine.size(); i++)
    {
        PField field = PField(new
_sqlite3_field_(*

(_sqlite3_field_*)table.FieldDefine[i].get()));
        FieldDefine.push_back(field);
        Fields[field->Name] = field;
    }
}

_sqlite3_table_::~_sqlite3_table_()
{}

Buffer _sqlite3_table_::Create()
{   //CREATE TABLE IF NOT EXISTS 表全名 (列定
义,……);
    //表全名 = 数据库.表名
    Buffer sql = "CREATE TABLE IF NOT EXISTS " +
(Buffer)*this + "(\r\n";
    for (size_t i = 0; i < FieldDefine.size();
i++) {
        if (i > 0)sql += ",";
        sql += FieldDefine[i]->Create();
    }
    sql += ");";
    TRACEI("sql = %s", (char*)sql);
    return sql;
}

Buffer _sqlite3_table_::Drop()
{
    Buffer sql = "DROP TABLE " + (Buffer)*this + ";";
    TRACEI("sql = %s", (char*)sql);
    return sql;
}

Buffer _sqlite3_table_::Insert(const _Table_& values)
{   //INSERT INTO 表全名 (列1,...,列n)
    //VALUES(值1,...,值n);
    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 += " );";
    TRACEI("sql = %s", (char*)sql);
    return sql;
}

Buffer _sqlite3_table_::Delete(const _Table_& values)
{// DELETE FROM 表全名 WHERE 条件
    Buffer sql = "DELETE FROM " + (Buffer)*this + " ";
    Buffer Where = "";
    bool isfirst = true;
    for (size_t i = 0; i < FieldDefine.size(); i++) {
        if (FieldDefine[i]->Condition & SQL_CONDITION) {
            if (!isfirst)Where += " AND ";
            else isfirst = false;
            Where += (Buffer)*FieldDefine[i] + "=" + FieldDefine[i]->toSqlStr();
        }
    }
    if (Where.size() > 0)
        sql += " WHERE " + Where;
    sql += ";";
    TRACEI("sql = %s", (char*)sql);
    return sql;
}

```

```
Buffer _sqlite3_table_::Modify(const _Table_& values)
{
    //UPDATE 表全名 SET 列1=值1 , ... , 列n=值n [WHERE 条件];
    Buffer sql = "UPDATE " + (Buffer)*this + " SET ";
    bool isfirst = true;
    for (size_t i = 0; i < values.FieldDefine.size(); i++) {
        if (values.FieldDefine[i]->Condition & SQL_MODIFY) {
            if (!isfirst)sql += ",";
            else isfirst = false;
            sql += (Buffer)*values.FieldDefine[i] + "=" + values.FieldDefine[i]->toSqlStr();
        }
    }

    Buffer Where = "";
    for (size_t i = 0; i < values.FieldDefine.size(); i++) {
        if (values.FieldDefine[i]->Condition & SQL_CONDITION) {
            if (!isfirst)Where += " AND ";
            else isfirst = false;
            Where += (Buffer)*values.FieldDefine[i] + "=" + values.FieldDefine[i]->toSqlStr();
        }
    }
    if (Where.size() > 0)
        sql += " WHERE " + Where;
    sql += " ;";
    TRACEI("sql = %s", (char*)sql);
    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 + '"';
}
```

```

_sqlite3_field_::_sqlite3_field_()
    :_Field_() {
    nType = TYPE_NULL;
    Value.Double = 0.0;
}

_sqlite3_field_::_sqlite3_field_(int ntype,
const Buffer& name, unsigned attr, const Buffer&
type, const Buffer& size, const Buffer&
default_, const Buffer& check)
{
    nType = ntype;
    switch (ntype)
    {
    case TYPE_VARCHAR:
    case TYPE_TEXT:
    case TYPE_BLOB:
        Value.String = new Buffer();
        break;
    }

    Name = name;
    Attr = attr;
    Type = type;
    Size = size;
    Default = default_;
    Check = check;
}

_sqlite3_field_::_sqlite3_field_(const
_sqlite3_field_& field)
{
    nType = field.nType;
    switch (field.nType)
    {
    case TYPE_VARCHAR:
```

```
    case TYPE_TEXT:
    case TYPE_BLOB:
        Value.String = new Buffer();
        *Value.String = *field.Value.String;
        break;
    }

    Name = field.Name;
    Attr = field.Attr;
    Type = field.Type;
    Size = field.Size;
    Default = field.Default;
    Check = field.Check;
}

_sqlite3_field_::~_sqlite3_field_()
{
    switch (nType)
    {
    case TYPE_VARCHAR:
    case TYPE_TEXT:
    case TYPE_BLOB:
        if (Value.String) {
            Buffer* p = Value.String;
            Value.String = NULL;
            delete p;
        }
        break;
    }
}

Buffer _sqlite3_field_::Create()
{   //"名称" 类型 属性
    Buffer sql = '"' + Name + "\" " + Type + " ";
    if (Attr & NOT_NULL) {
        sql += " NOT NULL ";
    }
    if (Attr & DEFAULT) {
        sql += " DEFAULT " + Default + " ";
    }
    if (Attr & UNIQUE) {
        sql += " UNIQUE ";
    }
    if (Attr & PRIMARY_KEY) {
        sql += " PRIMARY KEY ";
    }
    if (Attr & CHECK) {
        sql += " CHECK( " + Check + ") ";
    }
    if (Attr & AUTOINCREMENT) {
        sql += " AUTOINCREMENT ";
    }
    return sql;
}

void _sqlite3_field_::LoadFromStr(const Buffer& str)
{
    switch (nType)
    {
    case TYPE_NULL:
        break;
    case TYPE_BOOL:
    case TYPE_INT:
    case TYPE_DATETIME:
        Value.Integer = atoi(str);
        break;
    case TYPE_REAL:
        Value.Double = atof(str);
        break;
    case TYPE_VARCHAR:
    case TYPE_TEXT:
        *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;
    }
    return sql;
}

Buffer _sqlite3_field_::toSqlStr() const
{
    Buffer sql = "";
    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;
    }
    return sql;
}

_sqlite3_field_::operator const Buffer() const
{
    return '"' + Name + '"';
}

Buffer _sqlite3_field_::Str2Hex(const Buffer& data) const
{
    const char* hex = "0123456789ABCDEF";
    std::stringstream ss;
    for (auto ch : data)
        ss << hex[(unsigned char)ch >> 4] << hex[(unsigned char)ch & 0xF];
    return ss.str();
}




```

MySQL数据库的实现

MysqlClient.h

```
#pragma once
#pragma once
#include "Public.h"
#include "DatabaseHelper.h"
#include <mysql/mysql.h>

class CMysqlClient
    :public CDatabaseClient
{
public:
    CMysqlClient(const CMysqlClient&) = delete;
    CMysqlClient& operator=(const CMysqlClient&) = delete;
```

```
public:
    CMysqlClient() {
        bzero(&m_db, sizeof(m_db));
        m_bInit = false;
    }
    virtual ~CMysqlClient() {
        Close();
    }
public:
    //连接
    virtual int Connect(const KeyValue& args);
    //执行
    virtual int Exec(const Buffer& sql);
    //带结果的执行
    virtual int Exec(const Buffer& sql, Result& result, const _Table_& table);
    //开启事务
    virtual int StartTransaction();
    //提交事务
    virtual int CommitTransaction();
    //回滚事务
    virtual int RollbackTransaction();
    //关闭连接
    virtual int Close();
    //是否连接 true表示连接中 false表示未连接
    virtual bool IsConnected();
private:
    MYSQL m_db;
    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
private:
    class ExecParam {
    public:
        ExecParam(CMysqlClient* obj, Result& result, const _Table_& table)
            :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: \
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; }

#define DECLARE_TABLE_CLASS_EDN() }};

```

MysqlClient.cpp

```
#include "MysqlClient.h"
#include <sstream>

int CMysqlClient::Connect(const KeyValue& args)
{
    if (m_bInit)return -1;
    MYSQL* ret = mysql_init(&m_db);
    if (ret == NULL)return -2;
    ret = mysql_real_connect(&m_db,
        args.at("host"), args.at("user"),
        args.at("password"), args.at("db"),
        atoi(args.at("port")),
        NULL, 0);
    if ((ret == NULL) && (mysql_errno(&m_db) != 0)) {
        printf("%s %s\n", __FUNCTION__, mysql_errno(&m_db));
        mysql_close(&m_db);
        bzero(&m_db, sizeof(m_db));
        return -3;
```

```cpp
    }
    m_bInit = true;
    return 0;
}

int CMysqlClient::Exec(const Buffer& sql)
{
    if (!m_bInit)return -1;
    int ret = mysql_real_query(&m_db, sql, sql.size());
    if (ret != 0) {
        printf("%s %s\n", __FUNCTION__, mysql_errno(&m_db));
        return -2;
    }
    return 0;
}

int CMysqlClient::Exec(const Buffer& sql, Result& result, const _Table_& table)
{
    if (!m_bInit)return -1;
    int ret = mysql_real_query(&m_db, sql, sql.size());
    if (ret != 0) {
        printf("%s %s\n", __FUNCTION__, mysql_errno(&m_db));
        return -2;
    }
    MYSQL_RES* res = mysql_store_result(&m_db);
    MYSQL_ROW row;
    unsigned num_fields = mysql_num_fields(res);
    while ((row = mysql_fetch_row(res)) != NULL) {
        PTable pt = table.Copy();
        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_(*
(_mysql_field_*)table.FieldDefine[i].get()));
            FieldDefine.push_back(field);
            Fields[field->Name] = field;
        }
}

_mysql_table_::~_mysql_table_()
{}

Buffer _mysql_table_::Create()
{   //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 );
    Buffer sql = "CREATE TABLE IF NOT EXISTS " + (Buffer)*this + " (\r\n";
    for (unsigned i = 0; i < FieldDefine.size(); i++)
    {
        if (i > 0)sql += ",\r\n";
        sql += FieldDefine[i]->Create();
        if (FieldDefine[i]->Attr & PRIMARY_KEY) {
            sql += ",\r\n PRIMARY KEY (`" + FieldDefine[i]->Name + "`)";
        }
        if (FieldDefine[i]->Attr & UNIQUE) {
            sql += ",\r\n UNIQUE INDEX `" + FieldDefine[i]->Name + "_UNIQUE` (";
            sql += (Buffer)*FieldDefine[i] + " ASC) VISIBLE ";
        }
    }
    sql += ");";
    return sql;
}

```

```cpp
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;
}
```

```

Buffer _mysql_table_::Delete(const _Table_& values)
{
    Buffer sql = "DELETE FROM " + (Buffer)*this + " ";
    Buffer Where = "";
    bool isfirst = true;
    for (size_t i = 0; i < FieldDefine.size(); i++) {
        if (FieldDefine[i]->Condition & SQL_CONDITION) {
            if (!isfirst)Where += " AND ";
            else isfirst = false;
            Where += (Buffer)*FieldDefine[i] + "=" + FieldDefine[i]->toSqlStr();
        }
    }
    if (Where.size() > 0)
        sql += " WHERE " + Where;
    sql += ";";
    printf("sql = %s\r\n", (char*)sql);
    return sql;
}

Buffer _mysql_table_::Modify(const _Table_& values)
{
    Buffer sql = "UPDATE " + (Buffer)*this + " SET ";
    bool isfirst = true;
    for (size_t i = 0; i < values.FieldDefine.size(); i++) {
        if (values.FieldDefine[i]->Condition & SQL_MODIFY) {
            if (!isfirst)sql += ",";
            else isfirst = false;
            sql += (Buffer)*values.FieldDefine[i] + "=" + values.FieldDefine[i]->toSqlStr();
        }
    }

    Buffer Where = "";
    for (size_t i = 0; i < values.FieldDefine.size(); i++) {
        if (values.FieldDefine[i]->Condition & SQL_CONDITION) {
            if (!isfirst)Where += " AND ";
            else isfirst = false;
            Where += (Buffer)*values.FieldDefine[i] + "=" + values.FieldDefine[i]->toSqlStr();
        }
    }
    if (Where.size() > 0)
        sql += " WHERE " + Where;
    sql += " ;";
    printf("sql = %s\n", (char*)sql);
    return sql;
}

Buffer _mysql_table_::Query()
{
    Buffer sql = "SELECT ";
    for (size_t i = 0; i < FieldDefine.size(); i++)
    {
        if (i > 0)sql += ',';
        sql += '`' + FieldDefine[i]->Name + "` ";
    }
    sql += " FROM " + (Buffer)*this + ";";
    printf("sql = %s\n", (char*)sql);
    return sql;
}

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

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

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

_mysql_field_::_mysql_field_() :_Field_()
{
    nType = TYPE_NULL;
    Value.Double = 0.0;
}

_mysql_field_::_mysql_field_(int ntype, const Buffer& name, unsigned attr, const Buffer& type, const Buffer& size, const Buffer& default_, const Buffer& check)
{
    nType = ntype;
    switch (ntype)
    {
    case TYPE_VARCHAR:
    case TYPE_TEXT:
    case TYPE_BLOB:
        Value.String = new Buffer();
        break;
    }

    Name = name;
    Attr = attr;
    Type = type;
    Size = size;
    Default = default_;
    Check = check;
}

_mysql_field_::_mysql_field_(const
_mysql_field_& field)
{
    nType = field.nType;
    switch (field.nType)
    {
    case TYPE_VARCHAR:
    case TYPE_TEXT:
    case TYPE_BLOB:
        Value.String = new Buffer();
        *Value.String = *field.Value.String;
        break;
    }

    Name = field.Name;
    Attr = field.Attr;
    Type = field.Type;
    Size = field.Size;
    Default = field.Default;
    Check = field.Check;
}
```

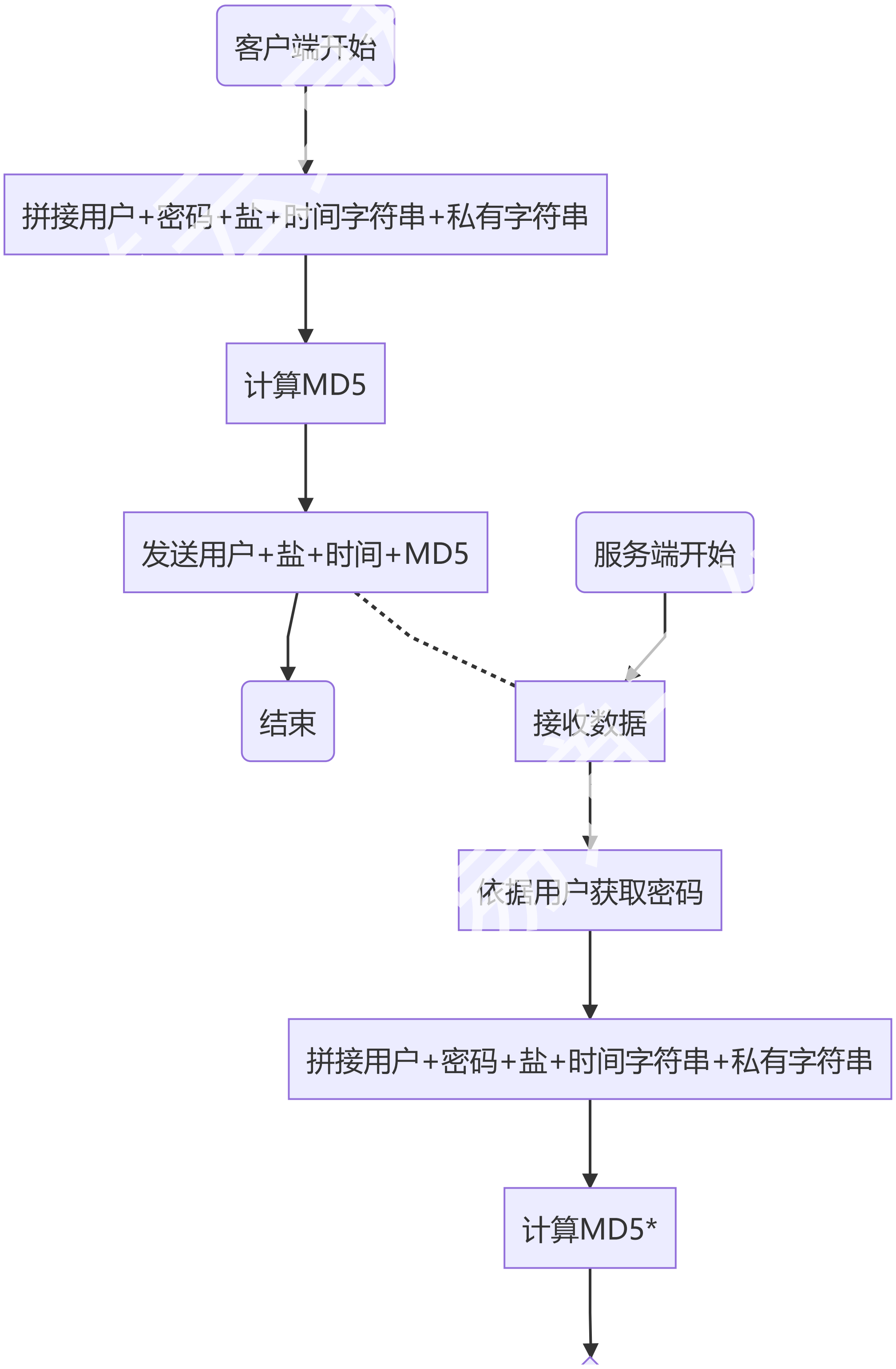
```cpp

_mysql_field_::~_mysql_field_()
{
    switch (nType)
    {
    case TYPE_VARCHAR:
    case TYPE_TEXT:
    case TYPE_BLOB:
        if (Value.String) {
            Buffer* p = Value.String;
            Value.String = NULL;
            delete p;
        }
        break;
    }
}

Buffer _mysql_field_::Create()
{
    Buffer sql = "`" + Name + "` " + Type + Size + " ";
    if (Attr & NOT_NULL) {
        sql += "NOT NULL";
    }
    else {
        sql += "NULL";
    }
    //BLOB TEXT GEOMETRY JSON不能有默认值的
    if ((Attr & DEFAULT) && (Default.size() > 0)&&(Type != "BLOB") && (Type != "TEXT") && (Type != "GEOMETRY") && (Type != "JSON"))
    {
        sql += " DEFAULT \"" + Default + "\" ";
    }
    //UNIQUE PRIMARY_KEY 外面处理
    //CHECK mysql不支持
    if (Attr & AUTOINCREMENT) {
        sql += " AUTO_INCREMENT ";
    }
    return sql;
}

void _mysql_field_::LoadFromStr(const Buffer& str)
{
    switch (nType)
    {
    case TYPE_NULL:
        break;
    case TYPE_BOOL:
    case TYPE_INT:
    case TYPE_DATETIME:
        Value.Integer = atoi(str);
        break;
    case TYPE_REAL:
        Value.Double = atof(str);
        break;
    case TYPE_VARCHAR:
    case TYPE_TEXT:
        *Value.String = str;
        break;
    case TYPE_BLOB:
        *Value.String = Str2Hex(str);
        break;
    default:
        printf("type=%d\n", nType);
        break;
    }
}

Buffer _mysql_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:
        printf("type=%d\n", nType);
        break;
    }
    return sql;
}

Buffer _mysql_field_::toSqlStr() const
{
    Buffer sql = "";
    std::stringstream ss;
    switch (nType)
    {
    case TYPE_NULL:
        sql += " NULL ";
        break;
    case TYPE_BOOL:
```

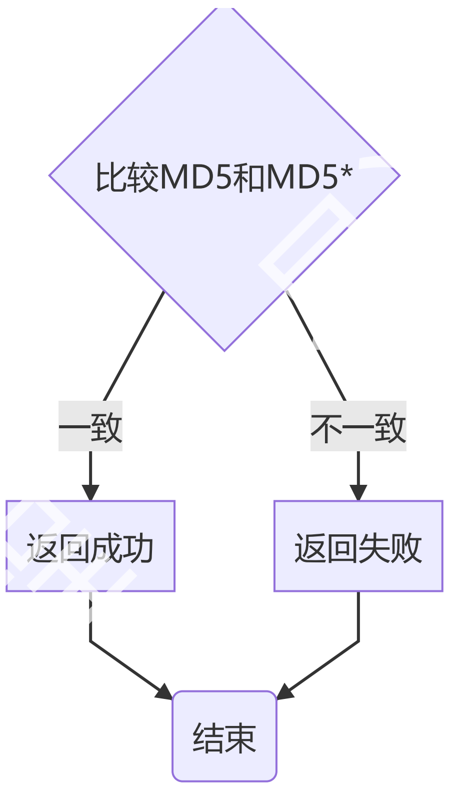
```
    case TYPE_INT:
    case TYPE_DATETIME:
        ss << Value.Integer;
        sql += ss.str() + " ";
        break;
    case TYPE_REAL:
        ss << Value.Double;
        sql += ss.str() + " ";
        break;
    case TYPE_VARCHAR:
    case TYPE_TEXT:
    case TYPE_BLOB:
        sql += '"' + *Value.String + "\" ";
        break;
    default:
        printf("type=%d\n", nType);
        break;
    }
    return sql;
}

_mysql_field_::operator const Buffer() const
{
    return '`' + Name + '`';
}

Buffer _mysql_field_::Str2Hex(const Buffer& data) const
{
    const char* hex = "0123456789ABCDEF";
    std::stringstream ss;
    for (auto ch : data)
        ss << hex[(unsigned char)ch >> 4] << hex[(unsigned char)ch & 0xF];
    return ss.str();
}

```

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

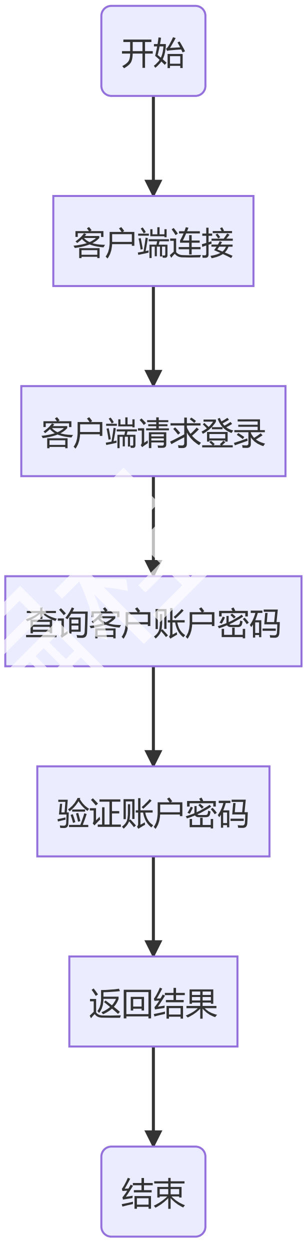
```cpp
class COpenSSLHelper
{
public:
    static Buffer MD5(const Buffer& text);
};

```

OpenSSLHelper.cpp

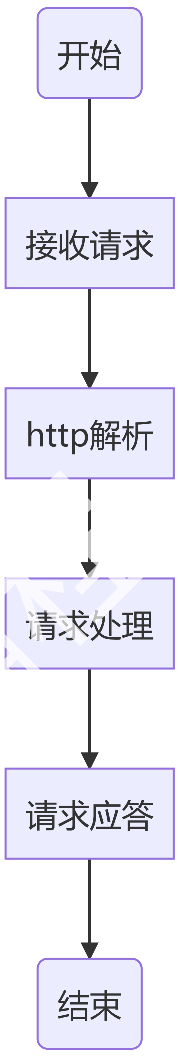
```cpp
#include "OpenSSLHelper.h"
#include "openssl/md5.h"

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_)
```

```
DECLARE_MYSQL_FIELD(TYPE_INT, user_id, NOT_NULL | PRIMARY_KEY | AUTOINCREMENT, "INTEGER", "", "", "")
DECLARE_MYSQL_FIELD(TYPE_VARCHAR, user_qq, NOT_NULL, "VARCHAR", "(15)", "", "")  //QQ号
DECLARE_MYSQL_FIELD(TYPE_VARCHAR, user_phone, DEFAULT, "VARCHAR", "(11)", "'18888888888'", "")  //手机
DECLARE_MYSQL_FIELD(TYPE_TEXT, user_name, NOT_NULL, "TEXT", "", "", "")    //姓名
DECLARE_MYSQL_FIELD(TYPE_TEXT, user_nick, NOT_NULL, "TEXT", "", "", "")    //昵称
DECLARE_MYSQL_FIELD(TYPE_TEXT, user_wechat, DEFAULT, "TEXT", "", "NULL", "")
DECLARE_MYSQL_FIELD(TYPE_TEXT, user_wechat_id, DEFAULT, "TEXT", "", "NULL", "")
DECLARE_MYSQL_FIELD(TYPE_TEXT, user_address, DEFAULT, "TEXT", "", "", "")
DECLARE_MYSQL_FIELD(TYPE_TEXT, user_province, DEFAULT, "TEXT", "", "", "")
DECLARE_MYSQL_FIELD(TYPE_TEXT, user_country, DEFAULT, "TEXT", "", "", "")
DECLARE_MYSQL_FIELD(TYPE_INT, user_age, DEFAULT | CHECK, "INTEGER", "", "18", "")
DECLARE_MYSQL_FIELD(TYPE_INT, user_male, DEFAULT, "BOOL", "", "1", "")
DECLARE_MYSQL_FIELD(TYPE_TEXT, user_flags, DEFAULT, "TEXT", "", "0", "")
DECLARE_MYSQL_FIELD(TYPE_REAL, user_experience, DEFAULT, "REAL", "", "0.0", "")
DECLARE_MYSQL_FIELD(TYPE_INT, user_level, DEFAULT | CHECK, "INTEGER", "", "0", "")
DECLARE_MYSQL_FIELD(TYPE_TEXT, user_class_priority, DEFAULT, "TEXT", "", "", "")
DECLARE_MYSQL_FIELD(TYPE_REAL, user_time_per_viewer, DEFAULT, "REAL", "", "", "")
DECLARE_MYSQL_FIELD(TYPE_TEXT, user_career, NONE, "TEXT", "", "", "")
DECLARE_MYSQL_FIELD(TYPE_TEXT, user_password, NOT_NULL, "TEXT", "", "", "")
DECLARE_MYSQL_FIELD(TYPE_INT, user_birthday, NONE, "DATETIME", "", "", "")
DECLARE_MYSQL_FIELD(TYPE_TEXT, user_describe, NONE, "TEXT", "", "", "")
DECLARE_MYSQL_FIELD(TYPE_TEXT, user_education, NONE, "TEXT", "", "", "")
DECLARE_MYSQL_FIELD(TYPE_INT, user_register_time, DEFAULT, "DATETIME", "", "LOCALTIME()", "")
DECLARE_TABLE_CLASS_EDN()

/*
* 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() {
        if (m_db) {
            CDatabaseClient* db = m_db;
            m_db = NULL;
            db->Close();
            delete db;
        }
        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)
{
        using namespace std::placeholders;
        int ret = 0;
        m_db = new CMysqlClient();
        if (m_db == NULL) {
            TRACEE("no more memory!");
            return -1;
        }
        KeyValue args;
        args["host"] = "192.168.1.100";
        args["user"] = "root";
        args["password"] = "123456";
        args["port"] = 3306;
        args["db"] = "edoyun";
        ret = m_db->Connect(args);
        ERR_RETURN(ret, -2);
        edoyunLogin_user_mysql user;
        ret = m_db->Exec(user.Create());
        ERR_RETURN(ret, -3);
        ret = setConnectedCallback(&CEdoyunPlayerServer::Connected, this, _1);
        ERR_RETURN(ret, -4);
        ret = setRecvCallback(&CEdoyunPlayerServer::Received, this, _1, _2);
        ERR_RETURN(ret, -5);
        ret = m_epoll.Create(m_count);
        ERR_RETURN(ret, -6);
        ret = m_pool.Start(m_count);
        ERR_RETURN(ret, -7);
        for (unsigned i = 0; i < m_count; i++) {
            ret = m_pool.AddTask(&CEdoyunPlayerServer::ThreadFunc, this);
            ERR_RETURN(ret, -8);
        }
        int sock = 0;
        sockaddr_in addrin;
        while (m_epoll != -1) {
            ret = proc->RecvSocket(sock, &addrin);
            TRACEI("RecvSocket ret=%d", ret);
            if (ret < 0 || (sock == 0))break;
            CSocketBase* pClient = new CSocket(sock);
            if (pClient == NULL)continue;
            ret = pClient->Init(CSockParam(&addrin, SOCK_ISIP));
            WARN_CONTINUE(ret);
            ret = m_epoll.Add(sock, EpollData((void*)pClient));
            if (m_connectedcallback) {
                (*m_connectedcallback)(pClient);
            }
            WARN_CONTINUE(ret);
        }
        return 0;
    }
private:
    int Connected(CSocketBase* pClient) {
        //TODO:客户端连接处理 简单打印一下客户端信息
        sockaddr_in* paddr = *pClient;
        TRACEI("client connected addr %s port:%d", inet_ntoa(paddr->sin_addr), paddr->sin_port);
        return 0;
    }

    int Received(CSocketBase* pClient, const Buffer& data) {
        TRACEI("接收到数据！");
        //TODO:主要业务，在此处理
        //HTTP 解析
        int ret = 0;
        Buffer response = "";
        ret = HttpParser(data);
        TRACEI("HttpParser ret=%d", ret);
        //验证结果的反馈
        if (ret != 0) {//验证失败
            TRACEE("http parser failed!%d", ret);
        }
        response = MakeResponse(ret);
        ret = pClient->Send(response);
        if (ret != 0) {
            TRACEE("http response failed!%d [%s]", ret, (char*)response);
        }
        else {
            TRACEI("http response success!%d", ret);
        }
        return 0;
    }
    int HttpParser(const Buffer& data) {
        CHttpParser parser;
        size_t size = parser.Parser(data);
        if (size == 0 || (parser.Errno() != 0)) {
            TRACEE("size %llu errno:%u", size, parser.Errno());
            return -1;
        }
        if (parser.Method() == HTTP_GET) {
            //get 处理
            UrlParser url("https://192.168.1.100" + parser.Url());
            int ret = url.Parser();
            if (ret != 0) {
                TRACEE("ret = %d url[%s]", ret, "https://192.168.1.100" + parser.Url());
                return -2;
            }
            Buffer uri = url.Uri();
            TRACEI("**** uri = %s", (char*)uri);
            if (uri == "login") {
                //处理登录
                Buffer time = url["time"];
                Buffer salt = url["salt"];
                Buffer user = url["user"];
                Buffer sign = url["sign"];
                TRACEI("time %s salt %s user %s sign %s", (char*)time, (char*)salt, (char*)user, (char*)sign);
                //数据库的查询
                edoyunLogin_user_mysql dbuser;
                Result result;
                Buffer sql = dbuser.Query("user_name=\"" + user + "\"");
                ret = m_db->Exec(sql, result, dbuser);
                if (ret != 0) {
                    TRACEE("sql=%s ret=%d", (char*)sql, ret);
                    return -3;
                }
                if (result.size() == 0) {
                    TRACEE("no result sql=%s ret=%d", (char*)sql, ret);
                    return -4;
                }
                if (result.size() != 1) {
                    TRACEE("more than one sql=%s ret=%d", (char*)sql, ret);
                    return -5;
                }
                auto user1 = result.front();
                Buffer pwd = *user1->Fields["user_password"]->Value.String;
                TRACEI("password = %s", (char*)pwd);
                //登录请求的验证
                const char* MD5_KEY = "*&^%$#@b.v+h-b*g/h@n!h#n$d^ssx,.kl<kl";
                Buffer md5str = time + MD5_KEY + pwd + salt;
                Buffer md5 = Crypto::MD5(md5str);
                TRACEI("md5 = %s", (char*)md5);
                if (md5 == sign) {
                    return 0;
                }
                return -6;
            }
```

```
        }
        else if (parser.Method() == HTTP_POST) {
            //post 处理
        }
        return -7;
    }
    Buffer MakeResponse(int ret) {
        Json::Value root;
        root["status"] = ret;
        if (ret != 0) {
            root["message"] = "登录失败，可能是用户名或者密码错误！";
        }
        else {
            root["message"] = "success";
        }
        Buffer json = root.toStyledString();
        Buffer result = "HTTP/1.1 200 OK\r\n";
        time_t t;
        time(&t);
        tm* ptm = localtime(&t);
        char temp[64] = "";
        strftime(temp, sizeof(temp), "%a, %d %b %G %T GMT\r\n", ptm);
        Buffer Date = Buffer("Date: ") + temp;
        Buffer Server = "Server: Edoyun/1.0\r\nContent-Type: text/html; charset=utf-8\r\nX-Frame-Options: DENY\r\n";
        snprintf(temp, sizeof(temp), "%d", json.size());
        Buffer Length = Buffer("Content-Length: ") + temp + "\r\n";
        Buffer Stub = "X-Content-Type-Options: nosniff\r\nReferrer-Policy: same-origin\r\n\r\n";
        result += Date + Server + Length + Stub + json;
        TRACEI("response: %s", (char*)result);
        return result;
    }
private:
    int ThreadFunc() {
        int ret = 0;
        EPEvents events;
        while (m_epoll != -1) {
            ssize_t size = m_epoll.WaitEvents(events);
            if (size < 0)break;
            if (size > 0) {
                for (ssize_t i = 0; i < size; i++)
                {
                    if (events[i].events & EPOLLERR) {
                        break;
                    }
                    else if (events[i].events & EPOLLIN) {
                        CSocketBase* pClient = (CSocketBase*)events[i].data.ptr;
                        if (pClient) {
                            Buffer data;
                            ret = pClient->Recv(data);
                            TRACEI("recv data size %d", ret);
                            if (ret <= 0) {
                                TRACEW("ret= %d errno = %d msg = [%s]", ret, errno, strerror(errno));
                                m_epoll.Del(*pClient);
                                continue;
                            }
                            if (m_recvcallback) {
                                (*m_recvcallback)(pClient, data);
                            }
                        }
                    }
                }
            }
        }
        return 0;
    }
private:
    CEpoll m_epoll;
    std::map<int, CSocketBase*> m_mapClients;
    CThreadPool m_pool;
    unsigned m_count;
    CDatabaseClient* m_db;
};
```

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。