零声教育出品 Mark 老师 QQ：2548898954

linux 开发调试环境安装

一、依赖安装

> https://trinitycore.info/en/install/requirements/linux

```
sudo apt-get update
sudo apt-get install git clang cmake make gcc g++ libmysqlclient-dev libssl-dev libbz2-dev libreadline-dev libncurses-dev libboost-all-dev mysql-server-5.7 p7zip

sudo update-alternatives --install /usr/bin/cc cc /usr/bin/clang 100
sudo update-alternatives --install /usr/bin/c++ c++ /usr/bin/clang 100
```

二、服务端编译

1. **源码下载**

```
mkdir game
cd game
git clone -b 3.3.5 https://github.com/TrinityCore/TrinityCore.git
```

2. **源码编译**

```
mkdir build
cd build
cmake ../ -DCMAKE_INSTALL_PREFIX=/home/mark/game -DCONF_DIR=/home/mark/game/bin
make -j2
make install
```

三、生成数据信息

1. 地图信息

```
cd ~
mkdir res
cd res

#dbc maps 地图数据
../game/bin/mapextractor

#vmaps 建筑物、山脉、水体     角色，怪物、npc
mkdir vmaps
../game/bin/vmap4extractor

../game/bin/vmap4assembler Buildings vmaps

# 地图移动数据
mkdir mmaps
../game/bin/mmaps_generator

```

2. 数据库信息

```
mysql -uroot -p123456

# auth,charactors,world
source /home/mark/game/TrinityCore/sql/create/create_mysql.sql;
```

四、启动服务端

authserver

> 第一次启动将会失败，没关系后面 worldserver 启动会加载数据库信息，之后能启动成功。

worldserver

> - 修改 worldserver.conf
>
>   ```
>   DataDir = "../../res"
>   ```
>
> - 下载 TDB_full_world_335.23011_2023_01_16.sql
>
>   点击选择 TDB 335.23011
>
>   拷贝到 `~/game/bin` 下
>
> - 修改 auth 库
>
>   ```
>   mysql> use auth;
>   # xx.xx.xx.xx 替换你服务端的 ip 地址
>   mysql> update realmlist set address = "xx.xx.xx.xx" where id = 1;
>   ```
>
> - 启动服务器

```
# 开启一个会话
./worldserver

# 开启另一个会话
./authserver
```

- 测试客户端连接

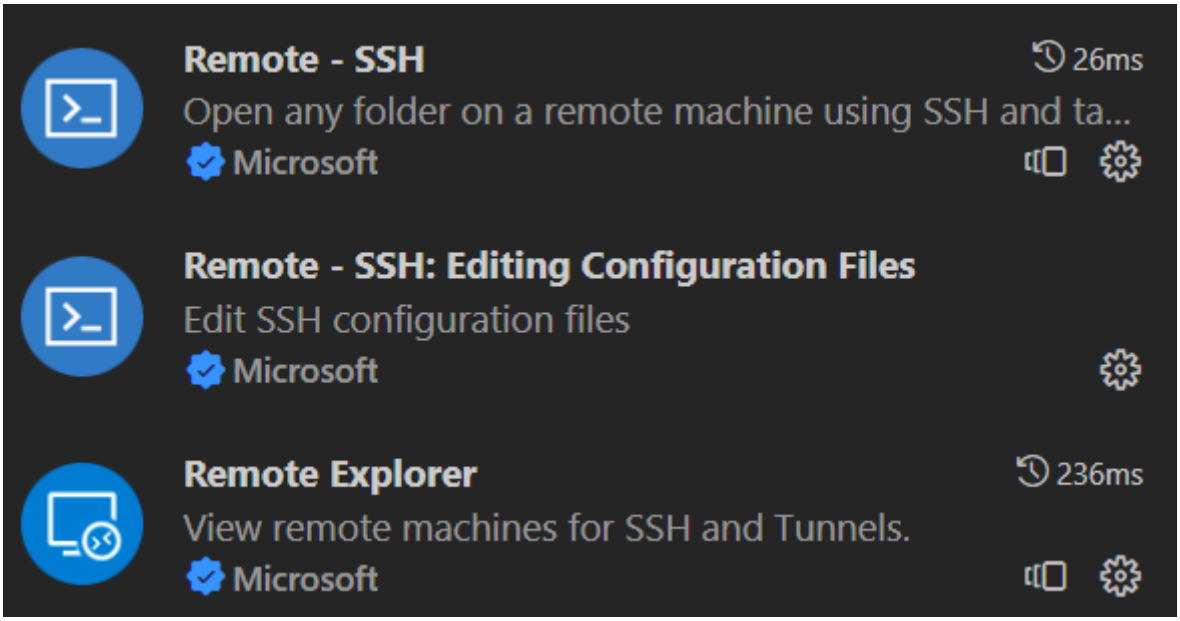
  使用客户端连接目标服务器

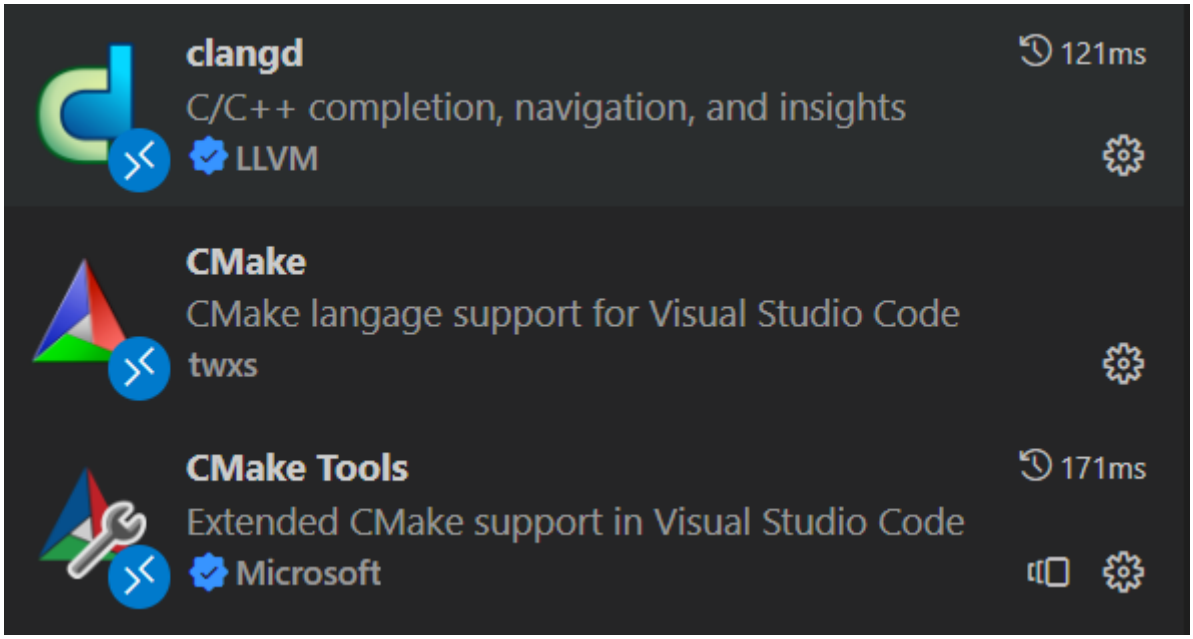
五、调试环境

vscode 安装

> https://code.visualstudio.com/

vscode 插件安装





在根目录下创建 .vscode

> 创建 settings.json

```
{
    "cmake.buildDirectory": "${workspaceFolder}/build",
    "cmake.buildEnvironment": {"CMAKE_EXPORT_COMPILE_COMMANDS": "ON"},
    "clangd.arguments": [
        "--background-index",
        "--compile-commands-dir=${workspaceFolder}/build"
    ],
    "clangd.fallbackFlags": [
        "-I${workspaceFolder}/src/common/",
        "-I${workspaceFolder}/src/common/Utilities/"
    ],
    "clangd.checkUpdates": true,
    "clangd.path": "/home/mark/clangd_17.0.3/bin/clangd"
}
```

六、客户端下载

```
https://www.wowdl.net/client/World-of-Warcraft-3.3.5a.12340-zhCN

或者：

https://pan.baidu.com/s/13PmQlCnCnfBPcFlAU4YGeg
提取码：zvtc
```

修改启动

> 下载客户端，解压后，在根目录创建一个 `魔兽登录器.bat`；添加如下内容：

```
#if not exist "WTF" md "WTF"
 echo set realmlist 127.0.0.1>realmlist.WTF
 echo set realmlist 127.0.0.1>data/enGB/realmlist.WTF
 echo set realmlist 127.0.0.1>data/zhcn/realmlist.WTF
 start wow.exe
 goto end
```

window 开发调试环境安装

环境要求

- Boost >= 1.73
- MySQL >= 5.7
- OpenSSL = 1.1.x
- CMake >= 3.18.4
- visual studio (Community) 2019

环境配置

- boost

  > https://sourceforge.net/projects/boost/files/boost-binaries/ 选择下载文件名boost_1_80_0-msvc-14.2-64
  >
  > 1_80_0 对应boost版本, msvc-14.2 对应支持 Visual Studio 版本, 64 对应64位, Visual Studio 版本对应编号 2015 = v140，2017 = v141，2019 = v142，2022 = v143
  >
  > 添加环境变量BOOST_ROOT到系统变量，值为Boost安装路径。
  >
  > 运行安装程序，选择custom模式,这样才可以选择安装路径。

- MySQL

  > 下载安装msi：https://dev.mysql.com/downloads/installer/

- OpenSSL

  > https://slproweb.com/products/Win32OpenSSL.html (查找最新1.1.x64位版本，不是“轻量级”或3.x版本。NOT the "light" or 3.x version)

- CMake

  > 下载安装：https://github.com/Kitware/CMake/releases (推荐安装最新版本 windows-x86_64.msi)

- MS Visual Studio

  > 下载安装：https://visualstudio.microsoft.com/zh-hans/

> 安装C++编译器，详细内容见官方文档 https://www.trinitycore.info/install/requirements/windows

- git

> 下载安装：https://git-scm.com/downloads

编译 TrinityCore

- 下载源码

```
git clone https://github.com/TrinityCore/TrinityCore.git -b 3.3.5
```

- CMake 配置生成项目文件
  - 源码同级目录创建 build 目录
  - 运行 CMake GUI
  - 浏览源 -> 选择源目录
  - 浏览构建 -> 选择构建目录（选择刚创建的 build 目录）
  - 单击配置
  - 生成（在 build 目录中可以看到 TrinityCore.sln 文件

其他步骤和 linux 一致

SRP-6 协议

> Secure Remote Password (安全远程密码)。使用 SRP 协议的双端可以在不传送明文密码的情况下安全验证。通过这种做法可以避免密码在传输过程中被劫持。
>
> 协议描述：http://srp.stanford.edu/design.html

名词解释

```
N - A large safe prime (N = 2q+1, where q is prime) All arithmetic is done modulo N.

g - A generator modulo(模数) N

k - Multiplier parameter(乘数参数) (k = H(N, g) in SRP-6a, k = 3 for legacy SRP-6)

s - User's salt(盐，用于混淆)

I - Username

p - Cleartext Password

H() - One-way hash function(单向散列函数)

^ - (Modular) Exponentiation(指数)

u - Random scrambling parameter(随机加扰参数)

a,b - Secret ephemeral values(临时随机值)

A,B - Public ephemeral values(临时公钥)

x - Private key (derived from p and s)(私钥)

v - Password verifier
```

TrinityCore 中实现：

```
N: 一个大的质数(N=2q+1)
  HexStrToByteArray<32>("894B645E89E1535BBDAD5B8B290650530801B18EBFBF5E8FAB3C82872A3E9BB7", true);
```

```
s: 32个字节的随机数  salt
g: 7
v: g^sha1(salt, sha1(username, ":", password)) mod N
b: 19个字节的随机数(服务端随机生成)
a: 19个字节的随机数(客户端随机生成)
B: 公钥 B = ((v*3) + g^b) mod N
K: sessionKey
  1) 客户端公钥 A = g^a mod N
  2) x = sha1(salt, sha1(username, ":", password))
  3) u = sha1(A,B)
  4) S = (B-g^x*3)^(a+u*x) 32 字节
  5) K 基于 S 的奇数部分和偶数部分分别进行hash，然后奇偶交错组合两个 hash 后的值
M: 20个字节的数 M = sha1(t3, t4, s, A, B, K)
  1) t3 = sha1(N)[i] ^ sha1(g)[i]
  2) t4 = sha1(username)
k: 3
```

SRP-6 交互流程

```
   C                                   S

username        →             生成 N,g,s

                ←             N,g,s

随机 a，计算 A(g^a mod N)

A               →             随机 b，计算 B <v+g^b mod N> 生成 u <H(A,B)>

                ←             B

计算 S <(B - g^x) ^ (a + ux)>
计算 K <H(S)>
计算 M1 = <H(A,B,K)>

M1              →             计算 S <(A-v^u)^b>
                              计算 K <H(S)>
                              计算 M1 = <H(A,B,K)>
                              如果不相同，密码错误
                              如果相同 M2 = H(A,M1,K)
                ←             M2

按照服务端相同的算法计算M2
并验证 M2
```

TrinityCore 中 SRP-6 交互流程

```
username                →             g,N,s from DB，生成 B
                                      随机 b
                        ←             g,N,s,B
随机 a，生成 A,M1

A,M1                    →
                                      验证 M1，生成 sessionK，M2
                        ←             M2
验证 M2
```

实现 web 注册服

环境安装

```
tar -xzvf openresty-VERSION.tar.gz
cd openresty-VERSION/
./configure \
   --with-cc-opt="-I/usr/local/opt/openssl/include/ -I/usr/local/opt/pcre/include/" \
   --with-ld-opt="-L/usr/local/opt/openssl/lib/ -L/usr/local/opt/pcre/lib/" \
   -j8
make
sudo make install

sudo opm get fffonion/lua-resty-openssl
```

修改 openssl 库

一、找到 c 函数声明的位置

```
sudo vi /usr/local/openresty/lualib/resty/openssl/include/bn.lua
```

二、在 include/bn.lua 中修改

```
ffi.cdef(
[[
-- ...
  BIGNUM *BN_lebin2bn(const unsigned char *s, int len, BIGNUM *ret);
  int BN_bn2lebinpad(const BIGNUM *a, unsigned char *to, int tolen);
-- ...
]])
```

三、找到具体实现，在 bn.lua（跟上面路径不一样）中修改

> `sudo vi /usr/local/openresty/lualib/resty/openssl/bn.lua`
>
> 主要修改了 77 行和 91 行。

```
 54 function _M:to_binary(pad)
 55   if pad then
 56     if type(pad) ~= "number" then
 57       return nil, "bn:to_binary: expect a number at #1"
 58     elseif OPENSSL_10 then
 59       return nil, "bn:to_binary: padding is only supported on OpenSSL 1.1.0 or later"
 60     end
 61   end
 62
 63   local length
 64   if not pad then
 65     length = (C.BN_num_bits(self.ctx)+7)/8
 66     -- align to bytes
 67     length = floor(length)
 68   else
 69     length = pad
 70   end
 71
 72   local buf = ctypes.uchar_array(length)
 73   local sz
 74   if not pad then
 75     sz = C.BN_bn2bin(self.ctx, buf)
 76   else
 77     sz = C.BN_bn2lebinpad(self.ctx, buf, pad)
 78   end
 79
 80   if sz <= 0 then
```

```
 81     return nil, format_error("bn:to_binary")
 82   end
 83   return ffi_str(buf, sz)
 84 end


 86 function _M.from_binary(s)
 87   if type(s) ~= "string" then
 88     return nil, "bn.from_binary: expect a string at #1"
 89   end
 90
 91   local ctx = C.BN_lebin2bn(s, #s, nil)
 92   if ctx == nil then
 93     return nil, format_error("bn.from_binary")
 94   end
 95   ffi_gc(ctx, C.BN_free)
 96   return setmetatable( { ctx = ctx }, mt), nil
 97 end
```