C++ - 在线五子棋对战

代码链接：https://gitee.com/qigezi/online_gobang.git

1. 项目介绍

本项目主要实现一个网页版的五子棋对战游戏， 其主要支持以下核心功能:

- 用户管理: 实现用户注册， 用户登录、获取用户信息、用户天梯分数记录、用户比赛场次记录等
- 匹配对战: 实现两个玩家在网页端根据天梯分数匹配游戏对手，并进行五子棋游戏对战的功能
- 聊天功能: 实现两个玩家在下棋的同时可以进行实时聊天的功能

2. 开发环境

- Linux（Centos-7.6 / Ubuntu-22.04）
- VSCode/Vim
- g++/gdb
- Makefile

3. 核心技术

- HTTP/WebSocket
- Websocket++
- JsonCpp
- Mysql
- C++11
- BlockQueue
- HTML/CSS/JS/AJAX

4. 环境搭建

4.1 Ubuntu-22.04环境搭建

4.1.1 更换软件源

```
1 dev@bite:~$ sudo cp /etc/apt/sources.list.d/original.list
  /etc/apt/sources.list.d/original.list.bak
```

```
dev@bite:~$ sudo vim /etc/apt/sources.list.d/original.list
```

在底行模式下，进行字符串替换，将文档中的 'cn.archive.ubuntu.com' 替换为 'mirrors.aliyun.com'，替换方式为示例第18行的写法。

```
1  # See http://help.ubuntu.com/community/UpgradeNotes for how to upgrade to
2  # newer versions of the distribution.
3  deb http://cn.archive.ubuntu.com/ubuntu jammy main restricted
4  # deb-src http://cn.archive.ubuntu.com/ubuntu jammy main restricted
5
6  ## Major bug fix updates produced after the final release of the
7  ## distribution.
8  deb http://cn.archive.ubuntu.com/ubuntu jammy-updates main restricted
9  # deb-src http://cn.archive.ubuntu.com/ubuntu jammy-updates main restricted
10
11 ## N.B. software from this repository is ENTIRELY UNSUPPORTED by the Ubuntu
12 ## team. Also, please note that software in universe WILL NOT receive any
13 ## review or updates from the Ubuntu security team.
14 deb http://cn.archive.ubuntu.com/ubuntu jammy universe
15 # deb-src http://cn.archive.ubuntu.com/ubuntu jammy universe
16 deb http://cn.archive.ubuntu.com/ubuntu jammy-updates universe
17 # deb-src http://cn.archive.ubuntu.com/ubuntu jammy-updates universe
18 :%s/cn.archive.ubuntu.com/mirrors.aliyun.com/g
```

更改完毕后，更新apt的source list

```
dev@bite:~$ sudo apt update
```

4.1.2 安装lrzsz传输工具

```
dev@bite:~$ sudo apt-get install lrzsz
...
dev@bite:~$ rz --version
rz (GNU lrzsz) 0.12.21rc
dev@bite:~$ sz --version
sz (lrzsz) 0.12.21rc
```

安装完毕后使用 rz --version 查看工具版本，正常显示则表示安装成功。

4.1.3 安装gcc/g++编译器

```
dev@bite:~$ sudo apt-get install gcc g++
...
dev@bite:~$ gcc --version
gcc (Ubuntu 11.3.0-1ubuntu1~22.04) 11.3.0
Copyright (C) 2021 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

dev@bite:~$ g++ --version
g++ (Ubuntu 11.3.0-1ubuntu1~22.04) 11.3.0
Copyright (C) 2021 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

dev@bite:~$
```

安装完毕后，查看版本，正常显示则安装成功。

4.1.4 安装gdb调试器

```
dev@bite:~$ sudo apt-get install gdb
...
dev@bite:~$ gdb --version
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 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.
dev@bite:~$
```

安装完毕后，查看版本，正常显示则安装成功。

4.1.5 安装git工具

```
dev@bite:~$ sudo apt-get install git
...
dev@bite:~$ git --version
git version 1.8.3.1
```

4.1.6 安装cmake项目构建工具

```
dev@bite:~$ sudo apt-get install cmake
...
dev@bite:~$ cmake --version
cmake version 3.22.1

CMake suite maintained and supported by Kitware (kitware.com/cmake).
dev@bite:~$
```

安装完毕后，查看版本，正常显示则安装成功。

4.1.7 安装boost库

```
dev@bite:~$ sudo apt-get install libboost-all-dev
...
dev@bite:~$ dpkg -S /usr/include/boost/version.hpp
libboost1.74-dev:amd64: /usr/include/boost/version.hpp
dev@bite:~$
```

安装完毕后，查看版本，正常显示则安装成功。其实只要 '/usr/include/' 下有 'boost' 目录，且其中有头文件就表示安装成功了。

4.1.8 安装jsoncpp库

```
dev@bite:~$ sudo apt-get install libjsoncpp-dev
...
dev@bite:~$ ls /usr/include/jsoncpp/json/
allocator.h  assertions.h  config.h  forwards.h  json_features.h  json.h
reader.h  value.h  version.h  writer.h

dev@bite:~$ dpkg -S /usr/include/jsoncpp/json/version.h
libjsoncpp-dev:amd64: /usr/include/jsoncpp/json/version.h

dev@bite:~$ ls /usr/lib/x86_64-linux-gnu/libjsoncpp*
/usr/lib/x86_64-linux-gnu/libjsoncpp.so
```

查看 '/usr/include' 下有 'jsoncpp' 目录，且其中包含有头文件, '/usr/lib/x86_64-linux-gnu/'下有对应的库文件就表示成功了。

4.1.9 安装mysql数据库

4.1.9.1 获取mysql-5.7安装源

```
dev@bite:~$ wget http://repo.mysql.com/mysql-apt-config_0.8.12-1_all.deb
...
dev@bite:~$
```

4.1.9.2 安装mysql官方源：

```
dev@bite:~$ sudo dpkg -i mysql-apt-config_0.8.12-1_all.deb
...
#1. 过程中出现apt源的安装选择，则选择 bionic 然后 OK
#2. 过程中出现mysql源版本选择时，选择mysql-5.7，然后 OK
...

```

4.1.9.3 更新apt源

```
dev@bite:~$ sudo apt-get update
dev@bite:~$
```

4.1.9.4 出现错误：E: The repository 'file:/cdrom jammy Release' no longer has a Release file

```
dev@bite:~$ sudo vi /etc/apt/sources.list
deb [check-date=no] file:///cdrom jammy main restricted

删除这行内容
```

4.1.9.5 出现gpg-key过期的情况解决方案：

```
dev@bite:~$ sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys
467B942D3A79BD29
```

4.1.9.6 出现更新警告：Key is stored in legacy trusted.gpg keyring (/etc/apt/trusted.gpg)

这个警告的意思是，需要在 trusted.gpg.d/ 目录下查找GPG密钥，因此将密钥文件拷贝过去即可。

```
dev@bite:~$ sudo cp /etc/apt/trusted.gpg /etc/apt/trusted.gpg.d/
dev@bite:~$ sudo apt-get update
```

4.1.9.7 查看当前可安装mysql版本：

```
dev@bite:~$ sudo apt-cache policy mysql-server
mysql-server:
  Installed: (none)
  Candidate: 8.0.33-0ubuntu0.22.04.2
  Version table:
     8.0.33-0ubuntu0.22.04.2 500
        500 http://cn.archive.ubuntu.com/ubuntu jammy-updates/main amd64
Packages
        500 http://cn.archive.ubuntu.com/ubuntu jammy-security/main amd64
Packages
     8.0.28-0ubuntu4 500
        500 http://cn.archive.ubuntu.com/ubuntu jammy/main amd64 Packages
     5.7.42-1ubuntu18.04 500
        500 http://repo.mysql.com/apt/ubuntu bionic/mysql-5.7 amd64 Packages
dev@bite:~$
```

4.1.9.8 安装5.7版本mysql服务及开发包：

```
dev@bite:~$ sudo apt install -f mysql-client=5.7* mysql-community-server=5.7*
mysql-server=5.7* libmysqlclient-dev=5.7*
```

4.1.9.9 修改配置文件：/etc/mysql/my.cnf

```
 The MariaDB configuration file
#
# The MariaDB/MySQL tools read configuration files in the following order:
# 0. "/etc/mysql/my.cnf" symlinks to this file, reason why all the rest is
read.
# 1. "/etc/mysql/mariadb.cnf" (this file) to set global defaults,
# 2. "/etc/mysql/conf.d/*.cnf" to set global options.
# 3. "/etc/mysql/mariadb.conf.d/*.cnf" to set MariaDB-only options.
# 4. "~/.my.cnf" to set user-specific options.
#
# If the same option is defined multiple times, the last one will apply.
#
# One can use all long options that the program supports.
```

```
# Run program with --help to get a list of available options and with
# --print-defaults to see which it would actually understand and use.
#
# If you are new to MariaDB, check out https://mariadb.com/kb/en/basic-mariadb-articles/

#
# This group is read both by the client and the server
# use it for options that affect everything
#
[client]
default-character-set = utf8
[mysql]
default-character-set = utf8
[mysqld]
character-set-server = utf8
bind-address = 0.0.0.0
[client-server]
# Port or socket location where to connect
# port = 3306
socket = /run/mysqld/mysqld.sock

# Import all .cnf files from configuration directory
!includedir /etc/mysql/conf.d/
!includedir /etc/mysql/mariadb.conf.d/
```

4.1.9.10 启动mysql服务：

```
dev@bite:~$ sudo systemctl start mysql
```

4.1.9.11 查看字符集：

```
dev@bite:~$ mysql -uroot -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 5.7.42 MySQL Community Server (GPL)

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.
```

```
12
13 Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
14
15 mysql> show variables like '%chara%';
16 +--------------------------+----------------------------+
17 | Variable_name            | Value                      |
18 +--------------------------+----------------------------+
19 | character_set_client     | utf8                       |
20 | character_set_connection | utf8                       |
21 | character_set_database   | utf8                       |
22 | character_set_filesystem | binary                     |
23 | character_set_results    | utf8                       |
24 | character_set_server     | utf8                       |
25 | character_set_system     | utf8                       |
26 | character_sets_dir       | /usr/share/mysql/charsets/ |
27 +--------------------------+----------------------------+
28 8 rows in set (0.001 sec)
29
30 mysql>
```

4.1.9.12 设置/修改mysql密码，以及设置密码强度等级：

先进性安全配置：

```
1  dev@bite:~$ sudo mysql_secure_installation
2  Securing the MySQL server deployment.
3
4  Enter password for user root:
5
6  VALIDATE PASSWORD PLUGIN can be used to test passwords
7  and improve security. It checks the strength of password
8  and allows the users to set only those passwords which are
9  secure enough. Would you like to setup VALIDATE PASSWORD plugin?
10
11 Press y|Y for Yes, any other key for No: y
12
13 There are three levels of password validation policy:
14
15 LOW    Length >= 8
16 MEDIUM Length >= 8, numeric, mixed case, and special characters
17 STRONG Length >= 8, numeric, mixed case, special characters and dictionary
18
19 Please enter 0 = LOW, 1 = MEDIUM and 2 = STRONG: 1
20 Using existing password for root.
21
22 Estimated strength of the password: 25
```

```
Change the password for root ? ((Press y|Y for Yes, any other key for No) : no

 ... skipping.
By default, a MySQL installation has an anonymous user,
allowing anyone to log into MySQL without having to have
a user account created for them. This is intended only for
testing, and to make the installation go a bit smoother.
You should remove them before moving into a production
environment.

Remove anonymous users? (Press y|Y for Yes, any other key for No) : y
Success.


Normally, root should only be allowed to connect from
'localhost'. This ensures that someone cannot guess at
the root password from the network.

Disallow root login remotely? (Press y|Y for Yes, any other key for No) : y
Success.

By default, MySQL comes with a database named 'test' that
anyone can access. This is also intended only for testing,
and should be removed before moving into a production
environment.


Remove test database and access to it? (Press y|Y for Yes, any other key for No)
 - Dropping test database...
Success.

 - Removing privileges on test database...
Success.

Reloading the privilege tables will ensure that all changes
made so far will take effect immediately.

Reload privilege tables now? (Press y|Y for Yes, any other key for No) : y
Success.

All done!
```

然后进行内部密码强度等级的进一步设置与查看。

```
dev@bite:~$ mysql -uroot -p
```

```
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4
Server version: 5.7.42 MySQL Community Server (GPL)

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> set global validate_password_policy=0;
Query OK, 0 rows affected (0.00 sec)

mysql> set global validate_password_length=1;
Query OK, 0 rows affected (0.00 sec)

mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY '你的密码';
Query OK, 0 rows affected (0.00 sec)

mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.00 sec)
```

4.1.10 安装websocketpp库

```
dev@bite:~$ git clone https://github.com/zaphoyd/websocketpp.git
Cloning into 'websocketpp'...
remote: Enumerating objects: 12791, done.
remote: Counting objects: 100% (2911/2911), done.
remote: Compressing objects: 100% (393/393), done.
remote: Total 12791 (delta 2790), reused 2519 (delta 2518), pack-reused 9880
Receiving objects: 100% (12791/12791), 8.16 MiB | 8.20 MiB/s, done.
Resolving deltas: 100% (8101/8101), done.
dev@bite:~$ ls
websocketpp
dev@bite:~$ cd websocketpp/
dev@bite:~/websocketpp$ mkdir build
dev@bite:~/websocketpp$ cd build/
dev@bite:~/websocketpp/build$ cmake -DCMAKE_INSTALL_PREFIX=/usr ..
dev@bite:~/websocketpp/build$ ls
CMakeCache.txt  CMakeFiles  cmake_install.cmake  Makefile  websocketpp
dev@bite:~/websocketpp/build$ sudo make install
dev@bite:~/websocketpp/build$ ls /usr/include/websocketpp/
```

```
base64      common       connection_base.hpp   endpoint.hpp   frame.hpp
logger
random      sha1         utf8_validator.hpp    client.hpp     concurrency
connection.hpp
error.hpp   http         message_buffer        roles          transport
utilities.hpp
close.hpp   config       endpoint_base.hpp     extensions     impl
processors
server.hpp  uri.hpp      version.hpp
dev@bite:~/websocketpp/build$ cd ..
dev@bite:~/websocketpp$ ls
build  changelog.md  cmake  CMakeLists.txt  COPYING  docs  Doxyfile  examples
readme.md  roadmap.md  SConstruct  test  tutorials  websocketpp  websocketpp-
config.cmake.in
dev@bite:~/websocketpp$ cd examples/echo_server
dev@bite:~/websocketpp/examples/echo_server$ ls
CMakeLists.txt  echo_handler.hpp  echo_server.cpp  SConscript
dev@bite:~/websocketpp/examples/echo_server$ g++ -std=c++11 echo_server.cpp -o
echo_server -lpthread -lboost_system
dev@bite:~/websocketpp/examples/echo_server$ ls
CMakeLists.txt  echo_handler.hpp  echo_server  echo_server.cpp  SConscript
dev@bite:~/websocketpp/examples/echo_server$
```

查看 '/usr/include/' 下拥有了 'websocketpp' 目录，且其中包含有头文件就表示安装成功了。

或者在 'examples/echo_server/' 中对样例进行编译，编译通过就没问题了~。~

4.2 Centos-7.6环境搭建

4.2.1 安装wget工具

```
[bite@localhost ~]$ sudo yum install wget
```

4.2.2 更换软件源

```
[bite@localhost ~]$ sudo mv /etc/yum.repos.d/CentOS-Base.repo
/etc/yum.repos.d/CentOS-Base.repo.bak
[bite@localhost ~]$ sudo wget -O /etc/yum.repos.d/CentOS-Base.repo
http://mirrors.aliyun.com/repo/Centos-7.repo
[bite@localhost ~]$ sudo yum clean all
Loaded plugins: fastestmirror
Cleaning repos: base extras updates
Cleaning up list of fastest mirrors
```

```
[bite@localhost ~]$ sudo yum makecache
...
```

4.2.3 安装scl软件源

```
[bite@localhost ~]$ sudo yum install centos-release-scl-rh centos-release-scl
```

4.2.4 安装epel软件源

```
[bite@localhost ~]$ sudo yum install epel-release
```

4.2.5 安装lrzsz传输工具

```
[bite@localhost ~]$ sudo yum install lrzsz
...
[bite@localhost ~]$ rz --version
rz (lrzsz) 0.12.20
```

4.2.6 安装高版本gcc/g++编译器

```
[bite@localhost ~]$ sudo yum install devtoolset-7-gcc  devtoolset-7-gcc-c++
...
[bite@localhost ~]$ echo "source /opt/rh/devtoolset-7/enable" >> ~/.bashrc
[bite@localhost ~]$ source ~/.bashrc
[bite@localhost ~]$ g++ -v
Using built-in specs.
COLLECT_GCC=g++
COLLECT_LTO_WRAPPER=/opt/rh/devtoolset-7/root/usr/libexec/gcc/x86_64-redhat-linux/7/lto-wrapper
Target: x86_64-redhat-linux
Configured with: ../configure --enable-bootstrap --enable-languages=c,c++,fortran,lto --prefix=/opt/rh/devtoolset-7/root/usr --mandir=/opt/rh/devtoolset-7/root/usr/share/man --infodir=/opt/rh/devtoolset-7/root/usr/share/info --with-bugurl=http://bugzilla.redhat.com/bugzilla --enable-shared --enable-threads=posix --enable-checking=release --enable-multilib --with-system-zlib --enable-__cxa_atexit --disable-libunwind-exceptions --enable-gnu-unique-object --enable-linker-build-id --with-gcc-major-version-only --enable-plugin --with-linker-hash-style=gnu --enable-
```

```
initfini-array --with-default-libstdcxx-abi=gcc4-compatible --with-
isl=/builddir/build/BUILD/gcc-7.3.1-20180303/obj-x86_64-redhat-linux/isl-
install --enable-libmpx --enable-gnu-indirect-function --with-tune=generic --
with-arch_32=i686 --build=x86_64-redhat-linux
Thread model: posix
gcc version 7.3.1 20180303 (Red Hat 7.3.1-5) (GCC)
```

4.2.7 安装gdb调试器

```
[bite@localhost ~]$ sudo yum install gdb
...
[bite@localhost ~]$ gdb --version
GNU gdb (GDB) Red Hat Enterprise Linux 7.6.1-120.el7
Copyright (C) 2013 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-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
```

4.2.8 安装git

```
[bite@localhost ~]$ sudo yum install git
...
[bite@localhost ~]$ git --version
git version 1.8.3.1
```

4.2.9 安装cmake

```
[bite@localhost ~]$ sudo yum install cmake
...
[bite@localhost ~]$ cmake --version
cmake version 2.8.12.2
```

4.2.10 安装boost库

```
[bite@localhost ~]$ sudo yum install boost-devel.x86_64
```

4.2.11 安装Jsoncpp库

```
[zsc@node ~]$ sudo yum install jsoncpp-devel
...
[zsc@node ~]$ ls /usr/include/jsoncpp/json/
assertions.h  config.h    forwards.h  reader.h  version.h
autolink.h    features.h  json.h      value.h   writer.h
```

4.2.12 安装Mysql数据库服务及开发包

4.2.12.1 获取mysql官方yum源

```
[zwc@VM-8-12-centos workspace]$ wget http://repo.mysql.com/mysql57-community-release-el7-10.noarch.rpm
```

4.2.12.2 安装mysql官方yum源

```
[zwc@VM-8-12-centos workspace]$ sudo rpm -ivh mysql57-community-release-el7-10.noarch.rpm
```

4.2.12.3 安装Mysql数据库服务

```
[zwc@VM-8-12-centos workspace]$ sudo yum install -y mysql-community-server
```

4.2.12.4 出错解决

如果因为GPG KEY的过期导致安装失败

```
[zwc@VM-8-12-centos workspace]$ GPG Keys are configured as: file:///etc/pki/rpm-gpg/RPM-GPG-KEY-mysql
```

则执行以下指令，然后重新安装

```
[zwc@VM-8-12-centos workspace]$ sudo rpm --import https://repo.mysql.com/RPM-GPG-KEY-mysql-2022
```

4.2.12.5 安装Mysql开发包

```
[zwc@VM-8-12-centos workspace]$ sudo yum install -y mysql-community-devel
```

4.2.12.6 进行Mysql配置修改

1. 配置 '/etc/my.cnf' 字符集

```
[zwc@VM-8-12-centos workspace]$ sudo vim /etc/my.cnf
```

```
# For advice on how to change settings please see
# http://dev.mysql.com/doc/refman/5.7/en/server-configuration-defaults.html
[client]
default-character-set=utf8
[mysql]
default-character-set=utf8
[mysqld]
character-set-server=utf8
#
# Remove leading # and set to the amount of RAM for the most important data
# cache in MySQL. Start at 70% of total RAM for dedicated server, else 10%.
# innodb_buffer_pool_size = 128M
#
# Remove leading # to turn on a very important data integrity option: logging
# changes to the binary log between backups.
# log_bin
#
# Remove leading # to set options mainly useful for reporting servers.
# The server defaults are faster for transactions and fast SELECTs.
# Adjust sizes as needed, experiment to find the optimal values.
# join_buffer_size = 128M
# sort_buffer_size = 2M
# read_rnd_buffer_size = 2M
datadir=/var/lib/mysql
socket=/var/lib/mysql/mysql.sock

# Disabling symbolic-links is recommended to prevent assorted security risks
symbolic-links=0

log-error=/var/log/mysqld.log
```

```
pid-file=/var/run/mysqld/mysqld.pid
```

4.2.12.7 启动Mysql服务

```
[zwc@VM-8-12-centos workspace]$ sudo systemctl start mysqld
[zwc@VM-8-12-centos workspace]$ sudo systemctl status mysqld
● mysqld.service - MySQL Server
   Loaded: loaded (/usr/lib/systemd/system/mysqld.service; enabled; vendor preset: disabled)
   Active: active (running) since Mon 2023-04-17 17:54:00 CST; 9min ago
     Docs: man:mysqld(8)
           http://dev.mysql.com/doc/refman/en/using-systemd.html
  Process: 20047 ExecStart=/usr/sbin/mysqld --daemonize --pid-file=/var/run/mysqld/mysqld.pid $MYSQLD_OPTS (code=exited, status=0/SUCCESS)
  Process: 19988 ExecStartPre=/usr/bin/mysqld_pre_systemd (code=exited, status=0/SUCCESS)
 Main PID: 20051 (mysqld)
    Tasks: 28
   Memory: 189.2M
   CGroup: /system.slice/mysqld.service
           └─20051 /usr/sbin/mysqld --daemonize --pid-file=/var/run/mysqld/mysqld.pid

Apr 17 17:53:59 VM-8-12-centos systemd[1]: Starting MySQL Server...
Apr 17 17:54:00 VM-8-12-centos systemd[1]: Started MySQL Server.
```

注意，如果启动的时候遇到了以下情况，输入系统的root管理员密码即可

```
[zwc@VM-8-12-centos workspace]$ systemctl start mysqld
==== AUTHENTICATING FOR org.freedesktop.systemd1.manage-units ===
Authentication is required to manage system services or units.
Authenticating as:
Password:
```

4.2.12.8 获取Mysql临时密码

```
[zwc@VM-8-12-centos workspace]$ sudo grep 'temporary password' /var/log/mysqld.l
```

4.2.12.9 设置mysql数据库密码

```
[zwc@VM-8-12-centos workspace]$ mysql -uroot -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4
Server version: 5.7.41 MySQL Community Server (GPL)

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> set global validate_password_policy=0;
Query OK, 0 rows affected (0.00 sec)

mysql> set global validate_password_length=1;
Query OK, 0 rows affected (0.00 sec)

mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY 'qwer@wu.888';
Query OK, 0 rows affected (0.00 sec)

mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.00 sec)

```

4.2.12.10 登录查看Mysql字符集是否正常

```
[zwc@VM-8-12-centos workspace]$ mysql -uroot -p Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4
Server version: 5.7.41 MySQL Community Server (GPL)

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show variables like '%chara%';
+--------------------------+----------------------------+
| Variable_name            | Value                      |
```

```
+--------------------------+----------------------------+
| character_set_client     | utf8                       | --客户端使用的字符集
| character_set_connection | utf8                       | --客户端连接时使用的字符集
| character_set_database   | utf8                       | --数据库创建默认字符集
| character_set_filesystem | binary                     | --文件系统编码格式
| character_set_results    | utf8                       | --服务器返回结果时的字符集
| character_set_server     | utf8                       | --存储系统元数据的字符集
| character_set_system     | utf8                       | --系统使用的编码格式,不影响
| character_sets_dir       | /usr/share/mysql/charsets/ |
+--------------------------+----------------------------+
8 rows in set (0.00 sec)

mysql> quit
```

4.2.13 安装Websocketpp库

```
[bite@localhost ~]$ git clone https://github.com/zaphoyd/websocketpp.git
Cloning into 'websocketpp'...
remote: Enumerating objects: 12791, done.
remote: Counting objects: 100% (1549/1549), done.
remote: Compressing objects: 100% (197/197), done.
remote: Total 12791 (delta 1504), reused 1353 (delta 1352), pack-reused 11242
Receiving objects: 100% (12791/12791), 8.37 MiB | 1.47 MiB/s, done.
Resolving deltas: 100% (7985/7985), done.
```

- 安装websocketpp

```
[bite@localhost ~]$ ls
websocketpp
[bite@localhost ~]$ cd websocketpp/
[bite@localhost websocketpp]$ mkdir build
[bite@localhost websocketpp]$ cd build
[bite@localhost build]$ cmake -DCMAKE_INSTALL_PREFIX=/usr ..
...
[bite@localhost build]$ sudo make install
...
```

- 验证websocketpp是否安装成功

```
[bite@localhost build]$ cd ../examples/echo_server
[bite@localhost echo_server]$ ls
CMakeLists.txt  echo_handler.hpp  echo_server.cpp  SConscript
[bite@localhost echo_server]$ g++ -std=c++11 echo_server.cpp -o echo_server -lpthread -lboost_system
[bite@localhost echo_server]$
```

编译成功，则表示安装成功了。

依赖的第三方库

5. 知识点代码用例

5.1 Websocketpp

Websocket介绍

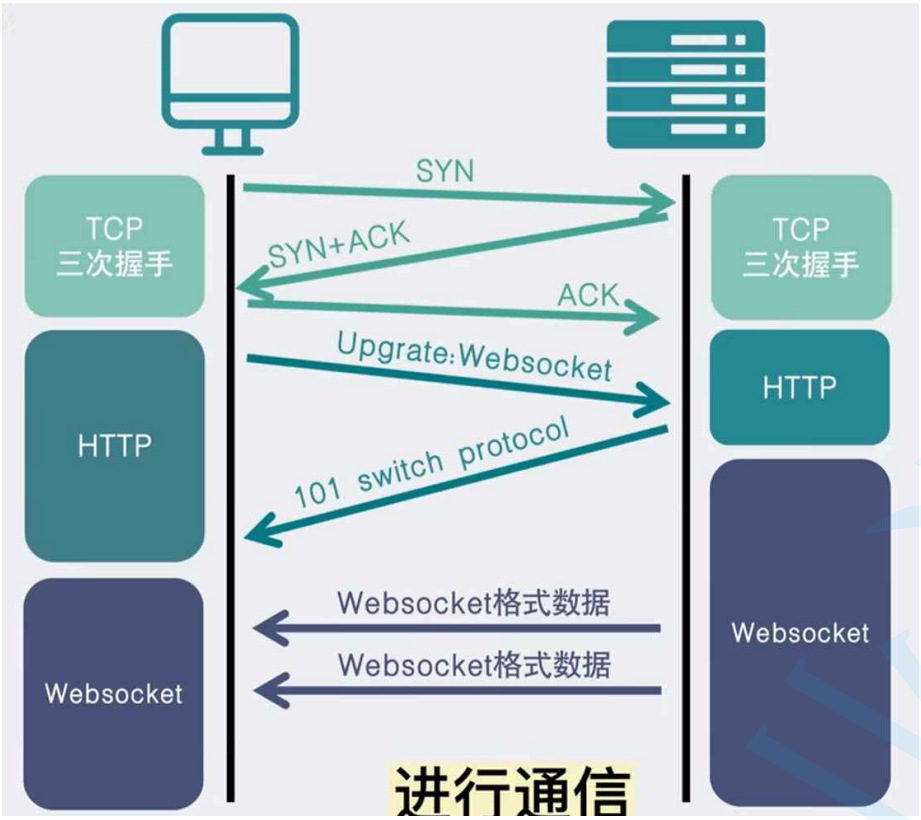
WebSocket 是从 HTML5 开始支持的一种网页端和服务端保持长连接的 消息推送机制。

- 传统的 web 程序都是属于 "一问一答" 的形式，即客户端给服务器发送了一个 HTTP 请求，服务器给客户端返回一个 HTTP 响应。这种情况下服务器是属于被动的一方，如果客户端不主动发起请求服务器就无法主动给客户端响应
- 像网页即时聊天或者我们做的五子棋游戏这样的程序都是非常依赖 "消息推送" 的， 即需要服务器主动推动消息到客户端。如果只是使用原生的 HTTP 协议，要想实现消息推送一般需要通过 "轮询" 的方式实现， 而轮询的成本比较高并且也不能及时的获取到消息的响应。

基于上述两个问题， 就产生了WebSocket协议。WebSocket 更接近于 TCP 这种级别的通信方式，一旦连接建立完成客户端或者服务器都可以主动的向对方发送数据。

原理解析

WebSocket 协议本质上是一个基于 TCP 的协议。为了建立一个 WebSocket 连接，客户端浏览器首先要向服务器发起一个 HTTP 请求，这个请求和通常的 HTTP 请求不同，包含了一些附加头信息，通过这个附加头信息完成握手过程并升级协议的过程。



具体协议升级的过程如下：

报文格式

报文字段比较多，我们重点关注这几个字段:

- FIN: WebSocket传输数据以消息为概念单位，一个消息有可能由一个或多个帧组成，FIN字段为1表示末尾帧。
- RSV1~3：保留字段，只在扩展时使用，若未启用扩展则应置1，若收到不全为0的数据帧，且未协商扩展则立即终止连接。
- opcode: 标志当前数据帧的类型
  - 0x0: 表示这是个延续帧，当 opcode 为 0 表示本次数据传输采用了数据分片，当前收到的帧为其中一个分片
  - 0x1: 表示这是文本帧
  - 0x2: 表示这是二进制帧
  - 0x3-0x7: 保留，暂未使用
  - 0x8: 表示连接断开
  - 0x9: 表示 ping 帧
  - 0xa: 表示 pong 帧
  - 0xb-0xf: 保留，暂未使用
- mask：表示Payload数据是否被编码，若为1则必有Mask-Key，用于解码Payload数据。仅客户端发送给服务端的消息需要设置。
- Payload length：数据载荷的长度，单位是字节， 有可能为7位、7+16位、7+64位。假设Payload length = x
  - x为0~126：数据的长度为x字节
  - x为126：后续2个字节代表一个16位的无符号整数，该无符号整数的值为数据的长度
  - x为127：后续8个字节代表一个64位的无符号整数（最高位为0），该无符号整数的值为数据的长度

- Mask-Key：当mask为1时存在，长度为4字节，解码规则： DECODED[i] = ENCODED[i] ^ MASK[i % 4]
- Payload data: 报文携带的载荷数据

注: B站的这个视频对Websocket协议的讲述非常清晰， 大家下去可以去看看。

https://www.bilibili.com/video/BV1684y1k7VP/?buvid=ZC4691C539D91BA74044…

为什么有HTTP还要有websocket_哔哩哔哩_bilibili

为什么有HTTP还要有websocket?, 视频播放量 119872、弹幕量 191、点赞数 5983、投硬币枚数 3648、收藏人数 6639、转发人数 536, 视频作者 小白debug, 作者简介 …

Websocketpp介绍

WebSocketpp是一个跨平台的开源（BSD许可证）头部专用C++库，它实现了RFC6455（WebSocket协议）和RFC7692（WebSocketCompression Extensions）。它允许将WebSocket客户端和服务器功能集成到C++程序中。在最常见的配置中，全功能网络I/O由Asio网络库提供。

WebSocketpp的主要特性包括：

- 事件驱动的接口
- 支持HTTP/HTTPS、WS/WSS、IPv6
- 灵活的依赖管理 — Boost库/C++11标准库
- 可移植性：Posix/Windows、32/64bit、Intel/ARM
- 线程安全

WebSocketpp同时支持HTTP和Websocket两种网络协议， 比较适用于我们本次的项目， 所以我们选用该库作为项目的依赖库用来搭建HTTP和WebSocket服务器。

下面是该项目的一些常用网站， 大家多去学习。

- github：https://github.com/zaphoyd/websocketpp
- 用户手册： http://docs.websocketpp.org/
- 官网：http://www.zaphoyd.com/websocketpp

Websocketpp使用

websocketpp常用接口介绍：

```
namespace websocketpp {
    typedef lib::weak_ptr<void> connection_hdl;

    template <typename config>
    class endpoint : public config::socket_type {
        typedef lib::shared_ptr<lib::asio::steady_timer> timer_ptr;
```

```
7         typedef typename connection_type::ptr connection_ptr;
8         typedef typename connection_type::message_ptr message_ptr;
9
10        typedef lib::function<void(connection_hdl)> open_handler;
11        typedef lib::function<void(connection_hdl)> close_handler;
12        typedef lib::function<void(connection_hdl)> http_handler;
13        typedef lib::function<void(connection_hdl,message_ptr)>
message_handler;
14        /* websocketpp::log::alevel::none  禁止打印所有日志*/
15        void set_access_channels(log::level channels);/*设置日志打印等级*/
16        void clear_access_channels(log::level channels);/*清除指定等级的日志*/
17        /*设置指定事件的回调函数*/
18        void set_open_handler(open_handler h);/*websocket握手成功回调处理函数*/
19        void set_close_handler(close_handler h);/*websocket连接关闭回调处理函数*/
20        void set_message_handler(message_handler h);/*websocket消息回调处理函数*/
21        void set_http_handler(http_handler h);/*http请求回调处理函数*/
22        /*发送数据接口*/
23        void send(connection_hdl hdl, std::string& payload,
frame::opcode::value op);
24        void send(connection_hdl hdl, void* payload, size_t len,
frame::opcode::value op);
25        /*关闭连接接口*/
26        void close(connection_hdl hdl, close::status::value code, std::string&
reason);
27        /*获取connection_hdl 对应连接的connection_ptr*/
28        connection_ptr get_con_from_hdl(connection_hdl hdl);
29        /*websocketpp基于asio框架实现，init_asio用于初始化asio框架中的io_service调度
器*/
30        void init_asio();
31        /*设置是否启用地址重用*/
32        void set_reuse_addr(bool value);
33        /*设置endpoint的绑定监听端口*/
34        void listen(uint16_t port);
35        /*对io_service对象的run接口封装，用于启动服务器*/
36        std::size_t run();
37        /*websocketpp提供的定时器，以毫秒为单位*/
38        timer_ptr set_timer(long duration, timer_handler callback);
39    };
40    template <typename config>
41    class server : public endpoint<connection<config>,config> {
42        /*初始化并启动服务端监听连接的accept事件处理*/
43        void start_accept();
44    };
45
46    template <typename config>
47    class connection
48     : public config::transport_type::transport_con_type
```

```
49      , public config::connection_base
50      {
51          /*发送数据接口*/
52          error_code send(std::string&payload, frame::opcode::value
    op=frame::opcode::text);
53          /*获取http请求头部*/
54          std::string const & get_request_header(std::string const & key)
55          /*获取请求正文*/
56          std::string const & get_request_body();
57          /*设置响应状态码*/
58          void set_status(http::status_code::value code);
59          /*设置http响应正文*/
60          void set_body(std::string const & value);
61          /*添加http响应头部字段*/
62          void append_header(std::string const & key, std::string const & val);
63          /*获取http请求对象*/
64          request_type const & get_request();
65          /*获取connection_ptr 对应的 connection_hdl */
66          connection_hdl get_handle();
67      };
68
69      namespace http {
70      namespace parser {
71          class parser {
72              std::string const & get_header(std::string const & key)
73          }
74          class request : public parser {
75              /*获取请求方法*/
76              std::string const & get_method()
77              /*获取请求uri接口*/
78              std::string const & get_uri()
79          };
80      }};
81
82      namespace message_buffer {
83          /*获取websocket请求中的payload数据类型*/
84          frame::opcode::value get_opcode();
85          /*获取websocket中payload数据*/
86          std::string const & get_payload();
87      };
88
89      namespace log {
90          struct alevel {
91              static level const none = 0x0;
92              static level const connect = 0x1;
93              static level const disconnect = 0x2;
94              static level const control = 0x4;
```

```
            static level const frame_header = 0x8;
            static level const frame_payload = 0x10;
            static level const message_header = 0x20;
            static level const message_payload = 0x40;
            static level const endpoint = 0x80;
            static level const debug_handshake = 0x100;
            static level const debug_close = 0x200;
            static level const devel = 0x400;
            static level const app = 0x800;
            static level const http = 0x1000;
            static level const fail = 0x2000;
            static level const access_core = 0x00003003;
            static level const all = 0xffffffff;
        };
    }

    namespace http {
    namespace status_code {
        enum value {
            uninitialized = 0,

            continue_code = 100,
            switching_protocols = 101,

            ok = 200,
            created = 201,
            accepted = 202,
            non_authoritative_information = 203,
            no_content = 204,
            reset_content = 205,
            partial_content = 206,

            multiple_choices = 300,
            moved_permanently = 301,
            found = 302,
            see_other = 303,
            not_modified = 304,
            use_proxy = 305,
            temporary_redirect = 307,

            bad_request = 400,
            unauthorized = 401,
            payment_required = 402,
            forbidden = 403,
            not_found = 404,
            method_not_allowed = 405,
            not_acceptable = 406,
```

```
            proxy_authentication_required = 407,
            request_timeout = 408,
            conflict = 409,
            gone = 410,
            length_required = 411,
            precondition_failed = 412,
            request_entity_too_large = 413,
            request_uri_too_long = 414,
            unsupported_media_type = 415,
            request_range_not_satisfiable = 416,
            expectation_failed = 417,
            im_a_teapot = 418,
            upgrade_required = 426,
            precondition_required = 428,
            too_many_requests = 429,
            request_header_fields_too_large = 431,

            internal_server_error = 500,
            not_implemented = 501,
            bad_gateway = 502,
            service_unavailable = 503,
            gateway_timeout = 504,
            http_version_not_supported = 505,
            not_extended = 510,
            network_authentication_required = 511
        };}}
        namespace frame {
        namespace opcode {
        enum value {
            continuation = 0x0,
            text = 0x1,
            binary = 0x2,
            rsv3 = 0x3,
            rsv4 = 0x4,
            rsv5 = 0x5,
            rsv6 = 0x6,
            rsv7 = 0x7,
            close = 0x8,
            ping = 0x9,
            pong = 0xA,
            control_rsvb = 0xB,
            control_rsvc = 0xC,
            control_rsvd = 0xD,
            control_rsve = 0xE,
            control_rsvf = 0xF,
        };}}
}
```

Simple http/websocket服务器

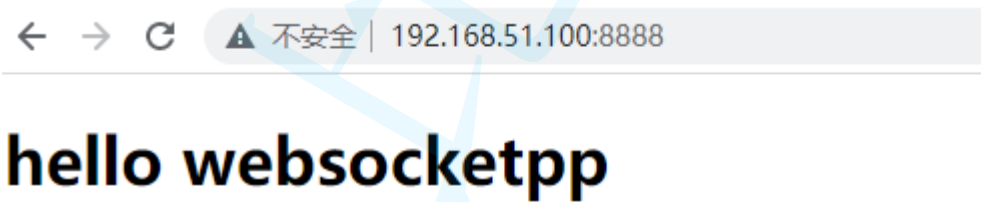
使用Websocketpp实现一个简单的http和websocket服务器

```
#include <iostream>
#include <websocketpp/config/asio_no_tls.hpp>
#include <websocketpp/server.hpp>

using namespace std;

typedef websocketpp::server<websocketpp::config::asio> websocketsvr;
typedef websocketsvr::message_ptr message_ptr;

using websocketpp::lib::placeholders::_1;
using websocketpp::lib::placeholders::_2;
using websocketpp::lib::bind;

// websocket连接成功的回调函数
void OnOpen(websocketsvr *server,websocketpp::connection_hdl hdl){
    cout<<"连接成功"<<endl;
}

// websocket连接成功的回调函数
void OnClose(websocketsvr *server,websocketpp::connection_hdl hdl){
    cout<<"连接关闭"<<endl;
}

// websocket连接收到消息的回调函数
void OnMessage(websocketsvr *server,websocketpp::connection_hdl hdl,message_ptr
    cout << "收到消息" << msg->get_payload() << endl;
    // 收到消息将相同的消息发回给websocket客户端
    server->send(hdl, msg->get_payload(), websocketpp::frame::opcode::text);
}

// websocket连接异常的回调函数
void OnFail(websocketsvr *server,websocketpp::connection_hdl hdl){
    cout<<"连接异常"<<endl;
}

// 处理http请求的回调函数 返回一个html欢迎页面
void OnHttp(websocketsvr *server,websocketpp::connection_hdl hdl){
    cout<<"处理http请求"<<endl;
    websocketsvr::connection_ptr con = server->get_con_from_hdl(hdl);
    std::stringstream ss;
    ss << "<!doctype html><html><head>"
```

```
42          << "<title>hello websocket</title><body>"
43          << "<h1>hello websocketpp</h1>"
44          << "</body></head></html>";
45      con->set_body(ss.str());
46      con->set_status(websocketpp::http::status_code::ok);
47  }
48
49  int main(){
50      // 使用websocketpp库创建服务器
51      websocketsvr server;
52      // 设置websocketpp库的日志级别
53      // all表示打印全部级别日志
54      // none表示什么日志都不打印
55      server.set_access_channels(websocketpp::log::alevel::none);
56      /*初始化asio*/
57      server.init_asio();
58      // 注册http请求的处理函数
59      server.set_http_handler(bind(&OnHttp, &server, ::_1));
60      // 注册websocket请求的处理函数
61      server.set_open_handler(bind(&OnOpen, &server, ::_1));
62      server.set_close_handler(bind(&OnClose, &server, _1));
63      server.set_message_handler(bind(&OnMessage,&server,_1,_2));
64      // 监听8888端口
65      server.listen(8888);
66      // 开始接收tcp连接
67      server.start_accept();
68      // 开始运行服务器
69      server.run();
70      return 0;
71  }
72
```

Http客户端

使用浏览器作为http客户端即可， 访问服务器的8888端口。

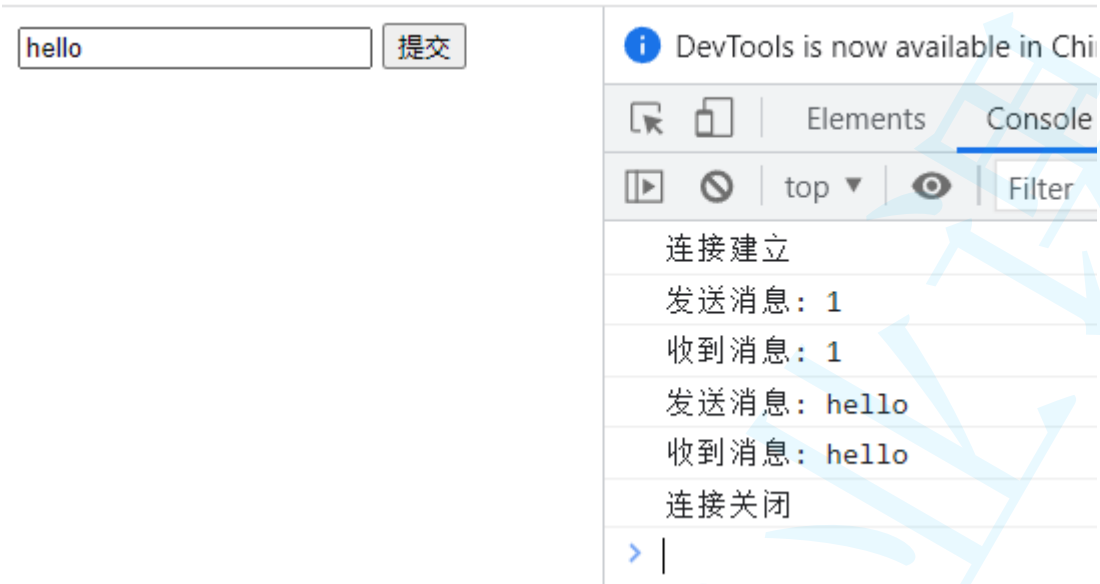


WS客户端

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Test Websocket</title>
</head>
<body>
    <input type="text" id="message">
    <button id="submit">提交</button>

<script>
    // 创建 websocket 实例
    // ws://192.168.51.100:8888
    // 类比http
    // ws表示websocket协议
    // 192.168.51.100 表示服务器地址
    // 8888表示服务器绑定的端口
    let websocket = new WebSocket("ws://192.168.51.100:8888");

    // 处理连接打开的回调函数
    websocket.onopen = function() {
        console.log("连接建立");
    }
    // 处理收到消息的回调函数
    // 控制台打印消息
    websocket.onmessage = function(e) {
        console.log("收到消息: " + e.data);
    }
    // 处理连接异常的回调函数
    websocket.onerror = function() {
        console.log("连接异常");
    }
    // 处理连接关闭的回调函数
    websocket.onclose = function() {
        console.log("连接关闭");
    }

    // 实现点击按钮后, 通过 websocket实例 向服务器发送请求
    let input = document.querySelector('#message');
    let button = document.querySelector('#submit');
    button.onclick = function() {
        console.log("发送消息: " + input.value);
        websocket.send(input.value);
    }
</script>
```

```
48 </body>
49 </html>
```

在控制台中我们可以看到连接建立、客户端和服务器通信以及断开连接的过程(关闭服务器就会看到断开连接的现象)



注: 通过f12 或者 fn + f12打开浏览器的调试模式

5.2 JsonCpp使用

Json数据格式

`Json` 是一种数据交换格式，它采用完全独立于编程语言的文本格式来存储和表示数据。

例如: 我们想表示一个同学的学生信息

- `C` 代码表示

```
1 char *name = "xx";
2 int age = 18;
3 float score[3] = {88.5, 99, 58};
```

- `Json` 表示

```
1 {
2     "姓名" : "xx",
3     "年龄" : 18,
4     "成绩" : [88.5, 99, 58]
5 }
6
```

```
[
    {"姓名":"小明", "年龄":18, "成绩":[23, 65, 78]},
    {"姓名":"小红", "年龄":19, "成绩":[88, 95, 78]}
]
```

`Json`的数据类型包括对象，数组，字符串，数字等。

- 对象：使用花括号`{}`括起来的表示一个对象
- 数组：使用中括号`[]`括起来的表示一个数组
- 字符串：使用常规双引号`""`括起来的表示一个字符串
- 数字：包括整形和浮点型，直接使用

JsonCpp介绍

`Jsoncpp`库主要是用于实现`Json`格式数据的序列化和反序列化，它实现了将多个数据对象组织成为`json`格式字符串，以及将`Json`格式字符串解析得到多个数据对象的功能。

先看一下`Json`数据对象类的表示

```
class Json::Value{
    Value &operator=(const Value &other); //Value重载了[]和=，因此所有的赋值和获取数据都可以通过
    Value& operator[](const std::string& key);//简单的方式完成 val["name"] = "xx";
    Value& operator[](const char* key);
    Value removeMember(const char* key);//移除元素
    const Value& operator[](ArrayIndex index) const; //val["score"][0]
    Value& append(const Value& value);//添加数组元素val["score"].append(88);
    ArrayIndex size() const;//获取数组元素个数 val["score"].size();
    bool isNull();  //用于判断是否存在某个字段
    std::string asString() const;//转string string name = val["name"].asString();
    const char* asCString() const;//转char* char *name = val["name"].asCString();
    Int asInt() const;//转int int age = val["age"].asInt();
    float asFloat() const;//转float float weight = val["weight"].asFloat();
    bool asBool() const;//转 bool bool ok = val["ok"].asBool();
};
```

`Jsoncpp`库主要借助三个类以及其对应的少量成员函数完成序列化及反序列化

- 序列化接口

```
class JSON_API StreamWriter {
    virtual int write(Value const& root, std::ostream* sout) = 0;
}
class JSON_API StreamWriterBuilder : public StreamWriter::Factory {
    virtual StreamWriter* newStreamWriter() const;
}
```

- 反序列化接口

```
class JSON_API CharReader {
    virtual bool parse(char const* beginDoc, char const* endDoc,
                       Value* root, std::string* errs) = 0;
}
class JSON_API CharReaderBuilder : public CharReader::Factory {
    virtual CharReader* newCharReader() const;
}
```

JsonCpp功能代码用例编写

```
#include <iostream>
#include <sstream>
#include <string>
#include <memory>
#include <jsoncpp/json/json.h>

int main()
{
    // 序列化
    Json::Value stu;
    stu["name"] = "zhangsan";
    stu["age"] = 19;
    stu["socre"].append(77.5);
    stu["socre"].append(88);
    stu["socre"].append(99.5);

    Json::StreamWriterBuilder swb;
    std::unique_ptr<Json::StreamWriter> sw(swb.newStreamWriter());

    std::stringstream ss;
    int ret = sw->write(stu, &ss);
    if (ret != 0) {
        std::cout << "Serialize  failed!\n";
        return -1;
```

```
    }
    std::cout << "序列化结果:\n" << ss.str() << std::endl;

    // 反序列化
    std::string str = ss.str();
    Json::Value root;
    Json::CharReaderBuilder crb;
    std::unique_ptr<Json::CharReader> cr(crb.newCharReader());

    bool ret1 = cr->parse(str.c_str(), str.c_str() + str.size(), &root,
nullptr);
    if (!ret1) {
        std::cout << "UnSerialize failed!" << std::endl;
        return -1;
    }
    std::cout << "反序列化结果:\n"
              << "name:" << root["name"].asString() << "\n"
              <<  "age:" << root["age"].asInt() << "\n"
              << "socre:" << root["socre"][0].asFloat() << " " << root["socre"]
[1].asInt()
              << " " << root["socre"][2].asFloat() << "\n";

    return 0;
}
```

编译运行程序查看序列化和反序列化结果

```
[zsc@node test_jsoncpp]$ g++ test_json.cpp -o test_json -ljsoncpp -std=c++11
[zsc@node test_jsoncpp]$ ./test_json
序列化结果:
{
        "age" : 19,
        "name" : "zhangsan",
        "socre" :
        [
                77.5,
                88,
                99.5
        ]
}
反序列化结果:
name:zhangsan
age:19
socre:77.5 88 99.5
```

封装Json工具类

```cpp
class json_util
{
public:
    // 序列化：Json对象 -> 字符串
    // 输入输出型参数
    // root输入参数：表示要序列化的json对象
    // str输出参数：表示序列化之后的字符串
    static bool serialize(const Json::Value &root, std::string &str)
    {
        Json::StreamWriterBuilder swb;
        std::unique_ptr<Json::StreamWriter> sw(swb.newStreamWriter());

        std::stringstream ss;
        int ret = sw->write(root, &ss);
        if (ret != 0) {
            std::cout << "Serialize failed!" << std::endl;
            return false;
        }
        str = ss.str();
        return true;
    }

    // 反序列化：字符串 ->Json对象
    // 输入输出型参数
    // str输入参数：表示需要反序列化的字符串
    // root输出参数：表示反序列化后的json对象
    static bool unserialize(const std::string &str, Json::Value &root)
    {
        Json::CharReaderBuilder crb;
        std::unique_ptr<Json::CharReader> cr(crb.newCharReader());

        bool ret = cr->parse(str.c_str(), str.c_str() + str.size(), &root,
nullptr);
        if (!ret) {
            std::cout << "UnSerialize failed!" << std::endl;
            return false;
        }
        return true;
    }
};
```

5.3 MySQL API

MySQL API介绍

- `MySQL` 是 `C/S` 模式， `C API` 其实就是一个 `MySQL` 客户端，提供一种用 `C` 语言代码操作数据库的流程
- 课堂上同学们已经学习了 `MySQL` 数据库的学习，在这里我们主要介绍一下 `MySQL` 的 `C API` 接口

```
// Mysql操作句柄初始化
// 参数说明:
//     mysql为空则动态申请句柄空间进行初始化
// 返回值: 成功返回句柄指针， 失败返回NULL
MYSQL *mysql_init(MYSQL *mysql);

// 连接mysql服务器
// 参数说明:
//     mysql--初始化完成的句柄
//     host---连接的mysql服务器的地址
//     user---连接的服务器的用户名
//     passwd-连接的服务器的密码
//     db ----默认选择的数据库名称
//     port---连接的服务器的端口： 默认0是3306端口
//     unix_socket---通信管道文件或者socket文件，通常置NULL
//     client_flag---客户端标志位，通常置0
// 返回值: 成功返回句柄指针，失败返回NULL
MYSQL *mysql_real_connect(MYSQL *mysql,  const char *host,  const char *user,
                          const char *passwd,const char *db,   unsigned int
port,
                          const char *unix_socket,   unsigned long
client_flag);

// 设置当前客户端的字符集
// 参数说明:
//    mysql--初始化完成的句柄
//    csname--字符集名称，通常："utf8"
// 返回值: 成功返回0， 失败返回非0
int mysql_set_character_set(MYSQL *mysql, const char *csname)

// 选择操作的数据库
// 参数说明:
//    mysql--初始化完成的句柄
//    db-----要切换选择的数据库名称
// 返回值: 成功返回0， 失败返回非0
int  mysql_select_db(MYSQL *mysql,  const char *db)

// 执行sql语句
// 参数说明:
```

```
38 //    mysql--初始化完成的句柄
39 //    stmt_str--要执行的sql语句
40 // 返回值：成功返回0， 失败返回非0
41 int  mysql_query(MYSQL *mysql,  const char *stmt_str)
42
43 // 保存查询结果到本地
44 // 参数说明：
45 //    mysql--初始化完成的句柄
46 // 返回值：成功返回结果集的指针， 失败返回NULL
47 MYSQL_RES *mysql_store_result(MYSQL *mysql)
48
49 // 获取结果集中的行数
50 // 参数说明：
51 //    result--保存到本地的结果集地址
52 // 返回值：结果集中数据的条数
53 uint64_t mysql_num_rows(MYSQL_RES *result);
54
55 // 获取结果集中的列数
56 // 参数说明：
57 //    result--保存到本地的结果集地址
58 // 返回值：结果集中每一条数据的列数
59 unsigned int mysql_num_fields(MYSQL_RES *result)
60
61 // 遍历结果集，并且这个接口会保存当前读取结果位置，每次获取的都是下一条数据
62 // 参数说明：
63 //    result--保存到本地的结果集地址
64 // 返回值：实际上是一个char **的指针，将每一条数据做成了字符串指针数组
65 //        row[0]-第0列 row[1]-第1列 ...
66 MYSQL_ROW mysql_fetch_row(MYSQL_RES *result)
67
68 // 释放结果集
69 // 参数说明：
70 //    result--保存到本地的结果集地址
71 void mysql_free_result(MYSQL_RES *result)
72
73 // 关闭数据库客户端连接，销毁句柄
74 // 参数说明：
75 //    mysql--初始化完成的句柄
76 void mysql_close(MYSQL *mysql)
77
78 // 获取mysql接口执行错误原因
79 // 参数说明：
80 //    mysql--初始化完成的句柄
81 const char *mysql_error(MYSQL *mysql)
```

MySQL API使用

下面我们使用 `C API` 来实现 `MySQL` 的增删改查操作

- 创建测试数据库

```
create database if not exists test_db;
use test_db;
create table stu(
    id int primary key auto_increment, -- 学生id
    age int,  -- 学生年龄
    name varchar(32)  -- 学生姓名
);
```

- 连接MySQL服务， 进入shell并执行sql语句

```
[zsc@node test_mysql]$ mysql -uroot -p123456
MariaDB [(none)]> create database if not exists test_db;
Query OK, 1 row affected (0.01 sec)

MariaDB [(none)]> use test_db;
Database changed
MariaDB [test_db]> create table stu(
    ->     id int primary key auto_increment, -- 学生id
    ->     age int,  -- 学生年龄
    ->     name varchar(32)  -- 学生姓名
    -> );
Query OK, 0 rows affected (0.02 sec)
```

- 实现增删改查操作

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <mysql/mysql.h>

#define HOST "127.0.0.1"
#define USER "root"
#define PASSWD "123456"
#define DBNAME "test_db"

void add(MYSQL *mysql) {
    char *sql = "insert into stu values(null, 18, '张三'), (null, 17, '李四');";
    int ret = mysql_query(mysql, sql);
```

```
15      if (ret != 0) {
16          printf("mysql query error:%s\n", mysql_error(mysql));
17          return;
18      }
19      return;
20  }
21
22  void del(MYSQL *mysql) {
23      char *sql = "delete from stu where name='张三';";
24      int ret = mysql_query(mysql, sql);
25      if (ret != 0) {
26          printf("mysql query error:%s\n", mysql_error(mysql));
27          return;
28      }
29      return;
30  }
31
32  void mod(MYSQL *mysql) {
33      char *sql = "update stu set age=15 where name='张三';";
34      int ret = mysql_query(mysql, sql);
35      if (ret != 0) {
36          printf("mysql query error:%s\n", mysql_error(mysql));
37          return;
38      }
39      return;
40  }
41
42  void get(MYSQL *mysql) {
43
44      char *sql = "select * from stu;";
45      int ret = mysql_query(mysql, sql);
46      if (ret != 0) {
47          printf("mysql query error:%s\n", mysql_error(mysql));
48          return ;
49      }
50      MYSQL_RES *res =  mysql_store_result(mysql);
51      if (res == NULL) {
52          printf("mysql store result error:%s\n", mysql_error(mysql));
53          return ;
54      }
55      int row = mysql_num_rows(res);
56      int col = mysql_num_fields(res);
57      printf("%10s%10s%10s\n", "ID", "年龄", "姓名");
58      for (int i = 0; i < row; i++) {
59          MYSQL_ROW row_data = mysql_fetch_row(res);
60          for (int i = 0; i < col; i++) {
61              printf("%10s", row_data[i]);
```

```
        }
        printf("\n");
    }
    mysql_free_result(res);
    return ;
}
int main()
{
    MYSQL *mysql = mysql_init(NULL);
    if (mysql == NULL) {
        printf("init mysql handle failed!\n");
        return -1;
    }
    if (mysql_real_connect(mysql, HOST, USER, PASSWD, DBNAME, 0, NULL, 0) ==
NULL) {
        printf("mysql connect error:%s\n", mysql_error(mysql));
        return -1;
    }
    mysql_set_character_set(mysql, "utf8");

    printf("====================== add ========================\n");
    add(mysql);
    get(mysql);
    printf("====================== mod ========================\n");
    mod(mysql);
    get(mysql);
    printf("====================== del ========================\n");
    del(mysql);
    get(mysql);

    mysql_close(mysql);
    return 0;
}
```

- 验证结果

```
[zsc@node test_mysql]$ g++ test_mysql.cpp -o test_mysql -L/usr/lib64/mysql -lmys
[zsc@node test_mysql]$ ./test_mysql
====================== add ========================
        ID    年龄    姓名
        11        18    张三
        12        17    李四
====================== mod ========================
        ID    年龄    姓名
        11        15    张三
```

```
        12        17    李四
====================== del ========================
        ID    年龄    姓名
        12        17    李四
```

封装MySQL工具类

```cpp
class mysql_util
{
public:
    // 创建mysql连接
    static MYSQL *mysql_create(const std::string &host,
                               const std::string &user,
                               const std::string &pass,
                               const std::string &name,
                               uint16_t port)
    {
        // 初始化mysql句柄
        MYSQL *mysql = mysql_init(nullptr);
        if (mysql == nullptr) {
            std::cout << "Init mysql instance failed!" << std::endl;
            return nullptr;
        }
        // 连接mysql服务
        if (mysql_real_connect(mysql, host.c_str(), user.c_str(),
pass.c_str(), name.c_str(), port, nullptr, 0) == nullptr) {
            std::cout << "Connect mysql server failed!" << std::endl;
            std::cout << mysql_error(mysql) << std::endl;
            mysql_close(mysql);
            return nullptr;
        }
        // 设置字符集为utf-8
        mysql_set_character_set(mysql, "utf8");
        return mysql;
    }

    // 关闭mysql连接
    static void mysql_destroy(MYSQL *mysql)
    {
        if (mysql != nullptr) {
            mysql_close(mysql);
        }
        return;
    }

```

```
    // 执行sql语句
    static bool mysql_exec(MYSQL *mysql, const std::string &sql)
    {
        int ret = mysql_query(mysql, sql.c_str());
        if (ret != 0) {
            std::cout << sql << std::endl;
            std::cout << mysql_error(mysql) << std::endl;
            return false;
        }
        return true;
    }
};
```

5.4 前端知识介绍

参考比特《前端扫盲》课件

6. 项目结构设计

6.1 项目模块划分说明

项目的实现，咱们将其划分为三个大模块来进行：

- 数据管理模块：基于Mysql数据库进行用户数据的管理
- 前端界面模块：基于JS实现前端页面(注册，登录，游戏大厅，游戏房间)的动态控制以及与服务器的通信。
- 业务处理模块：搭建WebSocket服务器与客户端进行通信，接收请求并进行业务处理。

在这里回顾一下我们要实现的项目功能，我们要实现的是一个在线五子棋对战服务器，提供用户通过浏览器进行用户注册，登录，以及实时匹配，对战，聊天等功能。

而如果要实现这些功能，那么就需要对业务处理模块再次进行细分为多个模块来实现各个功能。

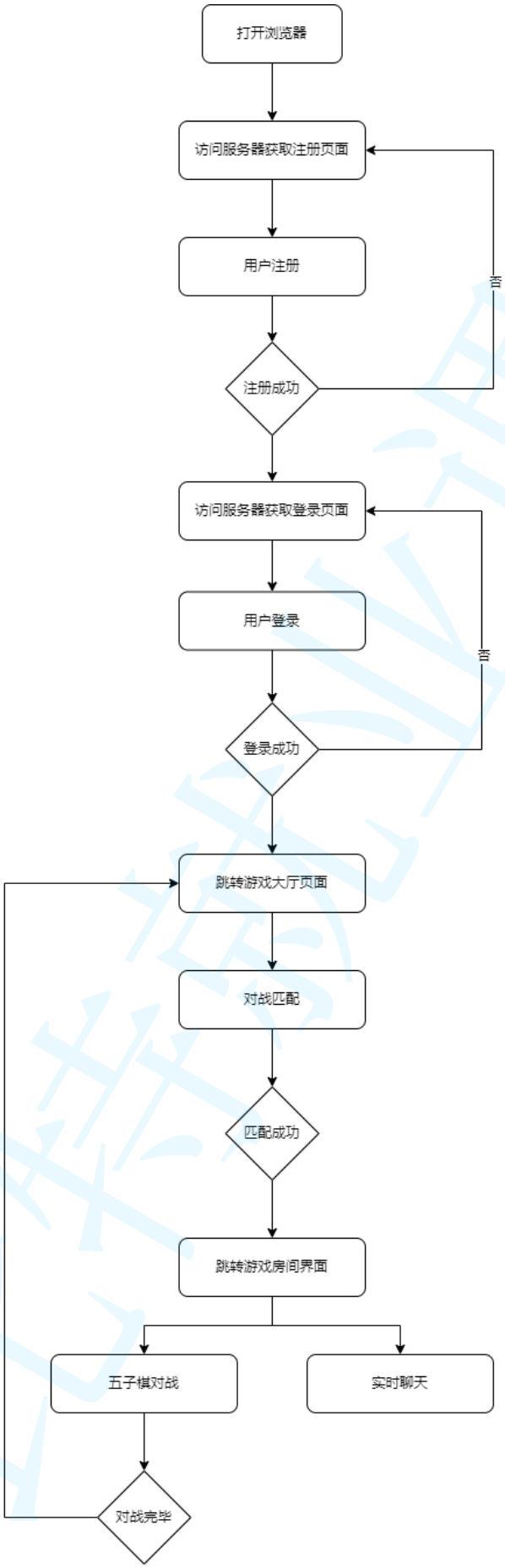
6.2 业务处理模块的子模块划分：

- 网络通信模块：基于websocketpp库实现Http&WebSocket服务器的搭建，提供网络通信功能。
- 会话管理模块：对客户端的连接进行cookie&session管理，实现http短连接时客户端身份识别功能。
- 在线管理模块：对进入游戏大厅与游戏房间中用户进行管理，提供用户是否在线以及获取用户连接的功能。
- 房间管理模块：为匹配成功的用户创建对战房间，提供实时的五子棋对战与聊天业务功能。

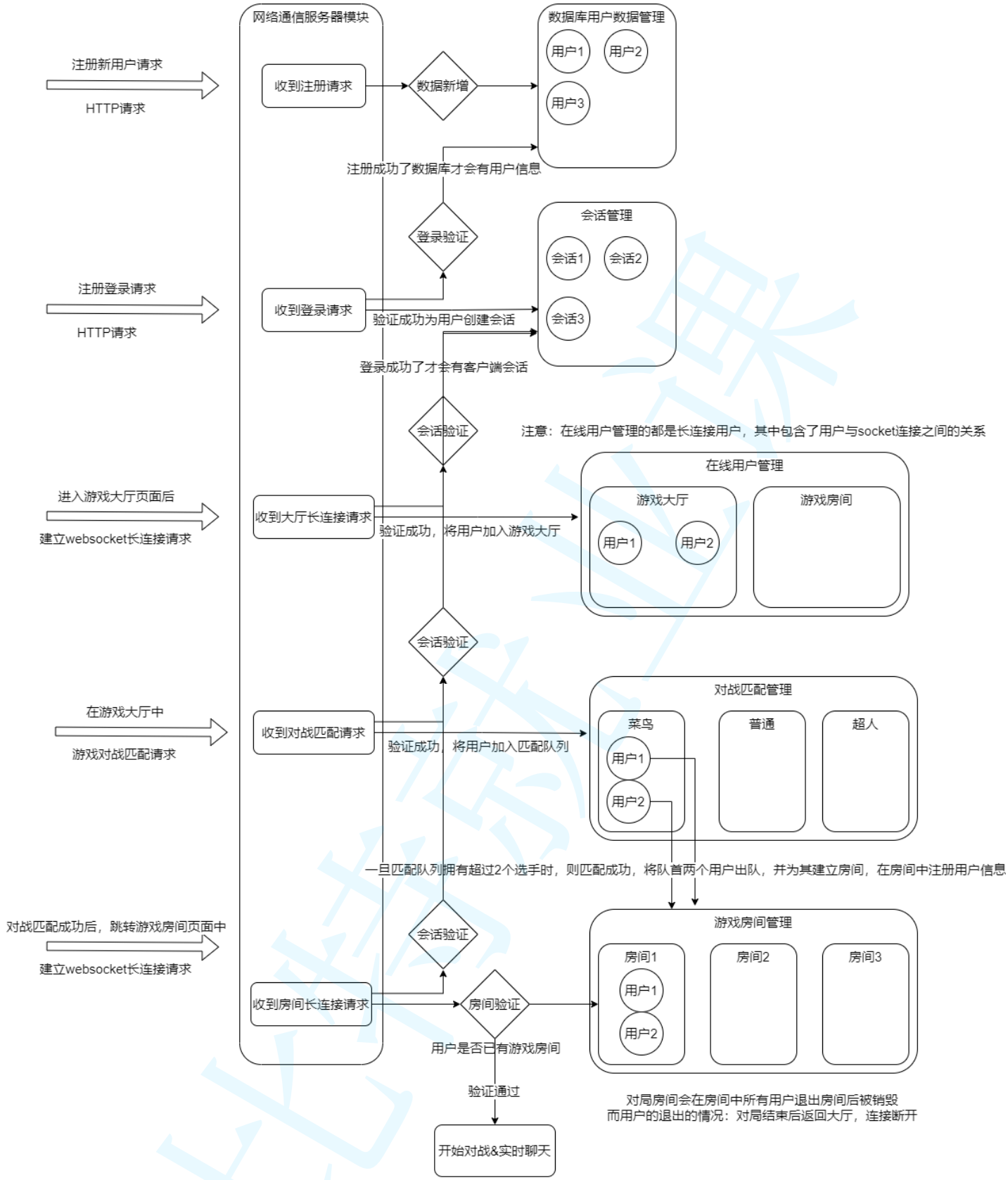
- 用户匹配模块：根据天梯分数不同进行不同层次的玩家匹配，为匹配成功的玩家创建房间并加入房间。

6.3 项目流程图

6.3.1 玩家用户角度流程图：



6.3.2 服务器流程结构图：



每一个用户接下来的通信都是发送请求给服务器，服务器通过用户ID获取到所在房间ID，然后将请求广播给房间内的所有用户（当然下棋会有棋局信息的逻辑处理）

每个客户端用户都是收到了一个服务器发送过来的请求后，才会将对应的棋子，或者聊天信息展示出来

7. 实用工具类模块代码实现

实用工具类模块主要是负责提前实现一些项目中会用到的边缘功能代码，提前实现好了就可以在项目中用到的时候直接使用了。

7.1 日志宏封装

```
#define INF 0
#define DBG 1
#define ERR 2
#define LOG_LEVEL DBG
#define LOG(level, format, ...) do{\
        if (level < LOG_LEVEL) break;\
        time_t t = time(NULL);\
        struct tm *ltm = localtime(&t);\
        char tmp[32] = {0};\
        strftime(tmp, 31, "%H:%M:%S", ltm);\
        fprintf(stdout, "[%p %s %s:%d] " format "\n", (void*)pthread_self(), tmp, __FILE__, __LINE__, ##__VA_ARGS__);\
    }while(0)

#define INF_LOG(format, ...) LOG(INF, format, ##__VA_ARGS__)
#define DBG_LOG(format, ...) LOG(DBG, format, ##__VA_ARGS__)
#define ERR_LOG(format, ...) LOG(ERR, format, ##__VA_ARGS__)
```

7.2 Mysql-API封装

```
class mysql_util {
    public:
        static MYSQL *mysql_create(
            const std::string &host,
            const std::string &user,
            const std::string &pass,
            const std::string &db,
            int port) {
            MYSQL *mysql = mysql_init(NULL);
            if (mysql == NULL) {
                ERR_LOG("mysql init failed!");
                return NULL;
            }
            if (mysql_real_connect(mysql, host.c_str(), user.c_str(), pass.c_str(), db.c_str(), port, NULL, 0) == NULL) {
                ERR_LOG("mysql connect server failed! %s", mysql_error(mysql));
                mysql_close(mysql);
                return NULL;
            }
            if (mysql_set_character_set(mysql, "utf8mb4") != 0) {
                ERR_LOG("mysql set character failed!");
                mysql_close(mysql);
                return NULL;
            }
```

```
            return mysql;
        }
        static void mysql_release(MYSQL *mysql) {
            if (mysql == NULL) {
                return;
            }
            mysql_close(mysql);
            return ;
        }
        static bool mysql_exec(MYSQL *mysql, const std::string &sql) {
            if (mysql_query(mysql, sql.c_str()) != 0) {
                ERR_LOG("SQL: %s", sql.c_str());
                ERR_LOG("ERR: %s", mysql_error(mysql));
                return false;
            }
            return true;
        }
};
```

7.3 Jsoncpp-API封装

```
class json_util {
    public:
        static bool serialize(const Json::Value &value, std::string &str) {
            Json::StreamWriterBuilder swb;
            std::unique_ptr<Json::StreamWriter> sw(swb.newStreamWriter());
            std::stringstream ss;
            int ret = sw->write(value, &ss);
            if (ret != 0) {
                std::cout << "json serialize failed!" << std::endl;
                return false;
            }
            str = ss.str();
            return true;
        }
        static bool unserialize(const std::string &str, Json::Value &value) {
            Json::CharReaderBuilder crb;
            std::unique_ptr<Json::CharReader> cr(crb.newCharReader());
            bool ret = cr->parse(str.c_str(), str.c_str() + str.size(),
&value, nullptr);
            if (!ret) {
                ERR_LOG("json unserialize failed!");
                return false;
            }
            return true;
```

```
        }
};
```

7.4 String-Split封装：

```
class string_util {
    public:
        static int split(const std::string &in, const std::string &sep, std::vector<std::string> &arry) {
            arry.clear();
            size_t pos, idx = 0;
            while(idx < in.size()) {
                pos = in.find(sep, idx);
                if (pos == std::string::npos) {
                    arry.push_back(in.substr(idx));
                    break;
                }
                if (pos != idx) {
                    arry.push_back(in.substr(idx, pos - idx));
                }
                idx = pos + sep.size();
            }
            return arry.size();
        }
};
```

7.5 File-read封装

```
class file_util {
    public:
        static bool read(const std::string &filename, std::string &body) {
            std::ifstream file;
            // 打开文件
            file.open(filename.c_str(), std::ios::in | std::ios::binary);
            if (!file) {
                std::cout << filename  << " Open failed!" << std::endl;
                return false;
            }
            // 计算文件大小
            file.seekg(0, std::ios::end);
            body.resize(file.tellg());
            file.seekg(0, std::ios::beg);
            file.read(&body[0], body.size());
```

```
            if (file.good() == false) {
                std::cout << filename  << " Read failed!" << std::endl;
                file.close();
                return false;
            }
            file.close();
            return true;
        }
};
```

8. 数据管理模块实现

数据管理模块主要负责对于数据库中数据进行统一的增删改查管理，其他模块要对数据操作都必须通过数据管理模块完成。

8.1 数据库设计

创建user表， 用来表示用户信息及积分信息

- 用户信息， 用来实现登录、注册、游戏对战数据管理等功能
- 积分信息， 用来实现匹配功能

```
create database if not exists online_gobang;

use online_gobang;

create table if not exists user (
    id int primary key auto_increment,
    username varchar(32),
    password varchar(32),
    score int,
    total_count int,
    win_count int
);

-- 构造几个测试用户数据
insert into user values(null, 'xiaobai', MD5('123'), 1000, 0, 0);
insert into user values(null, 'xiaohei', MD5('123'), 1000, 0, 0);
```

验证数据库是否创建成功

```
MariaDB [(none)]> show databases;
+--------------------+
```

```
| Database           |
+--------------------+
| information_schema |
| mysql              |
| online_gobang      |
| performance_schema |
| sys                |
+--------------------+
6 rows in set (0.000 sec)

MariaDB [(none)]> use online_gobang;
Database changed

MariaDB [online_gobang]> show tables;
+------------------------+
| Tables_in_online_gobang |
+------------------------+
| user                   |
+------------------------+
1 row in set (0.000 sec)

MariaDB [online_gobang]> select * from user;
+----+----------+----------------------------------+-------+-------------+-----
------+
| id | username | password                         | score | total_count |
win_count |
+----+----------+----------------------------------+-------+-------------+-----
------+
|  1 | xiaobai  | 202cb962ac59075b964b07152d234b70 |  1030 |           1 |
     1 |
|  2 | xiaohei  | 202cb962ac59075b964b07152d234b70 |   970 |           1 |
     0 |
+----+----------+----------------------------------+-------+-------------+-----
------+
2 rows in set (0.000 sec)

MariaDB [online_gobang]>
```

8.2 创建user_table类

数据库中有可能存在很多张表，每张表中管理的数据又有不同，要进行的数据操作也各不相同，因此我们可以为每一张表中的数据操作都设计一个类，通过类实例化的对象来访问这张数据库表中的数据，这样的话当我们要访问哪张表的时候，使用哪个类实例化的对象即可。

创建user_table类， 该类的作用是负责通过 `MySQL` 接口管理用户数据。主要提供了四个方法：

- select_by_name: 根据用户名查找用户信息， 用于实现登录功能

- insert: 新增用户，用户实现注册功能
- login: 登录验证，并获取完整的用户信息
- win: 用于给获胜玩家修改分数
- lose: 用户给失败玩家修改分数

```
#ifndef __M_DB_H__
#define __M_DB_H__
#include "util.hpp"
#include <mutex>
#include <cassert>

class user_table{
   private:
          MYSQL *_mysql; //mysql操作句柄
          std::mutex _mutex;//互斥锁保护数据库的访问操作
   public:
          user_table(const std::string &host,
               const std::string &username,
               const std::string &password,
               const std::string &dbname,
               uint16_t port = 3306) {
               _mysql = mysql_util::mysql_create(host, username, password, dbname, port);
               assert(_mysql != NULL);
          }
          ~user_table() {
               mysql_util::mysql_destroy(_mysql);
               _mysql = NULL;
          }
          //注册时新增用户
          bool insert(Json::Value &user) {
#define INSERT_USER "insert user values(null, '%s', password('%s'), 1000, 0, 0);"
               // sprintf(void *buf, char *format, ...)
               if (user["password"].isNull() || user["username"].isNull()) {
                    DLOG("INPUT PASSWORD OR USERNAME");
                    return false;
               }
               char sql[4096] = {0};
               sprintf(sql, INSERT_USER, user["username"].asCString(), user["password"].asCString());
               bool ret = mysql_util::mysql_exec(_mysql, sql);
               if (ret == false) {
                    DLOG("insert user info failed!!\n");
```

```
37                      return false;
38                 }
39                 return true;
40            }
41            //登录验证，并返回详细的用户信息
42            bool login(Json::Value &user) {
43                 if (user["password"].isNull() || user["username"].isNull()) {
44                      DLOG("INPUT PASSWORD OR USERNAME");
45                      return false;
46                 }
47                 //以用户名和密码共同作为查询过滤条件，查询到数据则表示用户名密码一致，没有信息则用户名密码错误
48 #define LOGIN_USER "select id, score, total_count, win_count from user where username='%s' and password=password('%s');"
49                 char sql[4096] = {0};
50                 sprintf(sql, LOGIN_USER, user["username"].asCString(), user["password"].asCString());
51                 MYSQL_RES *res = NULL;
52                 {
53                      std::unique_lock<std::mutex> lock(_mutex);
54                      bool ret = mysql_util::mysql_exec(_mysql, sql);
55                      if (ret == false) {
56                           DLOG("user login failed!!\n");
57                           return false;
58                      }
59                      //按理说要么有数据，要么没有数据，就算有数据也只能有一条数据
60                      res = mysql_store_result(_mysql);
61                      if (res == NULL) {
62                           DLOG("have no login user info!!");
63                           return false;
64                      }
65                 }
66                 int row_num = mysql_num_rows(res);
67                 if (row_num != 1) {
68                      DLOG("the user information queried is not unique!!");
69                      return false;
70                 }
71                 MYSQL_ROW row = mysql_fetch_row(res);
72                 user["id"] = (Json::UInt64)std::stol(row[0]);
73                 user["score"] = (Json::UInt64)std::stol(row[1]);
74                 user["total_count"] = std::stoi(row[2]);
75                 user["win_count"] = std::stoi(row[3]);
76                 mysql_free_result(res);
77                 return true;
78            }
79            // 通过用户名获取用户信息
80            bool select_by_name(const std::string &name, Json::Value &user) {
```

```
81 #define USER_BY_NAME "select id, score, total_count, win_count from user where
   username='%s';"
82                 char sql[4096] = {0};
83                 sprintf(sql, USER_BY_NAME, name.c_str());
84                 MYSQL_RES *res = NULL;
85                 {
86                       std::unique_lock<std::mutex> lock(_mutex);
87                       bool ret = mysql_util::mysql_exec(_mysql, sql);
88                       if (ret == false) {
89                             DLOG("get user by name failed!!\n");
90                             return false;
91                       }
92                       //按理说要么有数据，要么没有数据，就算有数据也只能有一条数据
93                       res = mysql_store_result(_mysql);
94                       if (res == NULL) {
95                             DLOG("have no user info!!");
96                             return false;
97                       }
98                 }
99                 int row_num = mysql_num_rows(res);
100                if (row_num != 1) {
101                      DLOG("the user information queried is not unique!!");
102                      return false;
103                }
104                MYSQL_ROW row = mysql_fetch_row(res);
105                user["id"] = (Json::UInt64)std::stol(row[0]);
106                user["username"] = name;
107                user["score"] = (Json::UInt64)std::stol(row[1]);
108                user["total_count"] = std::stoi(row[2]);
109                user["win_count"] = std::stoi(row[3]);
110                mysql_free_result(res);
111                return true;
112          }
113          // 通过用户名获取用户信息
114          bool select_by_id(uint64_t id, Json::Value &user) {
115 #define USER_BY_ID "select username, score, total_count, win_count from user
   where id=%d;"
116                char sql[4096] = {0};
117                sprintf(sql, USER_BY_ID, id);
118                MYSQL_RES *res = NULL;
119                {
120                      std::unique_lock<std::mutex> lock(_mutex);
121                      bool ret = mysql_util::mysql_exec(_mysql, sql);
122                      if (ret == false) {
123                            DLOG("get user by id failed!!\n");
124                            return false;
125                      }
```

```
126                     //按理说要么有数据，要么没有数据，就算有数据也只能有一条数据
127                     res = mysql_store_result(_mysql);
128                     if (res == NULL) {
129                         DLOG("have no user info!!");
130                         return false;
131                     }
132                 }
133                 int row_num = mysql_num_rows(res);
134                 if (row_num != 1) {
135                     DLOG("the user information queried is not unique!!");
136                     return false;
137                 }
138                 MYSQL_ROW row = mysql_fetch_row(res);
139                 user["id"] = (Json::UInt64)id;
140                 user["username"] = row[0];
141                 user["score"] = (Json::UInt64)std::stol(row[1]);
142                 user["total_count"] = std::stoi(row[2]);
143                 user["win_count"] = std::stoi(row[3]);
144                 mysql_free_result(res);
145                 return true;
146             }
147             //胜利时天梯分数增加30分，战斗场次增加1，胜利场次增加1
148             bool win(uint64_t id) {
149 #define USER_WIN "update user set score=score+30, total_count=total_count+1,
    win_count=win_count+1 where id=%d;"
150                 char sql[4096] = {0};
151                 sprintf(sql, USER_WIN, id);
152                 bool ret = mysql_util::mysql_exec(_mysql, sql);
153                 if (ret == false) {
154                     DLOG("update win user info failed!!\n");
155                     return false;
156                 }
157                 return true;
158             }
159             //失败时天梯分数减少30，战斗场次增加1，其他不变
160             bool lose(uint64_t id) {
161 #define USER_LOSE "update user set score=score-30, total_count=total_count+1
    where id=%d;"
162                 char sql[4096] = {0};
163                 sprintf(sql, USER_LOSE, id);
164                 bool ret = mysql_util::mysql_exec(_mysql, sql);
165                 if (ret == false) {
166                     DLOG("update lose user info failed!!\n");
167                     return false;
168                 }
169                 return true;
170             }
```

```
};
#endif
```

9. 在线用户管理模块实现

在线用户管理，是对于当前游戏大厅和游戏房间中的用户进行管理，主要是建立起用户与Socket连接的映射关系，这个模块具有两个功能：

1. 能够让程序中根据用户信息，进而找到能够与用户客户端进行通信的Socket连接，进而实现与客户端的通信。
2. 判断一个用户是否在线，或者判断用户是否已经掉线。

```

class online_manager {
    private:
        /*游戏大厅的客户端连接管理*/
        std::unordered_map<uint64_t, websocket_server::connection_ptr> _game_hall;
        /*游戏房间的客户端连接管理*/
        std::unordered_map<uint64_t, websocket_server::connection_ptr> _game_room;
        std::mutex _mutex;
    public:
        /*进入游戏大厅--游戏大厅连接建立成功后调用*/
        void enter_game_hall(uint64_t uid, const websocket_server::connection_ptr &conn) {
            std::unique_lock<std::mutex> lock(_mutex);
            _game_hall.insert(std::make_pair(uid, conn));
        }
        /*退出游戏大厅--游戏大厅连接断开后调用*/
        void exit_game_hall(uint64_t uid) {
            std::unique_lock<std::mutex> lock(_mutex);
            _game_hall.erase(uid);
        }
        /*进入游戏房间--游戏房间连接建立成功后调用*/
        void enter_game_room(uint64_t uid, const websocket_server::connection_ptr &conn) {
            std::unique_lock<std::mutex> lock(_mutex);
            _game_room.insert(std::make_pair(uid, conn));
        }
        /*退出游戏房间--游戏房间连接断开后调用*/
        void exit_game_room(uint64_t uid) {
            std::unique_lock<std::mutex> lock(_mutex);
            _game_room.erase(uid);
        }
```

```
30          /*判断用户是否在游戏大厅*/
31          bool in_game_hall(uint64_t uid) {
32              std::unique_lock<std::mutex> lock(_mutex);
33              auto it = _game_hall.find(uid);
34              if (it == _game_hall.end()) {
35                  return false;
36              }
37              return true;
38          }
39          /*判断用户是否在游戏房间*/
40          bool in_game_room(uint64_t uid) {
41              std::unique_lock<std::mutex> lock(_mutex);
42              auto it = _game_room.find(uid);
43              if (it == _game_room.end()) {
44                  return false;
45              }
46              return true;
47          }
48          /*从游戏大厅中获取指定用户关联的Socket连接*/
49          bool get_conn_from_game_hall(uint64_t uid,
    websocket_server::connection_ptr &conn) {
50              std::unique_lock<std::mutex> lock(_mutex);
51              auto it = _game_hall.find(uid);
52              if (it == _game_hall.end()) {
53                  return false;
54              }
55              conn = it->second;
56              return true;
57          }
58          /*从游戏房间中获取指定用户关联的Socket连接*/
59          bool get_conn_from_game_room(uint64_t uid,
    websocket_server::connection_ptr &conn) {
60              std::unique_lock<std::mutex> lock(_mutex);
61              auto it = _game_room.find(uid);
62              if (it == _game_room.end()) {
63                  return false;
64              }
65              conn = it->second;
66              return true;
67          }
68  };
```

10. 游戏房间管理模块

10.1 房间类实现

首先，需要设计一个房间类，能够实现房间的实例化，房间类主要是对匹配成对的玩家建立一个小范围的关联关系，一个房间中任意一个用户发生的任何动作，都会被广播给房间中的其他用户。

而房间中的动作主要包含两类：

1. 棋局对战
2. 实时聊天

```
1
2  #define BOARD_ROW 15
3  #define BOARD_COL 15
4  #define CHESS_WHITE 1
5  #define CHESS_BLACK 2
6  typedef enum { GAME_START, GAME_OVER }room_statu;
7  class room {
8      private:
9          uint64_t _room_id;
10         room_statu _statu;
11         int _player_count;
12         uint64_t _white_id;
13         uint64_t _black_id;
14         user_table *_tb_user;
15         online_manager *_online_user;
16         std::vector<std::vector<int>> _board;
17     private:
18         bool five(int row, int col, int row_off, int col_off, int color) {
19             //row和col是下棋位置，  row_off和col_off是偏移量，也是方向
20             int count = 1;
21             int search_row = row + row_off;
22             int search_col = col + col_off;
23             while(search_row >= 0 && search_row < BOARD_ROW &&
24                   search_col >= 0 && search_col < BOARD_COL &&
25                   _board[search_row][search_col] == color) {
26                 //同色棋子数量++
27                 count++;
28                 //检索位置继续向后偏移
29                 search_row += row_off;
30                 search_col += col_off;
31             }
32             search_row = row - row_off;
33             search_col = col - col_off;
34             while(search_row >= 0 && search_row < BOARD_ROW &&
35                   search_col >= 0 && search_col < BOARD_COL &&
36                   _board[search_row][search_col] == color) {
37                 //同色棋子数量++
38                 count++;
```

```
39                  //检索位置继续向后偏移
40                  search_row -= row_off;
41                  search_col -= col_off;
42              }
43              return (count >= 5);
44          }
45          uint64_t check_win(int row, int col, int color) {
46              // 从下棋位置的四个不同方向上检测是否出现了5个及以上相同颜色的棋子（横行，纵列，正斜，反斜）
47              if (five(row, col, 0, 1, color) ||
48                  five(row, col, 1, 0, color) ||
49                  five(row, col, -1, 1, color)||
50                  five(row, col, -1, -1, color)) {
51                  //任意一个方向上出现了true也就是五星连珠，则设置返回值
52                  return color == CHESS_WHITE ? _white_id : _black_id;
53              }
54              return 0;
55          }
56      public:
57          room(uint64_t room_id, user_table *tb_user, online_manager *online_user):
58              _room_id(room_id), _statu(GAME_START), _player_count(0),
59              _tb_user(tb_user), _online_user(online_user),
60              _board(BOARD_ROW, std::vector<int>(BOARD_COL, 0)){
61              DLOG("%lu 房间创建成功!!", _room_id);
62          }
63          ~room() {
64              DLOG("%lu 房间销毁成功!!", _room_id);
65          }
66          uint64_t id() { return _room_id; }
67          room_statu statu() { return _statu; }
68          int player_count() { return _player_count; }
69          void add_white_user(uint64_t uid) { _white_id = uid; _player_count++; }
70          void add_black_user(uint64_t uid) { _black_id = uid; _player_count++; }
71          uint64_t get_white_user() { return _white_id; }
72          uint64_t get_black_user() { return _black_id; }
73
74          /*处理下棋动作*/
75          Json::Value handle_chess(Json::Value &req) {
76              Json::Value json_resp = req;
77              // 2. 判断房间中两个玩家是否都在线，任意一个不在线，就是另一方胜利。
78              int chess_row = req["row"].asInt();
79              int chess_col = req["col"].asInt();
80              uint64_t cur_uid = req["uid"].asUInt64();
81              if (_online_user->is_in_game_room(_white_id) == false) {
82                  json_resp["result"] = true;
83                  json_resp["reason"] = "运气真好! 对方掉线，不战而胜! ";
```

```
84                  json_resp["winner"] = (Json::UInt64)_black_id;
85                  return json_resp;
86              }
87              if (_online_user->is_in_game_room(_black_id) == false) {
88                  json_resp["result"] = true;
89                  json_resp["reason"] = "运气真好！对方掉线，不战而胜！";
90                  json_resp["winner"] = (Json::UInt64)_white_id;
91                  return json_resp;
92              }
93              // 3. 获取走棋位置，判断当前走棋是否合理（位置是否已经被占用）
94              if (_board[chess_row][chess_col] != 0) {
95                  json_resp["result"] = false;
96                  json_resp["reason"] = "当前位置已经有了其他棋子！";
97                  return json_resp;
98              }
99              int cur_color = cur_uid == _white_id ? CHESS_WHITE : CHESS_BLACK;
100             _board[chess_row][chess_col] = cur_color;
101             // 4. 判断是否有玩家胜利（从当前走棋位置开始判断是否存在五星连珠）
102             uint64_t winner_id = check_win(chess_row, chess_col, cur_color);
103             if (winner_id != 0) {
104                 json_resp["reason"] = "五星连珠，战无敌！";
105             }
106             json_resp["result"] = true;
107             json_resp["winner"] = (Json::UInt64)winner_id;
108             return json_resp;
109         }
110         /*处理聊天动作*/
111         Json::Value handle_chat(Json::Value &req) {
112             Json::Value json_resp = req;
113             //检测消息中是否包含敏感词
114             std::string msg = req["message"].asString();
115             size_t pos = msg.find("垃圾");
116             if (pos != std::string::npos) {
117                 json_resp["result"] = false;
118                 json_resp["reason"] = "消息中包含敏感词，不能发送！";
119                 return json_resp;
120             }
121             //广播消息---返回消息
122             json_resp["result"] = true;
123             return json_resp;
124         }
125         /*处理玩家退出房间动作*/
126         void handle_exit(uint64_t uid) {
127             //如果是下棋中退出，则对方胜利，否则下棋结束了退出，则是正常退出
128             Json::Value json_resp;
129             if (_statu == GAME_START) {
```

```
130                 uint64_t winner_id = (Json::UInt64)(uid == _white_id ?
    _black_id : _white_id);
131                 json_resp["optype"] = "put_chess";
132                 json_resp["result"] = true;
133                 json_resp["reason"] = "对方掉线，不战而胜！";
134                 json_resp["room_id"] = (Json::UInt64)_room_id;
135                 json_resp["uid"] = (Json::UInt64)uid;
136                 json_resp["row"] = -1;
137                 json_resp["col"] = -1;
138                 json_resp["winner"] = (Json::UInt64)winner_id;
139                 uint64_t loser_id = winner_id == _white_id ? _black_id :
    _white_id;
140                 _tb_user->win(winner_id);
141                 _tb_user->lose(loser_id);
142                 _statu = GAME_OVER;
143                 broadcast(json_resp);
144             }
145             //房间中玩家数量--
146             _player_count--;
147             return;
148         }
149         /*总的请求处理函数，在函数内部，区分请求类型，根据不同的请求调用不同的处理函数，
    得到响应进行广播*/
150         void handle_request(Json::Value &req) {
151             //1. 校验房间号是否匹配
152             Json::Value json_resp;
153             uint64_t room_id = req["room_id"].asUInt64();
154             if (room_id != _room_id) {
155                 json_resp["optype"] = req["optype"].asString();
156                 json_resp["result"] = false;
157                 json_resp["reason"] = "房间号不匹配！";
158                 return broadcast(json_resp);
159             }
160             //2. 根据不同的请求类型调用不同的处理函数
161             if (req["optype"].asString() == "put_chess") {
162                 json_resp = handle_chess(req);
163                 if (json_resp["winner"].asUInt64() != 0) {
164                     uint64_t winner_id = json_resp["winner"].asUInt64();
165                     uint64_t loser_id = winner_id == _white_id ? _black_id :
    _white_id;
166                     _tb_user->win(winner_id);
167                     _tb_user->lose(loser_id);
168                     _statu = GAME_OVER;
169                 }
170             }else if (req["optype"].asString() == "chat") {
171                 json_resp = handle_chat(req);
172             }else {
```

```
                json_resp["optype"] = req["optype"].asString();
                json_resp["result"] = false;
                json_resp["reason"] = "未知请求类型";
            }
            std::string body;
            json_util::serialize(json_resp, body);
            DLOG("房间-广播动作: %s", body.c_str());
            return broadcast(json_resp);
        }
        /*将指定的信息广播给房间中所有玩家*/
        void broadcast(Json::Value &rsp) {
            //1. 对要响应的信息进行序列化，将Json::Value中的数据序列化成为json格式字符串
            std::string body;
            json_util::serialize(rsp, body);
            //2. 获取房间中所有用户的通信连接
            //3. 发送响应信息
            wsserver_t::connection_ptr wconn = _online_user->get_conn_from_room(_white_id);
            if (wconn.get() != nullptr) {
                wconn->send(body);
            }else {
                DLOG("房间-白棋玩家连接获取失败");
            }
            wsserver_t::connection_ptr bconn = _online_user->get_conn_from_room(_black_id);
            if (bconn.get() != nullptr) {
                bconn->send(body);
            }else {
                DLOG("房间-黑棋玩家连接获取失败");
            }
            return;
        }
};
```

10.2 房间管理类实现

实现对所有的游戏房间进行管理。

```
#ifndef __M_ROOM_H__
#define __M_ROOM_H__
#include "util.hpp"
#include "logger.hpp"
#include "online.hpp"
#include "db.hpp"
```

```
7
8  using room_ptr = std::shared_ptr<room>;
9  class room_manager{
10     private:
11         uint64_t _next_rid;
12         std::mutex _mutex;
13         user_table *_tb_user;
14         online_manager *_online_user;
15         std::unordered_map<uint64_t, room_ptr> _rooms;
16         std::unordered_map<uint64_t, uint64_t> _users;
17     public:
18         /*初始化房间ID计数器*/
19         room_manager(user_table *ut, online_manager *om):
20             _next_rid(1), _tb_user(ut), _online_user(om) {
21             DLOG("房间管理模块初始化完毕！");
22         }
23         ~room_manager() { DLOG("房间管理模块即将销毁！"); }
24         //为两个用户创建房间，并返回房间的智能指针管理对象
25         room_ptr create_room(uint64_t uid1, uint64_t uid2) {
26             //两个用户在游戏大厅中进行对战匹配，匹配成功后创建房间
27             //1. 校验两个用户是否都还在游戏大厅中，只有都在才需要创建房间。
28             if (_online_user->is_in_game_hall(uid1) == false) {
29                 DLOG("用户：%lu 不在大厅中，创建房间失败!", uid1);
30                 return room_ptr();
31             }
32             if (_online_user->is_in_game_hall(uid2) == false) {
33                 DLOG("用户：%lu 不在大厅中，创建房间失败!", uid2);
34                 return room_ptr();
35             }
36             //2. 创建房间，将用户信息添加到房间中
37
38             std::unique_lock<std::mutex> lock(_mutex);
39             room_ptr rp(new room(_next_rid, _tb_user, _online_user));
40             rp->add_white_user(uid1);
41             rp->add_black_user(uid2);
42             //3. 将房间信息管理起来
43             _rooms.insert(std::make_pair(_next_rid, rp));
44             _users.insert(std::make_pair(uid1, _next_rid));
45             _users.insert(std::make_pair(uid2, _next_rid));
46             _next_rid++;
47             //4. 返回房间信息
48             return rp;
49         }
50         /*通过房间ID获取房间信息*/
51         room_ptr get_room_by_rid(uint64_t rid) {
52             std::unique_lock<std::mutex> lock(_mutex);
53             auto it = _rooms.find(rid);
```

```
54              if (it == _rooms.end()) {
55                  return room_ptr();
56              }
57              return it->second;
58          }
59          /*通过用户ID获取房间信息*/
60          room_ptr get_room_by_uid(uint64_t uid) {
61              std::unique_lock<std::mutex> lock(_mutex);
62              //1. 通过用户ID获取房间ID
63              auto uit = _users.find(uid);
64              if (uit == _users.end()) {
65                  return room_ptr();
66              }
67              uint64_t rid = uit->second;
68              //2. 通过房间ID获取房间信息
69              auto rit = _rooms.find(rid);
70              if (rit == _rooms.end()) {
71                  return room_ptr();
72              }
73              return rit->second;
74          }
75          /*通过房间ID销毁房间*/
76          void remove_room(uint64_t rid) {
77              //因为房间信息，是通过shared_ptr在_rooms中进行管理，因此只要将shared_ptr
    从_rooms中移除
78              //则shared_ptr计数器==0，外界没有对房间信息进行操作保存的情况下就会释放
79              //1. 通过房间ID，获取房间信息
80              room_ptr rp = get_room_by_rid(rid);
81              if (rp.get() == nullptr) {
82                  return;
83              }
84              //2. 通过房间信息，获取房间中所有用户的ID
85              uint64_t uid1 = rp->get_white_user();
86              uint64_t uid2 = rp->get_black_user();
87              //3. 移除房间管理中的用户信息
88              std::unique_lock<std::mutex> lock(_mutex);
89              _users.erase(uid1);
90              _users.erase(uid2);
91              //4. 移除房间管理信息
92              _rooms.erase(rid);
93          }
94          /*删除房间中指定用户，如果房间中没有用户了，则销毁房间，用户连接断开时被调用*/
95          void remove_room_user(uint64_t uid) {
96              room_ptr rp = get_room_by_uid(uid);
97              if (rp.get() == nullptr) {
98                  return;
99              }
```

```
100             //处理房间中玩家退出动作
101             rp->handle_exit(uid);
102             //房间中没有玩家了，则销毁房间
103             if (rp->player_count() == 0) {
104                 remove_room(rp->id());
105             }
106             return ;
107         }
108 };
109
110 #endif
```

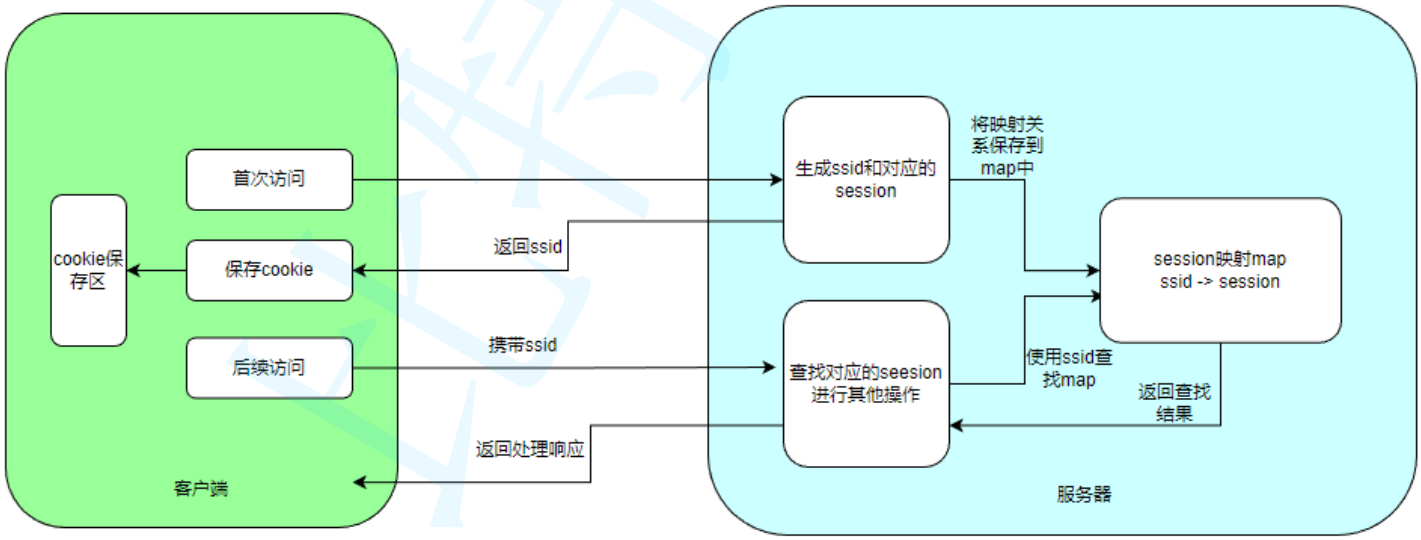
11. session管理模块设计

11.1 什么是session

在WEB开发中，HTTP协议是一种无状态短链接的协议，这就导致一个客户端连接到服务器上之后，服务器不知道当前的连接对应的是哪个用户，也不知道客户端是否登录成功，这时候为客户端提所有服务是不合理的。

因此，服务器为每个用户浏览器创建一个会话对象（session对象），注意：一个浏览器独占一个session对象(默认情况下)。因此，在需要保存用户数据时，服务器程序可以把用户数据写到用户浏览器独占的session中，当用户使用浏览器访问其它程序时，其它程序可以从用户的session中取出该用户的数据，识别该连接对应的用户，并为用户提供服务。

11.2 session工作原理



11.3 session类设计实现

- 这里我们简单的设计一个session类，但是session对象不能一直存在，这样是一种资源泄漏，因此需要使用定时器对每个创建的session对象进行定时销毁（一个客户端连接断开后，一段时间内都没有重新连接则销毁session）。

- _ssid使用时间戳填充。实际上， 我们通常使用唯一id生成器生成一个唯一的id
- _user保存当前用户的信息
- timer_ptr _tp保存当前session对应的定时销毁任务

```
typedef enum {UNLOGIN, LOGIN} ss_statu;
class session {
    private:
        uint64_t _ssid;//标识符
        uint64_t _uid;//session对应的用户ID
        ss_statu _statu;//用户状态：未登录，已登录
        wsserver_t::timer_ptr _tp;//session关联的定时器
    public:
        session(uint64_t ssid): _ssid(ssid){ DLOG("SESSION %p 被创建！！", this); }
        ~session() { DLOG("SESSION %p 被释放！！", this); }
        uint64_t ssid() { return _ssid; }
        void set_statu(ss_statu statu) { _statu = statu; }
        void set_user(uint64_t uid) { _uid = uid; }
        uint64_t get_user() { return _uid; }
        bool is_login() { return (_statu == LOGIN); }
        void set_timer(const wsserver_t::timer_ptr &tp) { _tp = tp;}
        wsserver_t::timer_ptr& get_timer() { return _tp; }
};
```

11.4 session管理设计实现

session的管理主要包含以下几个点：

1. 创建一个新的session
2. 通过ssid获取session
3. 通过ssid判断session是否存在
4. 销毁session。
5. 为session设置过期时间，过期后session被销毁

```
#ifndef __M_SS_H__
#define __M_SS_H__
#include "util.hpp"
#include <unordered_map>
#include <websocketpp/server.hpp>
#include <websocketpp/config/asio_no_tls.hpp>

```

```
 8 #define SESSION_TIMEOUT 30000
 9 #define SESSION_FOREVER -1
10 using session_ptr = std::shared_ptr<session>;
11 class session_manager {
12     private:
13         uint64_t _next_ssid;
14         std::mutex _mutex;
15         std::unordered_map<uint64_t, session_ptr> _session;
16         wsserver_t *_server;
17     public:
18         session_manager(wsserver_t *srv): _next_ssid(1), _server(srv){
19             DLOG("session管理器初始化完毕! ");
20         }
21         ~session_manager() { DLOG("session管理器即将销毁! "); }
22         session_ptr create_session(uint64_t uid, ss_statu statu) {
23             std::unique_lock<std::mutex> lock(_mutex);
24             session_ptr ssp(new session(_next_ssid));
25             ssp->set_statu(statu);
26             ssp->set_user(uid);
27             _session.insert(std::make_pair(_next_ssid, ssp));
28             _next_ssid++;
29             return ssp;
30         }
31         void append_session(const session_ptr &ssp) {
32             std::unique_lock<std::mutex> lock(_mutex);
33             _session.insert(std::make_pair(ssp->ssid(), ssp));
34         }
35         session_ptr get_session_by_ssid(uint64_t ssid) {
36             std::unique_lock<std::mutex> lock(_mutex);
37             auto it = _session.find(ssid);
38             if (it == _session.end()) {
39                 return session_ptr();
40             }
41             return it->second;
42         }
43         void remove_session(uint64_t ssid) {
44             std::unique_lock<std::mutex> lock(_mutex);
45             _session.erase(ssid);
46         }
47         void set_session_expire_time(uint64_t ssid, int ms) {
48             //依赖于websocketpp的定时器来完成session生命周期的管理。
49             // 登录之后，创建session，session需要在指定时间无通信后删除
50             // 但是进入游戏大厅，或者游戏房间，这个session就应该永久存在
51             // 等到退出游戏大厅，或者游戏房间，这个session应该被重新设置为临时，在长时间
   无通信后被删除
52             session_ptr ssp = get_session_by_ssid(ssid);
53             if (ssp.get() == nullptr) {
```

```
54                  return;
55              }
56              wsserver_t::timer_ptr tp = ssp->get_timer();
57              if (tp.get() == nullptr && ms == SESSION_FOREVER) {
58                  // 1. 在session永久存在的情况下，设置永久存在
59                  return ;
60              }else if (tp.get() == nullptr && ms != SESSION_FOREVER) {
61                  // 2. 在session永久存在的情况下，设置指定时间之后被删除的定时任务
62                  wsserver_t::timer_ptr tmp_tp = _server->set_timer(ms,
63                      std::bind(&session_manager::remove_session, this, ssid));
64                  ssp->set_timer(tmp_tp);
65              }else if (tp.get() != nullptr && ms == SESSION_FOREVER) {
66                  // 3. 在session设置了定时删除的情况下，将session设置为永久存在
67                  // 删除定时任务--- stready_timer删除定时任务会导致任务直接被执行
68                  tp->cancel();//因为这个取消定时任务并不是立即取消的
69                  //因此重新给session管理器中，添加一个session信息，且添加的时候需要使用
    定时器，而不是立即添加
70                  ssp->set_timer(wsserver_t::timer_ptr());//将session关联的定时器设
    置为空
71                  _server->set_timer(0,
    std::bind(&session_manager::append_session, this, ssp));
72              }else if (tp.get() != nullptr && ms != SESSION_FOREVER) {
73                  // 4. 在session设置了定时删除的情况下，将session重置删除时间。
74                  tp->cancel();//因为这个取消定时任务并不是立即取消的
75                  ssp->set_timer(wsserver_t::timer_ptr());
76                  _server->set_timer(0,
    std::bind(&session_manager::append_session, this, ssp));
77
78                  //重新给session添加定时销毁任务
79                  wsserver_t::timer_ptr tmp_tp  = _server->set_timer(ms,
80                      std::bind(&session_manager::remove_session, this, ssp-
    >ssid()));
81                  //重新设置session关联的定时器
82                  ssp->set_timer(tmp_tp);
83              }
84          }
85  };
86
87
88  #endif
```

12. 五子棋对战玩家匹配管理设计实现

12.1 匹配队列实现

五子棋对战的玩家匹配是根据自己的天梯分数进行匹配的，而服务器中将玩家天梯分数分为三个档次：

1. 青铜：天梯分数小于2000分
2. 白银：天梯分数介于2000~3000分之间
3. 黄金：天梯分数大于3000分

而实现玩家匹配的思想非常简单，为不同的档次设计各自的匹配队列，当一个队列中的玩家数量大于等于2的时候，则意味着同一档次中，有2个及以上的人要进行实战匹配，则出队队列中的前两个用户，相当于队首2个玩家匹配成功，这时候为其创建房间，并将两个用户信息加入房间中。

```
template <class T>
class match_queue {
    private:
        /*用链表而不直接使用queue是因为我们有中间删除数据的需要*/
        std::list<T> _list;
        /*实现线程安全*/
        std::mutex _mutex;
        /*这个条件变量主要为了阻塞消费者，后边使用的时候：队列中元素个数<2则阻塞*/
        std::condition_variable _cond;
    public:
        /*获取元素个数*/
        int size() {
            std::unique_lock<std::mutex> lock(_mutex);
            return _list.size();
        }
        /*判断是否为空*/
        bool empty() {
            std::unique_lock<std::mutex> lock(_mutex);
            return _list.empty();
        }
        /*阻塞线程*/
        void wait() {
            std::unique_lock<std::mutex> lock(_mutex);
            _cond.wait(lock);
        }
        /*入队数据，并唤醒线程*/
        void push(const T &data) {
            std::unique_lock<std::mutex> lock(_mutex);
            _list.push_back(data);
            _cond.notify_all();
        }
        /*出队数据*/
        bool pop(T &data) {
            std::unique_lock<std::mutex> lock(_mutex);
```

```
            if (_list.empty() == true) {
                return false;
            }
            data = _list.front();
            _list.pop_front();
            return true;
        }
        /*移除指定的数据*/
        void remove(T &data) {
            std::unique_lock<std::mutex> lock(_mutex);
            _list.remove(data);
        }
};
```

12.2 玩家匹配管理模块设计实现

```
#ifndef __M_MATCHER_H__
#define __M_MATCHER_H__

#include "util.hpp"
#include "online.hpp"
#include "db.hpp"
#include "room.hpp"
#include <list>
#include <mutex>
#include <condition_variable>
template <class T>
class match_queue {
    private:
        /*用链表而不直接使用queue是因为我们有中间删除数据的需要*/
        std::list<T> _list;
        /*实现线程安全*/
        std::mutex _mutex;
        /*这个条件变量主要为了阻塞消费者，后边使用的时候：队列中元素个数<2则阻塞*/
        std::condition_variable _cond;
    public:
        /*获取元素个数*/
        int size() {
            std::unique_lock<std::mutex> lock(_mutex);
            return _list.size();
        }
        /*判断是否为空*/
        bool empty() {
```

```
28              std::unique_lock<std::mutex> lock(_mutex);
29              return _list.empty();
30          }
31          /*阻塞线程*/
32          void wait() {
33              std::unique_lock<std::mutex> lock(_mutex);
34              _cond.wait(lock);
35          }
36          /*入队数据，并唤醒线程*/
37          void push(const T &data) {
38              std::unique_lock<std::mutex> lock(_mutex);
39              _list.push_back(data);
40              _cond.notify_all();
41          }
42          /*出队数据*/
43          bool pop(T &data) {
44              std::unique_lock<std::mutex> lock(_mutex);
45              if (_list.empty() == true) {
46                  return false;
47              }
48              data = _list.front();
49              _list.pop_front();
50              return true;
51          }
52          /*移除指定的数据*/
53          void remove(T &data) {
54              std::unique_lock<std::mutex> lock(_mutex);
55              _list.remove(data);
56          }
57  };
58
59  class matcher {
60      private:
61          /*普通选手匹配队列*/
62          match_queue<uint64_t> _q_normal;
63          /*高手匹配队列*/
64          match_queue<uint64_t> _q_high;
65          /*大神匹配队列*/
66          match_queue<uint64_t> _q_super;
67          /*对应三个匹配队列的处理线程*/
68          std::thread _th_normal;
69          std::thread _th_high;
70          std::thread _th_super;
71          room_manager *_rm;
72          user_table *_ut;
73          online_manager *_om;
74      private:
```

```
75          void handle_match(match_queue<uint64_t> &mq) {
76              while(1) {
77                  //1. 判断队列人数是否大于2，<2则阻塞等待
78                  while (mq.size() < 2) {
79                      mq.wait();
80                  }
81                  //2. 走下来代表人数够了，出队两个玩家
82                  uint64_t uid1, uid2;
83                  bool ret = mq.pop(uid1);
84                  if (ret == false) {
85                      continue;
86                  }
87                  ret = mq.pop(uid2);
88                  if (ret == false) {
89                      this->add(uid1);
90                      continue;
91                  }
92                  //3. 校验两个玩家是否在线，如果有人掉线，则要吧另一个人重新添加入队列
93                  wsserver_t::connection_ptr conn1 = _om-
    >get_conn_from_hall(uid1);
94                  if (conn1.get() == nullptr) {
95                      this->add(uid2);
96                      continue;
97                  }
98                  wsserver_t::connection_ptr conn2 = _om-
    >get_conn_from_hall(uid2);
99                  if (conn2.get() == nullptr) {
100                     this->add(uid1);
101                     continue;
102                 }
103                 //4. 为两个玩家创建房间，并将玩家加入房间中
104                 room_ptr rp = _rm->create_room(uid1, uid2);
105                 if (rp.get() == nullptr) {
106                     this->add(uid1);
107                     this->add(uid2);
108                     continue;
109                 }
110                 //5. 对两个玩家进行响应
111                 Json::Value resp;
112                 resp["optype"] = "match_success";
113                 resp["result"] = true;
114                 std::string body;
115                 json_util::serialize(resp, body);
116                 conn1->send(body);
117                 conn2->send(body);
118             }
119         }
```

```
120         void th_normal_entry() { return handle_match(_q_normal); }
121         void th_high_entry() { return handle_match(_q_high); }
122         void th_super_entry() { return handle_match(_q_super); }
123     public:
124         matcher(room_manager *rm, user_table *ut, online_manager *om):
125             _rm(rm), _ut(ut), _om(om),
126             _th_normal(std::thread(&matcher::th_normal_entry, this)),
127             _th_high(std::thread(&matcher::th_high_entry, this)),
128             _th_super(std::thread(&matcher::th_super_entry, this)){
129             DLOG("游戏匹配模块初始化完毕....");
130         }
131         bool add(uint64_t uid) {
132             //根据玩家的天梯分数，来判定玩家档次，添加到不同的匹配队列
133             // 1. 根据用户ID，获取玩家信息
134             Json::Value user;
135             bool ret = _ut->select_by_id(uid, user);
136             if (ret == false) {
137                 DLOG("获取玩家:%d 信息失败！！", uid);
138                 return false;
139             }
140             int score = user["score"].asInt();
141             // 2. 添加到指定的队列中
142             if (score < 2000) {
143                 _q_normal.push(uid);
144             }else if (score >= 2000 && score < 3000) {
145                 _q_high.push(uid);
146             }else {
147                 _q_super.push(uid);
148             }
149             return true;
150         }
151         bool del(uint64_t uid) {
152             Json::Value user;
153             bool ret = _ut->select_by_id(uid, user);
154             if (ret == false) {
155                 DLOG("获取玩家:%d 信息失败！！", uid);
156                 return false;
157             }
158             int score = user["score"].asInt();
159             // 2. 添加到指定的队列中
160             if (score < 2000) {
161                 _q_normal.remove(uid);
162             }else if (score >= 2000 && score < 3000) {
163                 _q_high.remove(uid);
164             }else {
165                 _q_super.remove(uid);
166             }
```

```
167             return true;
168         }
169 };
170 #endif
```

13. 整合封装服务器模块设计实现

服务器模块，是对当前所实现的所有模块的一个整合，并进行服务器搭建的一个模块，最终封装实现出一个gobang_server的服务器模块类，向外提供搭建五子棋对战服务器的接口。通过实例化的对象可以简便的完成服务器的搭建。

13.1 通信接口设计（Restful风格）

13.1.1 静态资源请求

```
静态资源页面，在后台服务器上就是个html/css/js文件
静态资源请求的处理，其实就是将文件中的内容发送给客户端

1. 注册页面请求
请求：GET /register.html HTTP/1.1
响应：
HTTP/1.1 200 OK
Content-Length: xxx
Content-Type: text/html

register.html文件的内容数据

2. 登录页面请求
请求：GET /login.html HTTP/1.1
3. 大厅页面请求
请求：GET /game_hall.html HTTP/1.1
4. 房间页面请求
请求：GET /game_room.html HTTP/1.1
```

13.1.2 注册用户

```
POST /reg HTTP/1.1
Content-Type: application/json
Content-Length: 32

{"username":"xiaobai", "password":"123123"}
```

```
#成功时的响应
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 15

{"result":true}

#失败时的响应
HTTP/1.1 400 Bad Request
Content-Type: application/json
Content-Length: 43

{"result":false, "reason": "用户名已经被占用"}
```

13.1.3 用户登录

```
POST /login HTTP/1.1
Content-Type: application/json
Content-Length: 32

{"username":"xiaobai", "password":"123123"}
```

```
#成功时的响应
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 15

{"result":true}

#失败时的响应
HTTP/1.1 400 Bad Request
Content-Type: application/json
Content-Length: 43

{"result":false, "reason": "用户名或密码错误"}
```

13.1.4 获取客户端信息

```
GET /userinfo HTTP/1.1
Content-Type: application/json
Content-Length: 0
```

```
#成功时的响应
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 58

{"id":1, "username":"xiaobai", "score":1000, "total_count":4, "win_count":2}


#失败时的响应
HTTP/1.1 401 Unauthorized
Content-Type: application/json
Content-Length: 43

{"result":false, "reason": "用户还未登录"}
```

13.1.5 websocket长连接协议切换请求（进入游戏大厅）

```
/* ws://localhost:9000/match */
GET /match HTTP/1.1
Connection: Upgrade
Upgrade: WebSocket
......
```

```
HTTP/1.1 101 Switching
......
```

WebSocket握手成功后的回复：表示游戏大厅已经进入成功。

```
{
    "optype": "hall_ready",
    "uid": 1
}
```

13.1.6 开始对战匹配

```
{
    "optype": "match_start"
}
```

```
/*后台正确处理后回复*/
{
    "optype": "match_start", //表示成功加入匹配队列
    "result": true
}
/*后台处理出错回复*/
{
    "optype": "match_start"
    "result": false,
    "reason": "具体原因...."
}
```

```
/*匹配成功了给客户端的回复*/
{
    "optype": "match_success", //表示成匹配成功
    "result": true
}
```

13.1.7 停止匹配

```
{
    "optype": "match_stop"
}
```

```
/*后台正确处理后回复*/
{
    "optype": "match_stop"
    "result": true
}
/*后台处理出错回复*/
{
    "optype": "match_stop"
    "result": false,
    "reason": "具体原因...."
```

```
}
```

13.1.8 websocket长连接协议切换请求（进入游戏房间）

```
/* ws://localhost:9000/game */
GET /game HTTP/1.1
Connection: Upgrade
Upgrade: WebSocket
......
```

```
HTTP/1.1 101 Switching
......
```

WebSocket握手成功后的回复：表示游戏房间已经进入成功。

```
/*协议切换成功， 房间已经建立*/
{
    "optype": "room_ready",
    "room_id": 222,   //房间ID
    "self_id": 1,     //自身ID
    "white_id": 1,    //白棋ID
    "black_id": 2,    //黑棋ID
}
```

13.1.9 走棋

```
{
    "optype": "put_chess",  // put_chess表示当前请求是下棋操作
    "room_id": 222,         // room_id  表示当前动作属于哪个房间
    "uid": 1,               // 当前的下棋操作是哪个用户发起的
    "row": 3,               // 当前下棋位置的行号
    "col": 2                // 当前下棋位置的列号
}
```

```
{
    "optype": "put_chess",
    "result": false
```

```
    "reason": "走棋失败具体原因...."
}
{
    "optype": "put_chess",
    "result": true,
    "reason": "对方掉线，不战而胜！" / "对方/己方五星连珠，战无敌/虽败犹荣！",
    "room_id": 222,
    "uid": 1,
    "row": 3,
    "col": 2,
    "winner": 0  // 0-未分胜负，  !0-已分胜负（uid是谁，谁就赢了）
}
```

13.1.10 聊天

```
{
    "optype": "chat",
    "room_id": 222,
    "uid": 1,
    "message": "赶紧点"
}
```

```
{
    "optype": "chat",
    "result": false
    "reason": "聊天失败具体原因....比如有敏感词..."
}
{
    "optype": "chat",
    "result": true,
    "room_id": 222,
    "uid": 1,
    "message": "赶紧点"
}
```

13.2 服务器模块实现

```
#ifndef __M_SRV_H__
#define __M_SRV_H__
#include "db.hpp"
#include "matcher.hpp"
```

```cpp
#include "online.hpp"
#include "room.hpp"
#include "session.hpp"
#include "util.hpp"

#define WWWROOT "./wwwroot/"
class gobang_server{
    private:
        std::string _web_root;//静态资源根目录 ./wwwroot/      /register.html -> ./wwwroot/register.html
        wsserver_t _wssrv;
        user_table _ut;
        online_manager _om;
        room_manager _rm;
        matcher _mm;
        session_manager _sm;
    private:
        void file_handler(wsserver_t::connection_ptr &conn) {
            //静态资源请求的处理
            //1. 获取到请求uri-资源路径，了解客户端请求的页面文件名称
            websocketpp::http::parser::request req = conn->get_request();
            std::string uri = req.get_uri();
            //2. 组合出文件的实际路径   相对根目录 + uri
            std::string realpath = _web_root + uri;
            //3. 如果请求的是个目录，增加一个后缀  login.html,    /  -> /login.html
            if (realpath.back() == '/') {
                realpath += "login.html";
            }
            //4. 读取文件内容
            Json::Value resp_json;
            std::string body;
            bool ret = file_util::read(realpath, body);
            //  1. 文件不存在，读取文件内容失败，返回404
            if (ret == false) {
                body += "<html>";
                body += "<head>";
                body += "<meta charset='UTF-8'/>";
                body += "</head>";
                body += "<body>";
                body += "<h1> Not Found </h1>";
                body += "</body>";
                conn->set_status(websocketpp::http::status_code::not_found);
                conn->set_body(body);
                return;
            }
            //5. 设置响应正文
```

```
50              conn->set_body(body);
51              conn->set_status(websocketpp::http::status_code::ok);
52          }
53          void http_resp(wsserver_t::connection_ptr &conn, bool result,
54              websocketpp::http::status_code::value code, const std::string
    &reason) {
55              Json::Value resp_json;
56              resp_json["result"] = result;
57              resp_json["reason"] = reason;
58              std::string resp_body;
59              json_util::serialize(resp_json, resp_body);
60              conn->set_status(code);
61              conn->set_body(resp_body);
62              conn->append_header("Content-Type", "application/json");
63              return;
64          }
65          void reg(wsserver_t::connection_ptr &conn) {
66              //用户注册功能请求的处理
67              websocketpp::http::parser::request req = conn->get_request();
68              //1. 获取到请求正文
69              std::string req_body = conn->get_request_body();
70              //2. 对正文进行json反序列化，得到用户名和密码
71              Json::Value login_info;
72              bool ret = json_util::unserialize(req_body, login_info);
73              if (ret == false) {
74                  DLOG("反序列化注册信息失败");
75                  return http_resp(conn, false,
    websocketpp::http::status_code::bad_request, "请求的正文格式错误");
76              }
77              //3. 进行数据库的用户新增操作
78              if (login_info["username"].isNull() ||
    login_info["password"].isNull()) {
79                  DLOG("用户名密码不完整");
80                  return http_resp(conn, false,
    websocketpp::http::status_code::bad_request, "请输入用户名/密码");
81              }
82              ret = _ut.insert(login_info);
83              if (ret == false) {
84                  DLOG("向数据库插入数据失败");
85                  return http_resp(conn, false,
    websocketpp::http::status_code::bad_request, "用户名已经被占用!");
86              }
87              //  如果成功了，则返回200
88              return http_resp(conn, true, websocketpp::http::status_code::ok, "注
    册用户成功");
89          }
90          void login(wsserver_t::connection_ptr &conn) {
```

```
91              //用户登录功能请求的处理
92              //1. 获取请求正文，并进行json反序列化，得到用户名和密码
93              std::string req_body = conn->get_request_body();
94              Json::Value login_info;
95              bool ret = json_util::unserialize(req_body, login_info);
96              if (ret == false) {
97                  DLOG("反序列化登录信息失败");
98                  return http_resp(conn, false,
    websocketpp::http::status_code::bad_request, "请求的正文格式错误");
99              }
100             //2. 校验正文完整性，进行数据库的用户信息验证
101             if (login_info["username"].isNull() ||
    login_info["password"].isNull()) {
102                 DLOG("用户名密码不完整");
103                 return http_resp(conn, false,
    websocketpp::http::status_code::bad_request, "请输入用户名/密码");
104             }
105             ret = _ut.login(login_info);
106             if (ret == false) {
107                 //  1. 如果验证失败，则返回400
108                 DLOG("用户名密码错误");
109                 return http_resp(conn, false,
    websocketpp::http::status_code::bad_request, "用户名密码错误");
110             }
111             //3. 如果验证成功，给客户端创建session
112             uint64_t uid = login_info["id"].asUInt64();
113             session_ptr ssp = _sm.create_session(uid, LOGIN);
114             if (ssp.get() == nullptr) {
115                 DLOG("创建会话失败");
116                 return http_resp(conn, false,
    websocketpp::http::status_code::internal_server_error , "创建会话失败");
117             }
118             _sm.set_session_expire_time(ssp->ssid(), SESSION_TIMEOUT);
119             //4. 设置响应头部: Set-Cookie,将sessionid通过cookie返回
120             std::string cookie_ssid = "SSID=" + std::to_string(ssp->ssid());
121             conn->append_header("Set-Cookie", cookie_ssid);
122             return http_resp(conn, true, websocketpp::http::status_code::ok ,
    "登录成功");
123         }
124         bool get_cookie_val(const std::string &cookie_str, const std::string
    &key,  std::string &val) {
125             // Cookie: SSID=XXX; path=/;
126             //1. 以 ; 作为间隔，对字符串进行分割，得到各个单个的cookie信息
127             std::string sep = "; ";
128             std::vector<std::string> cookie_arr;
129             string_util::split(cookie_str, sep, cookie_arr);
130             for (auto str : cookie_arr) {
```

```
131                 //2. 对单个cookie字符串，以 = 为间隔进行分割，得到key和val
132                 std::vector<std::string> tmp_arr;
133                 string_util::split(str, "=", tmp_arr);
134                 if (tmp_arr.size() != 2) { continue; }
135                 if (tmp_arr[0] == key) {
136                     val = tmp_arr[1];
137                     return true;
138                 }
139             }
140             return false;
141         }
142         void info(wsserver_t::connection_ptr &conn) {
143             //用户信息获取功能请求的处理
144             Json::Value err_resp;
145             // 1. 获取请求信息中的Cookie，从Cookie中获取ssid
146             std::string cookie_str = conn->get_request_header("Cookie");
147             if (cookie_str.empty()) {
148                 //如果没有cookie，返回错误：没有cookie信息，让客户端重新登录
149                 return http_resp(conn, true,
    websocketpp::http::status_code::bad_request, "找不到cookie信息，请重新登录");
150             }
151             // 1.5. 从cookie中取出ssid
152             std::string ssid_str;
153             bool ret = get_cookie_val(cookie_str, "SSID", ssid_str);
154             if (ret == false) {
155                 //cookie中没有ssid，返回错误：没有ssid信息，让客户端重新登录
156                 return http_resp(conn, true,
    websocketpp::http::status_code::bad_request, "找不到ssid信息，请重新登录");
157             }
158             // 2. 在session管理中查找对应的会话信息
159             session_ptr ssp = _sm.get_session_by_ssid(std::stol(ssid_str));
160             if (ssp.get() == nullptr) {
161                 //没有找到session，则认为登录已经过期，需要重新登录
162                 return http_resp(conn, true,
    websocketpp::http::status_code::bad_request, "登录过期，请重新登录");
163             }
164             // 3. 从数据库中取出用户信息，进行序列化发送给客户端
165             uint64_t uid = ssp->get_user();
166             Json::Value user_info;
167             ret = _ut.select_by_id(uid, user_info);
168             if (ret == false) {
169                 //获取用户信息失败，返回错误：找不到用户信息
170                 return http_resp(conn, true,
    websocketpp::http::status_code::bad_request, "找不到用户信息，请重新登录");
171             }
172             std::string body;
173             json_util::serialize(user_info, body);
```

```
174             conn->set_body(body);
175             conn->append_header("Content-Type", "application/json");
176             conn->set_status(websocketpp::http::status_code::ok);
177             // 4. 刷新session的过期时间
178             _sm.set_session_expire_time(ssp->ssid(), SESSION_TIMEOUT);
179         }
180         void http_callback(websocketpp::connection_hdl hdl) {
181             wsserver_t::connection_ptr conn = _wssrv.get_con_from_hdl(hdl);
182             websocketpp::http::parser::request req = conn->get_request();
183             std::string method = req.get_method();
184             std::string uri = req.get_uri();
185             if (method == "POST" && uri == "/reg") {
186                 return reg(conn);
187             }else if (method == "POST" && uri == "/login") {
188                 return login(conn);
189             }else if (method == "GET" && uri == "/info") {
190                 return info(conn);
191             }else {
192                 return file_handler(conn);
193             }
194         }
195         void ws_resp(wsserver_t::connection_ptr conn, Json::Value &resp) {
196             std::string body;
197             json_util::serialize(resp, body);
198             conn->send(body);
199         }
200         session_ptr get_session_by_cookie(wsserver_t::connection_ptr conn) {
201             Json::Value err_resp;
202             // 1. 获取请求信息中的Cookie，从Cookie中获取ssid
203             std::string cookie_str = conn->get_request_header("Cookie");
204             if (cookie_str.empty()) {
205                 //如果没有cookie，返回错误：没有cookie信息，让客户端重新登录
206                 err_resp["optype"] = "hall_ready";
207                 err_resp["reason"] = "没有找到cookie信息，需要重新登录";
208                 err_resp["result"] = false;
209                 ws_resp(conn, err_resp);
210                 return session_ptr();
211             }
212             // 1.5. 从cookie中取出ssid
213             std::string ssid_str;
214             bool ret = get_cookie_val(cookie_str, "SSID", ssid_str);
215             if (ret == false) {
216                 //cookie中没有ssid，返回错误：没有ssid信息，让客户端重新登录
217                 err_resp["optype"] = "hall_ready";
218                 err_resp["reason"] = "没有找到SSID信息，需要重新登录";
219                 err_resp["result"] = false;
220                 ws_resp(conn, err_resp);
```

```
221                 return session_ptr();
222             }
223             // 2. 在session管理中查找对应的会话信息
224             session_ptr ssp = _sm.get_session_by_ssid(std::stol(ssid_str));
225             if (ssp.get() == nullptr) {
226                 //没有找到session，则认为登录已经过期，需要重新登录
227                 err_resp["optype"] = "hall_ready";
228                 err_resp["reason"] = "没有找到session信息，需要重新登录";
229                 err_resp["result"] = false;
230                 ws_resp(conn, err_resp);
231                 return session_ptr();
232             }
233             return ssp;
234         }
235         void wsopen_game_hall(wsserver_t::connection_ptr conn) {
236             //游戏大厅长连接建立成功
237             Json::Value resp_json;
238             //1. 登录验证--判断当前客户端是否已经成功登录
239             session_ptr ssp = get_session_by_cookie(conn);
240             if (ssp.get() == nullptr) {
241                 return;
242             }
243             //2. 判断当前客户端是否是重复登录
244             if (_om.is_in_game_hall(ssp->get_user()) ||
    _om.is_in_game_room(ssp->get_user())) {
245                 resp_json["optype"] = "hall_ready";
246                 resp_json["reason"] = "玩家重复登录! ";
247                 resp_json["result"] = false;
248                 return ws_resp(conn, resp_json);
249             }
250             //3. 将当前客户端以及连接加入到游戏大厅
251             _om.enter_game_hall(ssp->get_user(), conn);
252             //4. 给客户端响应游戏大厅连接建立成功
253             resp_json["optype"] = "hall_ready";
254             resp_json["result"] = true;
255             ws_resp(conn, resp_json);
256             //5. 记得将session设置为永久存在
257             _sm.set_session_expire_time(ssp->ssid(), SESSION_FOREVER);
258         }
259         void wsopen_game_room(wsserver_t::connection_ptr conn) {
260             Json::Value resp_json;
261             //1. 获取当前客户端的session
262             session_ptr ssp = get_session_by_cookie(conn);
263             if (ssp.get() == nullptr) {
264                 return;
265             }
266             //2. 当前用户是否已经在在线用户管理的游戏房间或者游戏大厅中---在线用户管理
```

```
267             if (_om.is_in_game_hall(ssp->get_user()) ||
    _om.is_in_game_room(ssp->get_user())) {
268                 resp_json["optype"] = "room_ready";
269                 resp_json["reason"] = "玩家重复登录! ";
270                 resp_json["result"] = false;
271                 return ws_resp(conn, resp_json);
272             }
273             //3. 判断当前用户是否已经创建好了房间 --- 房间管理
274             room_ptr rp = _rm.get_room_by_uid(ssp->get_user());
275             if (rp.get() == nullptr) {
276                 resp_json["optype"] = "room_ready";
277                 resp_json["reason"] = "没有找到玩家的房间信息";
278                 resp_json["result"] = false;
279                 return ws_resp(conn, resp_json);
280             }
281             //4. 将当前用户添加到在线用户管理的游戏房间中
282             _om.enter_game_room(ssp->get_user(), conn);
283             //5. 将session重新设置为永久存在
284             _sm.set_session_expire_time(ssp->ssid(), SESSION_FOREVER);
285             //6. 回复房间准备完毕
286             resp_json["optype"] = "room_ready";
287             resp_json["result"] = true;
288             resp_json["room_id"] = (Json::UInt64)rp->id();
289             resp_json["uid"] = (Json::UInt64)ssp->get_user();
290             resp_json["white_id"] = (Json::UInt64)rp->get_white_user();
291             resp_json["black_id"] = (Json::UInt64)rp->get_black_user();
292             return ws_resp(conn, resp_json);
293         }
294         void wsopen_callback(websocketpp::connection_hdl hdl) {
295             //websocket长连接建立成功之后的处理函数
296             wsserver_t::connection_ptr conn = _wssrv.get_con_from_hdl(hdl);
297             websocketpp::http::parser::request req = conn->get_request();
298             std::string uri = req.get_uri();
299             if (uri == "/hall") {
300                 //建立了游戏大厅的长连接
301                 return wsopen_game_hall(conn);
302             }else if (uri == "/room") {
303                 //建立了游戏房间的长连接
304                 return wsopen_game_room(conn);
305             }
306         }
307         void wsclose_game_hall(wsserver_t::connection_ptr conn) {
308             //游戏大厅长连接断开的处理
309             //1. 登录验证--判断当前客户端是否已经成功登录
310             session_ptr ssp = get_session_by_cookie(conn);
311             if (ssp.get() == nullptr) {
312                 return;
```

```
313             }
314             //1. 将玩家从游戏大厅中移除
315             _om.exit_game_hall(ssp->get_user());
316             //2. 将session恢复生命周期的管理，设置定时销毁
317             _sm.set_session_expire_time(ssp->ssid(), SESSION_TIMEOUT);
318         }
319         void wsclose_game_room(wsserver_t::connection_ptr conn) {
320             //获取会话信息，识别客户端
321             session_ptr ssp = get_session_by_cookie(conn);
322             if (ssp.get() == nullptr) {
323                 return;
324             }
325             //1. 将玩家从在线用户管理中移除
326             _om.exit_game_room(ssp->get_user());
327             //2. 将session回复生命周期的管理，设置定时销毁
328             _sm.set_session_expire_time(ssp->ssid(), SESSION_TIMEOUT);
329             //3. 将玩家从游戏房间中移除，房间中所有用户退出了就会销毁房间
330             _rm.remove_room_user(ssp->get_user());
331         }
332         void wsclose_callback(websocketpp::connection_hdl hdl) {
333             //websocket连接断开前的处理
334             wsserver_t::connection_ptr conn = _wssrv.get_con_from_hdl(hdl);
335             websocketpp::http::parser::request req = conn->get_request();
336             std::string uri = req.get_uri();
337             if (uri == "/hall") {
338                 //建立了游戏大厅的长连接
339                 return wsclose_game_hall(conn);
340             }else if (uri == "/room") {
341                 //建立了游戏房间的长连接
342                 return wsclose_game_room(conn);
343             }
344         }
345         void wsmsg_game_hall(wsserver_t::connection_ptr conn,
    wsserver_t::message_ptr msg) {
346             Json::Value resp_json;
347             std::string resp_body;
348             //1. 身份验证，当前客户端到底是哪个玩家
349             session_ptr ssp = get_session_by_cookie(conn);
350             if (ssp.get() == nullptr) {
351                 return;
352             }
353             //2. 获取请求信息
354             std::string req_body = msg->get_payload();
355             Json::Value req_json;
356             bool ret = json_util::unserialize(req_body, req_json);
357             if (ret == false) {
358                 resp_json["result"] = false;
```

```
359                 resp_json["reason"] = "请求信息解析失败";
360                 return ws_resp(conn, resp_json);
361             }
362             //3. 对于请求进行处理:
363             if (!req_json["optype"].isNull() && req_json["optype"].asString()
    == "match_start"){
364                 //  开始对战匹配：通过匹配模块，将用户添加到匹配队列中
365                 _mm.add(ssp->get_user());
366                 resp_json["optype"] = "match_start";
367                 resp_json["result"] = true;
368                 return ws_resp(conn, resp_json);
369             }else if (!req_json["optype"].isNull() &&
    req_json["optype"].asString() == "match_stop") {
370                 //  停止对战匹配：通过匹配模块，将用户从匹配队列中移除
371                 _mm.del(ssp->get_user());
372                 resp_json["optype"] = "match_stop";
373                 resp_json["result"] = true;
374                 return ws_resp(conn, resp_json);
375             }
376             resp_json["optype"] = "unknow";
377             resp_json["reason"] = "请求类型未知";
378             resp_json["result"] = false;
379             return ws_resp(conn, resp_json);
380         }
381         void wsmsg_game_room(wsserver_t::connection_ptr conn,
    wsserver_t::message_ptr msg) {
382             Json::Value resp_json;
383             //1. 获取客户端session，识别客户端身份
384             session_ptr ssp = get_session_by_cookie(conn);
385             if (ssp.get() == nullptr) {
386                 DLOG("房间-没有找到会话信息");
387                 return;
388             }
389             //2. 获取客户端房间信息
390             room_ptr rp = _rm.get_room_by_uid(ssp->get_user());
391             if (rp.get() == nullptr) {
392                 resp_json["optype"] = "unknow";
393                 resp_json["reason"] = "没有找到玩家的房间信息";
394                 resp_json["result"] = false;
395                 DLOG("房间-没有找到玩家房间信息");
396                 return ws_resp(conn, resp_json);
397             }
398             //3. 对消息进行反序列化
399             Json::Value req_json;
400             std::string req_body = msg->get_payload();
401             bool ret = json_util::unserialize(req_body, req_json);
402             if (ret == false) {
```

```
403                 resp_json["optype"] = "unknow";
404                 resp_json["reason"] = "请求解析失败";
405                 resp_json["result"] = false;
406                 DLOG("房间-反序列化请求失败");
407                 return ws_resp(conn, resp_json);
408             }
409             DLOG("房间: 收到房间请求，开始处理....");
410             //4. 通过房间模块进行消息请求的处理
411             return rp->handle_request(req_json);
412         }
413         void wsmsg_callback(websocketpp::connection_hdl hdl,
    wsserver_t::message_ptr msg) {
414             //websocket长连接通信处理
415             wsserver_t::connection_ptr conn = _wssrv.get_con_from_hdl(hdl);
416             websocketpp::http::parser::request req = conn->get_request();
417             std::string uri = req.get_uri();
418             if (uri == "/hall") {
419                 //建立了游戏大厅的长连接
420                 return wsmsg_game_hall(conn, msg);
421             }else if (uri == "/room") {
422                 //建立了游戏房间的长连接
423                 return wsmsg_game_room(conn, msg);
424             }
425         }
426     public:
427         /*进行成员初始化，以及服务器回调函数的设置*/
428         gobang_server(const std::string &host,
429                 const std::string &user,
430                 const std::string &pass,
431                 const std::string &dbname,
432                 uint16_t port = 3306,
433                 const std::string &wwwroot = WWWROOT):
434                 _web_root(wwwroot), _ut(host, user, pass, dbname, port),
435                 _rm(&_ut, &_om), _sm(&_wssrv), _mm(&_rm, &_ut, &_om) {
436             _wssrv.set_access_channels(websocketpp::log::alevel::none);
437             _wssrv.init_asio();
438             _wssrv.set_reuse_addr(true);
439             _wssrv.set_http_handler(std::bind(&gobang_server::http_callback,
    this, std::placeholders::_1));
440             _wssrv.set_open_handler(std::bind(&gobang_server::wsopen_callback,
    this, std::placeholders::_1));
441 
    _wssrv.set_close_handler(std::bind(&gobang_server::wsclose_callback, this,
    std::placeholders::_1));
442 
    _wssrv.set_message_handler(std::bind(&gobang_server::wsmsg_callback, this,
    std::placeholders::_1, std::placeholders::_2));
```

```
443         }
444         /*启动服务器*/
445         void start(int port) {
446             _wssrv.listen(port);
447             _wssrv.start_accept();
448             _wssrv.run();
449         }
450 };
451 #endif
```

14. 客户端开发

登录页面: login.html

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta http-equiv="X-UA-Compatible" content="IE=edge">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>登录</title>
8
9      <link rel="stylesheet" href="./css/common.css">
10     <link rel="stylesheet" href="./css/login.css">
11 </head>
12 <body>
13     <div class="nav">
14         网络五子棋对战游戏
15     </div>
16     <div class="login-container">
17         <!-- 登录界面的对话框 -->
18         <div class="login-dialog">
19             <!-- 提示信息 -->
20             <h3>登录</h3>
21             <!-- 这个表示一行 -->
22             <div class="row">
23                 <span>用户名</span>
24                 <input type="text" id="user_name">
25             </div>
26             <!-- 这是另一行 -->
27             <div class="row">
28                 <span>密码</span>
29                 <input type="password" id="password">
30             </div>
31             <!-- 提交按钮 -->
```

```
            <div class="row">
                <button id="submit" onclick="login()">提交</button>
            </div>
        </div>

    </div>

    <script src="./js/jquery.min.js"></script>
    <script>
        //1. 给按钮添加点击事件，调用登录请求函数
        //2. 封装登录请求函数
        function login() {
            //  1. 获取输入框中的用户名和密码，并组织json对象
            var login_info = {
                username: document.getElementById("user_name").value,
                password: document.getElementById("password").value
            };
            //  2. 通过ajax向后台发送登录验证请求
            $.ajax({
                url: "/login",
                type: "post",
                data: JSON.stringify(login_info),
                success: function(result) {
                    //  3. 如果验证通过，则跳转游戏大厅页面
                    alert("登录成功");
                    window.location.assign("/game_hall.html");
                },
                error: function(xhr) {
                    //  4. 如果验证失败，则提示错误信息，并清空输入框
                    alert(JSON.stringify(xhr));
                    document.getElementById("user_name").value = "";
                    document.getElementById("password").value = "";
                }
            })
        }

    </script>
</body>
</html>
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
```

```
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>登录</title>

    <link rel="stylesheet" href="./css/common.css">
    <link rel="stylesheet" href="./css/login.css">
</head>
<body>
    <div class="nav">
        网络五子棋对战游戏
    </div>
    <div class="login-container">
        <!-- 登录界面的对话框 -->
        <div class="login-dialog">
            <!-- 提示信息 -->
            <h3>登录</h3>
            <!-- 这个表示一行 -->
            <div class="row">
                <span>用户名</span>
                <input type="text" id="user_name">
            </div>
            <!-- 这是另一行 -->
            <div class="row">
                <span>密码</span>
                <input type="password" id="password">
            </div>
            <!-- 提交按钮 -->
            <div class="row">
                <button id="submit">提交</button>
            </div>
        </div>

    </div>

    <script src="./js/jquery.min.js"></script>
    <script>
        // 获取用户在前端输入的用户名和密码
        let usernameInput = document.getElementById('user_name');
        let passwordInput = document.getElementById('password');
        let submitButton = document.getElementById('submit');
        // 点击提交按钮的回调函数
        submitButton.onclick = function() {
            // 通过 ajax 向服务器发起登录请求 实现登录功能
            $.ajax({
                // 构造请求
                type: 'post',
                url: '/login',
                data: {
```

```
                    username: usernameInput.value,
                    password: passwordInput.value,
                },
                success: function(body) {
                    alert("登录成功!");
                    // 重定向跳转到 "游戏大厅页面"
                    location.assign('/game_hall.html');
                },
                error: function(body) {
                    alert(JSON.stringify(body));
                }
            });
        }
    </script>
</body>
</html>
```

注册页面: register.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>注册</title>
    <link rel="stylesheet" href="./css/common.css">
    <link rel="stylesheet" href="./css/login.css">
</head>
<body>
    <div class="nav">
        网络五子棋对战游戏
    </div>
    <div class="login-container">
        <!-- 登录界面的对话框 -->
        <div class="login-dialog">
            <!-- 提示信息 -->
            <h3>注册</h3>
            <!-- 这个表示一行 -->
            <div class="row">
                <span>用户名</span>
                <input type="text" id="user_name" name="username">
            </div>
            <!-- 这是另一行 -->
            <div class="row">
```

```
                <span>密码</span>
                <input type="password" id="password" name="password">
            </div>
            <!-- 提交按钮 -->
            <div class="row">
                <button id="submit">提交</button>
            </div>
        </div>
    </div>

    <script src="js/jquery.min.js"></script>
</body>
</html>
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>注册</title>
    <link rel="stylesheet" href="./css/common.css">
    <link rel="stylesheet" href="./css/login.css">
</head>
<body>
    <div class="nav">
        网络五子棋对战游戏
    </div>
    <div class="login-container">
        <!-- 登录界面的对话框 -->
        <div class="login-dialog">
            <!-- 提示信息 -->
            <h3>注册</h3>
            <!-- 这个表示一行 -->
            <div class="row">
                <span>用户名</span>
                <input type="text" id="user_name" name="username">
            </div>
            <!-- 这是另一行 -->
            <div class="row">
                <span>密码</span>
                <input type="password" id="password" name="password">
            </div>
            <!-- 提交按钮 -->
            <div class="row">
```

```
                <button id="submit" onclick="reg()">提交</button>
            </div>
        </div>
    </div>

    <script src="js/jquery.min.js"></script>
    <script>
        //1. 给按钮添加点击事件，调用注册函数
        //2. 封装实现注册函数
        function reg() {
            //  1. 获取两个输入框空间中的数据，组织成为一个json串
            var reg_info = {
                username: document.getElementById("user_name").value,
                password: document.getElementById("password").value
            };
            console.log(JSON.stringify(reg_info));
            //  2. 通过ajax向后台发送用户注册请求
            $.ajax({
                url : "/reg",
                type : "post",
                data : JSON.stringify(reg_info),
                success : function(res) {
                    if (res.result == false) {
                        //  4. 如果请求失败，则清空两个输入框内容，并提示错误原因
                        document.getElementById("user_name").value = "";
                        document.getElementById("password").value = "";
                        alert(res.reason);
                    }else {
                        //  3. 如果请求成功，则跳转的登录页面
                        alert(res.reason);
                        window.location.assign("/login.html");
                    }
                },
                error : function(xhr) {
                    document.getElementById("user_name").value = "";
                    document.getElementById("password").value = "";
                    alert(JSON.stringify(xhr));
                }
            })
        }
    </script>
</body>
</html>
```

游戏大厅页面：game_hall.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>游戏大厅</title>
    <link rel="stylesheet" href="./css/common.css">
    <link rel="stylesheet" href="./css/game_hall.css">
</head>
<body>
    <div class="nav">网络五子棋对战游戏</div>
    <!-- 整个页面的容器元素 -->
    <div class="container">
        <!-- 这个 div 在 container 中是处于垂直水平居中这样的位置的 -->
        <div>
            <!-- 展示用户信息 -->
            <div id="screen">
                玩家：小白 分数：1860</br>
                比赛场次：23 获胜场次：18
            </div>
            <!-- 匹配按钮 -->
            <div id="match-button">开始匹配</div>
        </div>
    </div>

    <script src="./js/jquery.min.js"></script>
</body>
</html>
```

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>游戏大厅</title>
    <link rel="stylesheet" href="./css/common.css">
    <link rel="stylesheet" href="./css/game_hall.css">
</head>
<body>
    <div class="nav">网络五子棋对战游戏</div>
    <!-- 整个页面的容器元素 -->
    <div class="container">
        <!-- 这个 div 在 container 中是处于垂直水平居中这样的位置的 -->
```

```
        <div>
            <!-- 展示用户信息 -->
            <div id="screen"></div>
            <!-- 匹配按钮 -->
            <div id="match-button">开始匹配</div>
        </div>
    </div>

    <script src="./js/jquery.min.js"></script>
    <script>
        var ws_url = "ws://" + location.host + "/hall";
        var ws_hdl = null;

        window.onbeforeunload = function() {
            ws_hdl.close();
        }
        //按钮有两个状态：没有进行匹配的状态，正在匹配中的状态
        var button_flag = "stop";
        //点击按钮的事件处理：
        var be = document.getElementById("match-button");
        be.onclick = function() {
            if (button_flag == "stop") {
                //1. 没有进行匹配的状态下点击按钮，发送对战匹配请求
                var req_json = {
                    optype: "match_start"
                }
                ws_hdl.send(JSON.stringify(req_json));
            }else {
                //2. 正在匹配中的状态下点击按钮，发送停止对战匹配请求
                var req_json = {
                    optype: "match_stop"
                }
                ws_hdl.send(JSON.stringify(req_json));
            }
        }
        function get_user_info() {
            $.ajax({
                url: "/info",
                type: "get",
                success: function(res) {
                    var info_html = "<p>" + "用户: " + res.username + " 积分: "
+ res.score +
                        "</br>" + "比赛场次: " + res.total_count + " 获胜场次: "
+ res.win_count + "</p>";
                    var screen_div = document.getElementById("screen");
                    screen_div.innerHTML = info_html;

```

```
                    ws_hdl = new WebSocket(ws_url);
                    ws_hdl.onopen = ws_onopen;
                    ws_hdl.onclose = ws_onclose;
                    ws_hdl.onerror = ws_onerror;
                    ws_hdl.onmessage = ws_onmessage;
                },
                error: function(xhr) {
                    alert(JSON.stringify(xhr));
                    location.replace("/login.html");
                }
            })
        }
        function ws_onopen() {
            console.log("websocket onopen");
        }
        function ws_onclose() {
            console.log("websocket onopen");
        }
        function ws_onerror() {
            console.log("websocket onopen");
        }
        function ws_onmessage(evt) {
            var rsp_json = JSON.parse(evt.data);
            if (rsp_json.result == false) {
                alert(evt.data);
                location.replace("/login.html");
                return;
            }
            if (rsp_json["optype"] == "hall_ready") {
                alert("游戏大厅连接建立成功! ");
            }else if (rsp_json["optype"] == "match_success") {
                //对战匹配成功
                alert("对战匹配成功，进入游戏房间! ");
                location.replace("/game_room.html");
            }else if (rsp_json["optype"] == "match_start") {
                console.log("玩家已经加入匹配队列");
                button_flag = "start";
                be.innerHTML = "匹配中....点击按钮停止匹配!";
                return;
            }else if (rsp_json["optype"] == "match_stop"){
                console.log("玩家已经移除匹配队列");
                button_flag = "stop";
                be.innerHTML = "开始匹配";
                return;
            }else {
                alert(evt.data);
                location.replace("/login.html");
```

```
                return;
            }
        }
        get_user_info();
    </script>
</body>
</html>
```

游戏房间页面：game_room.html

在游戏房间页面中，关于棋盘的绘制部分已经直接提供。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>游戏房间</title>
    <link rel="stylesheet" href="css/common.css">
    <link rel="stylesheet" href="css/game_room.css">
</head>
<body>
    <div class="nav">网络五子棋对战游戏</div>
    <div class="container">
        <div id="chess_area">
            <!-- 棋盘区域，需要基于 canvas 进行实现 -->
            <canvas id="chess" width="450px" height="450px"></canvas>
            <!-- 显示区域 -->
            <div id="screen"> 等待玩家连接中... </div>
        </div>
        <div id="chat_area" width="400px" height="300px">
            <div id="chat_show">
                <p id="self_msg">你好! </p></br>
                <p id="peer_msg">你好! </p></br>
            </div>
            <div id="msg_show">
                <input type="text" id="chat_input">
                <button id="chat_button">发送</button>
            </div>
        </div>
    </div>
    <script>
        let chessBoard = [];
        let BOARD_ROW_AND_COL = 15;
        let chess = document.getElementById('chess');
```

```
35          //获取chess控件区域2d画布
36          let context = chess.getContext('2d');
37          function initGame() {
38              initBoard();
39              // 背景图片
40              let logo = new Image();
41              logo.src = "image/sky.jpeg";
42              logo.onload = function () {
43                  // 绘制图片
44                  context.drawImage(logo, 0, 0, 450, 450);
45                  // 绘制棋盘
46                  drawChessBoard();
47              }
48          }
49          function initBoard() {
50              for (let i = 0; i < BOARD_ROW_AND_COL; i++) {
51                  chessBoard[i] = [];
52                  for (let j = 0; j < BOARD_ROW_AND_COL; j++) {
53                      chessBoard[i][j] = 0;
54                  }
55              }
56          }
57          // 绘制棋盘网格线
58          function drawChessBoard() {
59              context.strokeStyle = "#BFBFBF";
60              for (let i = 0; i < BOARD_ROW_AND_COL; i++) {
61                  //横向的线条
62                  context.moveTo(15 + i * 30, 15);
63                  context.lineTo(15 + i * 30, 430);
64                  context.stroke();
65                  //纵向的线条
66                  context.moveTo(15, 15 + i * 30);
67                  context.lineTo(435, 15 + i * 30);
68                  context.stroke();
69              }
70          }
71          //绘制棋子
72          function oneStep(i, j, isWhite) {
73              if (i < 0 || j < 0) return;
74              context.beginPath();
75              context.arc(15 + i * 30, 15 + j * 30, 13, 0, 2 * Math.PI);
76              context.closePath();
77              //createLinearGradient() 方法创建放射状/圆形渐变对象
78              var gradient = context.createRadialGradient(15 + i * 30 + 2, 15 +
    j * 30 - 2, 13, 15 + i * 30 + 2, 15 + j * 30 - 2, 0);
79              // 区分黑白子
80              if (!isWhite) {
```

```
                gradient.addColorStop(0, "#0A0A0A");
                gradient.addColorStop(1, "#636766");
            } else {
                gradient.addColorStop(0, "#D1D1D1");
                gradient.addColorStop(1, "#F9F9F9");
            }
            context.fillStyle = gradient;
            context.fill();
        }
        //棋盘区域的点击事件
        chess.onclick = function (e) {
            let x = e.offsetX;
            let y = e.offsetY;
            // 注意，横坐标是列，纵坐标是行
            // 这里是为了让点击操作能够对应到网格线上
            let col = Math.floor(x / 30);
            let row = Math.floor(y / 30);
            if (chessBoard[row][col] != 0) {
                alert("当前位置已有棋子！");
                return;
            }
            oneStep(col, row, true);
        }
        initGame();
    </script>
</body>
</html>
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>游戏房间</title>
    <link rel="stylesheet" href="css/common.css">
    <link rel="stylesheet" href="css/game_room.css">
</head>
<body>
    <div class="nav">网络五子棋对战游戏</div>
    <div class="container">
        <div id="chess_area">
            <!-- 棋盘区域，需要基于 canvas 进行实现 -->
            <canvas id="chess" width="450px" height="450px"></canvas>
            <!-- 显示区域 -->
```

```
            <div id="screen"> 等待玩家连接中... </div>
        </div>
        <div id="chat_area" width="400px" height="300px">
            <div id="chat_show">
                <p id="self_msg">你好! </p></br>
                <p id="peer_msg">你好! </p></br>
                <p id="peer_msg">leihoua~</p></br>
            </div>
            <div id="msg_show">
                <input type="text" id="chat_input">
                <button id="chat_button">发送</button>
            </div>
        </div>
    </div>
    <script>
        let chessBoard = [];
        let BOARD_ROW_AND_COL = 15;
        let chess = document.getElementById('chess');
        let context = chess.getContext('2d');//获取chess控件的2d画布

        var ws_url = "ws://" + location.host + "/room";
        var ws_hdl = new WebSocket(ws_url);

        var room_info = null;//用于保存房间信息
        var is_me;

        function initGame() {
            initBoard();
            context.strokeStyle = "#BFBFBF";
            // 背景图片
            let logo = new Image();
            logo.src = "image/sky.jpeg";
            logo.onload = function () {
                // 绘制图片
                context.drawImage(logo, 0, 0, 450, 450);
                // 绘制棋盘
                drawChessBoard();
            }
        }
        function initBoard() {
            for (let i = 0; i < BOARD_ROW_AND_COL; i++) {
                chessBoard[i] = [];
                for (let j = 0; j < BOARD_ROW_AND_COL; j++) {
                    chessBoard[i][j] = 0;
                }
            }
        }
```

```
65          // 绘制棋盘网格线
66          function drawChessBoard() {
67              for (let i = 0; i < BOARD_ROW_AND_COL; i++) {
68                  context.moveTo(15 + i * 30, 15);
69                  context.lineTo(15 + i * 30, 430); //横向的线条
70                  context.stroke();
71                  context.moveTo(15, 15 + i * 30);
72                  context.lineTo(435, 15 + i * 30); //纵向的线条
73                  context.stroke();
74              }
75          }
76          //绘制棋子
77          function oneStep(i, j, isWhite) {
78              if (i < 0 || j < 0) return;
79              context.beginPath();
80              context.arc(15 + i * 30, 15 + j * 30, 13, 0, 2 * Math.PI);
81              context.closePath();
82              var gradient = context.createRadialGradient(15 + i * 30 + 2, 15 +
    j * 30 - 2, 13, 15 + i * 30 + 2, 15 + j * 30 - 2, 0);
83              // 区分黑白子
84              if (!isWhite) {
85                  gradient.addColorStop(0, "#0A0A0A");
86                  gradient.addColorStop(1, "#636766");
87              } else {
88                  gradient.addColorStop(0, "#D1D1D1");
89                  gradient.addColorStop(1, "#F9F9F9");
90              }
91              context.fillStyle = gradient;
92              context.fill();
93          }
94          //棋盘区域的点击事件
95          chess.onclick = function (e) {
96              //  1. 获取下棋位置，判断当前下棋操作是否正常
97              //      1. 当前是否轮到自己走棋了
98              //      2. 当前位置是否已经被占用
99              //  2. 向服务器发送走棋请求
100             if (!is_me) {
101                 alert("等待对方走棋....");
102                 return;
103             }
104             let x = e.offsetX;
105             let y = e.offsetY;
106             // 注意，横坐标是列，纵坐标是行
107             // 这里是为了让点击操作能够对应到网格线上
108             let col = Math.floor(x / 30);
109             let row = Math.floor(y / 30);
110             if (chessBoard[row][col] != 0) {
```

```
111                 alert("当前位置已有棋子! ");
112                 return;
113             }
114             //oneStep(col, row, true);
115             //向服务器发送走棋请求，收到响应后，再绘制棋子
116             send_chess(row, col);
117         }
118         function send_chess(r, c) {
119             var chess_info = {
120                 optype : "put_chess",
121                 room_id: room_info.room_id,
122                 uid: room_info.uid,
123                 row: r,
124                 col: c
125             };
126             ws_hdl.send(JSON.stringify(chess_info));
127             console.log("click:" + JSON.stringify(chess_info));
128         }
129 
130         window.onbeforeunload = function() {
131             ws_hdl.close();
132         }
133         ws_hdl.onopen = function() {
134             console.log("房间长连接建立成功");
135         }
136         ws_hdl.onclose = function() {
137             console.log("房间长连接断开");
138         }
139         ws_hdl.onerror = function() {
140             console.log("房间长连接出错");
141         }
142         function set_screen(me) {
143             var screen_div = document.getElementById("screen");
144             if (me) {
145                 screen_div.innerHTML = "轮到己方走棋...";
146             }else {
147                 screen_div.innerHTML = "轮到对方走棋...";
148             }
149         }
150         ws_hdl.onmessage = function(evt) {
151             //1. 在收到room_ready之后进行房间的初始化
152             //  1. 将房间信息保存起来
153             var info = JSON.parse(evt.data);
154             console.log(JSON.stringify(info));
155             if (info.optype == "room_ready") {
156                 room_info = info;
157                 is_me = room_info.uid == room_info.white_id ? true : false;
```

```
158                 set_screen(is_me);
159                 initGame();
160             }else if (info.optype == "put_chess"){
161                 console.log("put_chess" + evt.data);
162                 //2. 走棋操作
163                 //  3. 收到走棋消息，进行棋子绘制
164                 if (info.result == false) {
165                     alert(info.reason);
166                     return;
167                 }
168                 //当前走棋的用户id，与我自己的用户id相同，就是我自己走棋，走棋之后，就轮
    到对方了
169                 is_me = info.uid == room_info.uid ? false : true;
170                 //绘制棋子的颜色，应该根据当前下棋角色的颜色确定
171                 isWhite = info.uid == room_info.white_id ? true : false;
172                 //绘制棋子
173                 if (info.row != -1 && info.col != -1){
174                     oneStep(info.col, info.row, isWhite);
175                     //设置棋盘信息
176                     chessBoard[info.row][info.col] = 1;
177                 }
178                 //是否有胜利者
179                 if (info.winner == 0) {
180                     return;
181                 }
182                 var screen_div = document.getElementById("screen");
183                 if (room_info.uid == info.winner) {
184                     screen_div.innerHTML = info.reason;
185                 }else {
186                     screen_div.innerHTML = "你输了";
187                 }
188 
189                 var chess_area_div = document.getElementById("chess_area");
190                 var button_div = document.createElement("div");
191                 button_div.innerHTML = "返回大厅";
192                 button_div.onclick = function() {
193                     ws_hdl.close();
194                     location.replace("/game_hall.html");
195                 }
196                 chess_area_div.appendChild(button_div);
197             } else if (info.optype == "chat") {
198                 //收到一条消息，判断result，如果为true则渲染一条消息到显示框中
199                 if(info.result == false) {
200                     alert(info.reason);
201                     return;
202                 }
203                 var msg_div = document.createElement("p");
```

```
                msg_div.innerHTML = info.message;
                if (info.uid == room_info.uid) {
                    msg_div.setAttribute("id", "self_msg");
                }else {
                    msg_div.setAttribute("id", "peer_msg");
                }
                var br_div = document.createElement("br");
                var msg_show_div = document.getElementById("chat_show");
                msg_show_div.appendChild(msg_div);
                msg_show_div.appendChild(br_div);
                document.getElementById("chat_input").value = "";
            }
        }
        //3. 聊天动作
        //  1. 捕捉聊天输入框消息
        //  2. 给发送按钮添加点击事件，点击俺就的时候，获取到输入框消息，发送给服务器
        var cb_div = document.getElementById("chat_button");
        cb_div.onclick = function() {
            var send_msg = {
                optype : "chat",
                room_id : room_info.room_id,
                uid : room_info.uid,
                message : document.getElementById("chat_input").value
            };
            ws_hdl.send(JSON.stringify(send_msg));
        }
    </script>
</body>
</html>
```

我们必须使用两个浏览器或者一个浏览器的无痕模式打开两个标签页， 避免cookie和session相互影响导致检测到多开。

15. 项目扩展

15.1 实现局时/步时

- 局时: 一局游戏中玩家能思考的总时间
- 步时: 一步落子过程中，玩家能思考的时间

15.2 保存棋谱&录像回放

- 服务器可以把每一局对局、玩家轮流落子的位置都记录下来
- 玩家可以在游戏大厅页面选定某个曾经的比赛，在页面上回放出对局的过程

15.3 观战功能

- 在游戏大厅显示当前所有的对局房间
- 玩家可以选中某个房间以观众的形式加入到房间中，实时的看到选手的对局情况

15.4 虚拟对手&人机对战

- 如果当前长时间匹配不到选手，则自动分配一个 AI 对手， 实现人机对战