《部落冲突》联网对战使用说明

目录

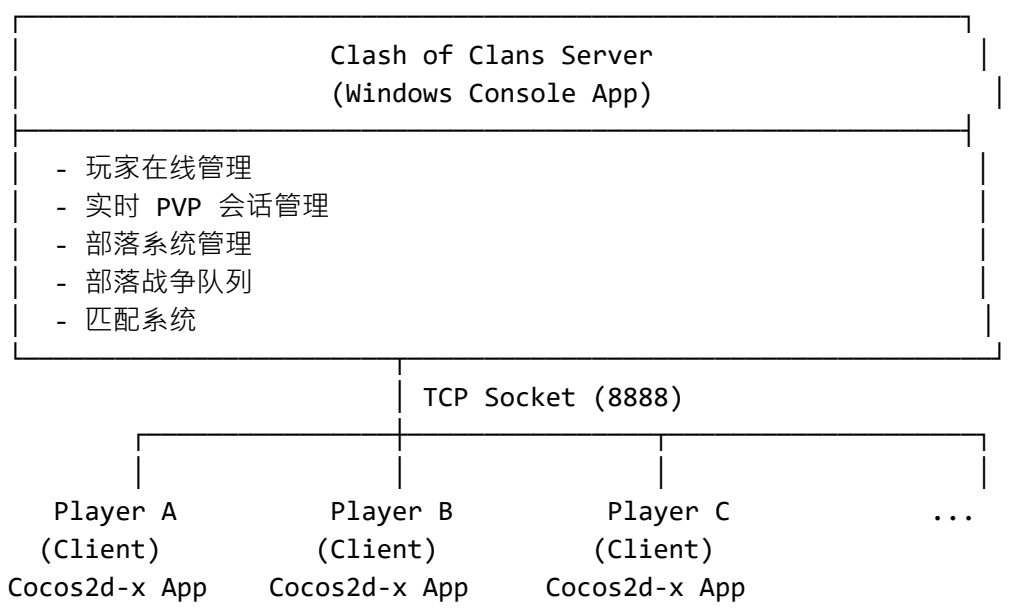
概述

本项目实现了一个基于 C++ 和 Cocos2d-x 的《部落冲突》风格的联网对战系统。系统采用 **客户端-服务器架构**，支持：

- ? **实时 PVP 对战**：玩家可实时挑战其他在线玩家
- ? **玩家列表系统**：获取可攻击目标的实时列表
- ? **部落系统**：创建、加入、管理部落
- ? **部落战争**：多个部落之间的群组对战
- ? **对战观战**：第三方玩家可观看实时对战过程
- ? **战斗回放**：保存并回放对战过程
- ? **资源掠夺**：对战获胜后掠夺对手金币和圣水

系统架构

整体拓扑



核心类结构

服务器端 (`Server.h / Server.cpp`)

类/结构体	说明
`PlayerContext`	玩家运行时上下文（Socket、ID、名称、资源、奖杯等）
`PvpSession`	实时 PVP 会话（攻击者、防御者、观战者列表、地图数据）
`ClanInfo`	部落信息（ID、名称、成员、总奖杯、加入条件）
`ClanWarInfo`	部落战争信息（参战部落、星数、攻击记录）
`AttackResult`	单场对战结果（攻击者、被攻击者、星数、掠夺资源等）
`Server`	服务器主类（管理连接、消息处理、业务逻辑）

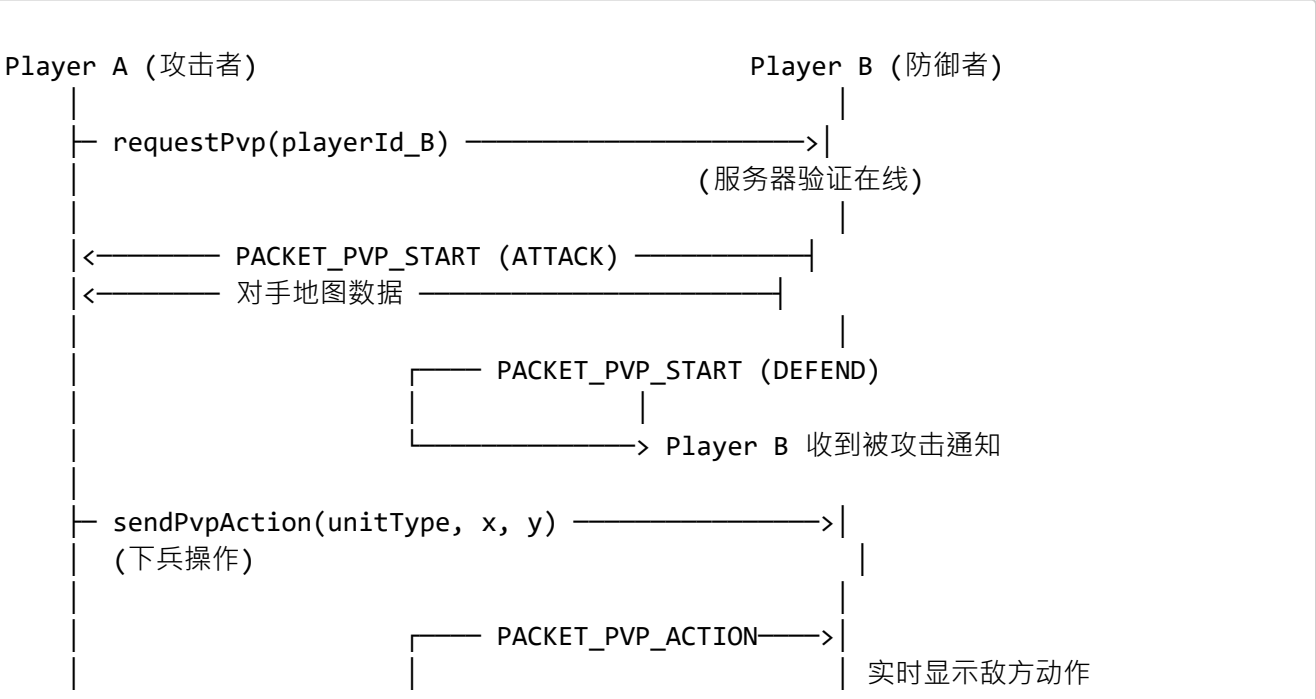
客户端 (`SocketClient.h / SocketClient.cpp`)

类/结构体	说明
`SocketClient`	单例网络管理器（连接、发送消息、接收处理）
`ReceivedPacket`	接收到的数据包
`MatchInfo`	匹配信息
`ClanInfoClient`	部落信息（客户端版本）

功能模块

1. 实时 PVP 对战

流程图：



```
    |                          |
    | (180秒对战时间...)
    |
    └─ endPvp() ──────────────────────────────────────>|
       (战斗结束)                                      |
                              ┌──── PACKET_PVP_END ────>|
                              |          (清空对战状态)
```

使用示例：

```
// 1. 获取玩家列表
socketClient.requestUserList();

// 2. 选择目标并请求 PVP
socketClient.requestPvp("player_id_123");

// 3. 注册 PVP 开始回调
socketClient.setOnPvpStart([](const std::string& role, const std::string&
opponentId, const std::string& mapData) {
    if (role == "ATTACK") {
        // 进入攻击界面，显示对手地图
        BattleManager::getInstance()->init(opponentMapData, opponentId, false);
    } else if (role == "DEFEND") {
        // 收到被攻击通知，等待对方行动
    }
});

// 4. 玩家下兵（点击地图位置）
socketClient.sendPvpAction(UNIT_BARBARIAN, 100.0f, 200.0f);

// 5. 战斗结束
socketClient.endPvp();
```

关键数据包：

- `PACKET_PVP_REQUEST (40)` - 请求发起 PVP
- `PACKET_PVP_START (41)` - PVP 开始通知（含地图数据）
- `PACKET_PVP_ACTION (42)` - PVP 操作（兵种、坐标）
- `PACKET_PVP_END (43)` - PVP 结束

---

2. 玩家列表系统

功能：

- 获取所有在线玩家（排除自己）
- 显示玩家的名称、等级、资源、奖杯
- 作为选择攻击目标的源

使用示例：

```
// 请求玩家列表
socketClient.requestUserList();

// 注册回调
socketClient.setOnUserListReceived([](const std::string& userListJson) {
    // 解析 JSON：playerid,username,thLevel,gold,elixir|...
    // 显示在 UI 列表中
});
```

数据格式：

```
REQ_USER_LIST (5) - 请求
RESP_USER_LIST (6) - 响应

响应数据格式:
userId1,username1,1,1000,1000|userId2,username2,2,2000,2000|...
```

---

3. 部落系统

核心功能：

创建部落

```
socketClient.createClan("My Clan");
socketClient.setOnClanCreated([](bool success, const std::string& clanId) {
    if (success) {
        // 部落创建成功，保存 clanId
    }
});
```

加入部落

```
socketClient.joinClan("CLAN_123456");
socketClient.setOnClanJoined([](bool success) {
    if (success) {
        // 成功加入部落
    }
});
```

获取部落列表

```
socketClient.getClanList();
socketClient.setOnClanList([](const std::vector<ClanInfoClient>& clans) {
    for (const auto& clan : clans) {
        // 显示部落：名称、成员数、总奖杯、加入条件
    }
});
```

获取部落成员

```
socketClient.getClanMembers("CLAN_123456");
socketClient.setOnClanMembers([](const std::string& membersJson) {
    // 显示成员列表，标记在线/离线状态
});
```

关键数据包：

- `PACKET_CREATE_CLAN (20)` - 创建部落
- `PACKET_JOIN_CLAN (21)` - 加入部落
- `PACKET_LEAVE_CLAN (22)` - 离开部落
- `PACKET_CLAN_LIST (23)` - 获取部落列表
- `PACKET_CLAN_MEMBERS (24)` - 获取成员列表

---

4. 部落战争

流程：

```
第 1 阶段：搜索匹配
  └ 部落首领发起 searchClanWar()
      └ 服务器加入部落战队列
      └ 系统自动匹配奖杯相近的部落

第 2 阶段：战争确认
  └ 两个部落的所有在线成员收到通知
      └ 显示对手部落信息

第 3 阶段：攻击
  └ 部落成员选择目标部落成员
      └ 发起 attackInClanWar(warId, targetId)
      └ 获取对方地图数据
      └ 进行对战（同 PVP）

第 4 阶段：统计结果
  └ 每次攻击完成，上报结果
      └ 服务器累计星数
      └ 实时更新两边星数
      └ 24小时后自动结算
```

使用示例：

```
// 搜索部落战
socketClient.searchClanWar();

// 收到匹配通知
socketClient.setOnClanWarMatch([](const std::string& warId, const std::string&
clan1Id, const std::string& clan2Id) {
    // 显示对手部落信息
});

// 选择目标进行攻击
socketClient.attackInClanWar(warId, targetMemberId);

// 战斗完成后上报结果
AttackResult result;
result.attackerId = playerId;
result.defenderId = targetId;
result.starsEarned = 2;
result.goldLooted = 500;
result.trophyChange = 10;
socketClient.submitClanWarResult(warId, result);

// 监听战争状态
socketClient.setOnClanWarStatus([](const std::string& warId, int clan1Stars, int
clan2Stars) {
    // 更新 UI 显示当前星数
});
```

关键数据包：

- `PACKET_CLAN_WAR_SEARCH (30)` - 搜索部落战
- `PACKET_CLAN_WAR_MATCH (31)` - 匹配成功通知
- `PACKET_CLAN_WAR_ATTACK (32)` - 获取目标地图
- `PACKET_CLAN_WAR_RESULT (33)` - 上报攻击结果
- `PACKET_CLAN_WAR_STATUS (34)` - 战争状态更新

---

5. 对战观战

功能：允许第三方玩家实时观看两名玩家的对战过程

使用示例：

```
// 请求观战某个在线玩家
socketClient.requestSpectate("player_id_123");

// 收到观战许可
socketClient.setOnSpectateJoin([](bool success, const std::string& attackerId,
```

```
const std::string& defenderId, const std::string& mapData) {
    if (success) {
        // 进入观战模式
        // 显示地图和双方动作
        BattleManager::getInstance()->setNetworkMode(true, false); // 观战模式
    }
});

// 实时接收对战双方的操作
socketClient.setOnPvpAction([](int unitType, float x, float y) {
    // 在观战界面同步显示
});
```

关键数据包：

- `PACKET_SPECTATE_REQUEST (44)` - 请求观战
- `PACKET_SPECTATE_JOIN (45)` - 加入观战通知

服务器部署

1. 环境要求

项目	要求
操作系统	Windows 7/8/10/11 或 Windows Server 2012+
编译器	Visual Studio 2015+ 或 MinGW（GCC 5.0+）
C++ 标准	C++14
依赖库	WinSock2（Windows 自带）
网络	外网访问权限（允许入站 TCP 8888 端口）

2. 编译服务器

```
# 使用 Visual Studio
cd proj.win32
# 在 Visual Studio 中打开 Server.vcxproj
# 选择 Release 配置
# 按 F5 或菜单 Build -> Build Solution

# 或使用命令行编译
cd ..\Server
cl /O2 /EHsc Server.cpp ServerMain.cpp /link Ws2_32.lib
```

3. 运行服务器

```
```

```
# 直接运行编译后的可执行文件
.\Server.exe

# 或在 VS 中调试运行
# 输出示例：
# === Clash of Clans Server ===
# Server started on port 8888
# Waiting for players...
```

4. 网络配置

本地测试：

- 服务器地址：`127.0.0.1` 或 `localhost`
- 端口：`8888`

局域网测试：

- 查询服务器 IP：`ipconfig` 找到 IPv4 地址（如 `192.168.1.100`）
- 客户端连接地址：`192.168.1.100:8888`

公网部署：

1. **端口转发**（路由器）
    - 将外网 TCP 8888 映射到服务器内网 IP 的 8888 端口
2. **动态 DNS**（可选）
    - 如果服务器 IP 不固定，使用 DDNS 服务
3. **防火墙**
    - 开放 Windows 防火墙的入站规则
    - 允许 `Server.exe` 访问网络

5. 监控与日志

服务器输出关键信息到控制台：

```
[Login] User: player_1 (Trophies: 500)
[Match] Player player_1 joined queue (Trophies: 500)
[Match] Matched: player_1 vs player_2
[Clan] Created: My Clan (ID: CLAN_123456) by player_1
[ClanWar] Clan CLAN_123456 searching for war...
[ClanWar] Started: Clan_A vs Clan_B
[PVP] Started: player_1 vs player_2
[PVP] player_3 is spectating player_1 vs player_2
[Battle] Result - Stars: 2, Gold: 500, Trophies: 10
```

6. 性能参数（可调）

在 `Server.cpp` 中修改：

```
// 匹配范围
int maxDiff = 200 + static_cast<int>(waitTime * 10);  // 奖杯差距

// 部落战匹配
if (diff <= 500) { /* 部落奖杯差距 */ }

// 战斗时间
float _battleTime = 180.0f;  // 180 秒对战时间
```

客户端使用

1. 环境要求

项目	要求
操作系统	Windows 7+ 或 Android 5.0+
引擎	Cocos2d-x 3.x 或 4.x
编译器	Visual Studio 或 Android Studio
C++ 标准	C++14

2. 初始化网络

```
// 在场景中
auto socketClient = SocketClient::getInstance();

// 注册回调
socketClient.setOnConnected([](bool success) {
    if (success) {
        CCLOG("Connected to server");
        // 发送登录消息
    }
});

// 连接服务器
socketClient.connect("192.168.1.100", 8888);  // 修改为实际服务器 IP

// 登录
socketClient.login("player_001", "MyPlayer", 500);
```

3. UI 集成

玩家列表界面

```
// 打开玩家列表
socketClient.requestUserList();

socketClient.setOnUserListReceived([this](const std::string& userListJson) {
    // 解析玩家数据
    std::vector<PlayerInfo> players;
    // ... 解析逻辑 ...

    // 创建并显示列表
    auto layer = PlayerListLayer::create(players);
    layer->setOnPlayerSelected([this](const std::string& playerId) {
        // 用户选择了一个玩家，发起 PVP
        SocketClient::getInstance()->requestPvp(playerId);
    });
    this->addChild(layer);
});
```

部落界面

```
// 打开部落列表
socketClient.getClanList();

socketClient.setOnClanList([this](const std::vector<ClanInfoClient>& clans) {
    // 显示部落列表
    // 允许用户加入或创建部落
});

// 创建部落
socketClient.createClan("My Clan Name");

// 查看部落成员
socketClient.getClanMembers(clanId);
```

对战界面

```
// 对战开始
socketClient.setOnPvpStart([this](const std::string& role, const std::string&
opponentId, const std::string& mapData) {
    // 初始化对战管理器
    auto battleMgr = BattleManager::getInstance();

    if (role == "ATTACK") {
        battleMgr->setNetworkMode(true, true);  // 网络攻击模式
    } else {
        battleMgr->setNetworkMode(true, false);  // 网络防守模式
```

```
    }

    // 切换到对战场景
    auto scene = BattleScene::create(mapData, opponentId, true);
    Director::getInstance()->pushScene(scene);
});

// 在对战中同步操作
socketClient.setOnPvpAction([this](int unitType, float x, float y) {
    // 显示对方的单位部署
    battleMgr->deployUnitRemote((UnitType)unitType, Vec2(x, y));
});

// 对战结束
socketClient.setOnPvpEnd([this](const std::string& result) {
    // 显示战斗结果，返回主菜单
});
```

4. 完整流程示例

```
// ===== GameScene.cpp =====

bool GameScene::init() {
    // ... UI 初始化 ...

    auto socketClient = SocketClient::getInstance();

    // 设置所有回调
    socketClient.setOnConnected([this](bool success) {
        if (success) socketClient.login(userId, userName, trophies);
    });

    socketClient.setOnLoginResult([this](bool success, const std::string& msg) {
        if (success) {
            CCLOG("Login successful");
            // 上传或查询地图
            socketClient.uploadMap(mapData);
        }
    });

    socketClient.setOnUserListReceived([this](const std::string& userList) {
        // 显示玩家列表
        showPlayerList(userList);
    });

    socketClient.setOnPvpStart([this](const std::string& role, const std::string&
opponentId, const std::string& mapData) {
        // 进入对战
        enterBattle(role, opponentId, mapData);
    });
```

```
    socketClient.setOnPvpEnd([this](const std::string& result) {
        // 对战结束
        exitBattle(result);
    });

    // 连接服务器
    socketClient.connect("127.0.0.1", 8888);

    // 在 update 中处理回调
    schedule([this](float dt) {
        SocketClient::getInstance()->processCallbacks();
    }, 0.016f, "socket_callback");

    return true;
}

void GameScene::attackPlayer(const std::string& targetId) {
    SocketClient::getInstance()->requestPvp(targetId);
}
```

网络通信协议

数据包结构

```
struct PacketHeader {
    uint32_t type;      // 消息类型（1-45）
    uint32_t length;    // 消息体长度
};

// 完整消息 = PacketHeader (8字节) + 消息体 (length 字节)
```

消息类型完整列表

类型	ID	方向	说明
基础功能			
PACKET_LOGIN	1	双向	玩家登录
PACKET_UPLOAD_MAP	2	C→S	上传地图数据
PACKET_QUERY_MAP	3	双向	查询地图
PACKET_ATTACK_DATA	4	C→S	上传对战数据
REQ_USER_LIST	5	C→S	请求玩家列表
RESP_USER_LIST	6	S→C	返回玩家列表
匹配系统			

类型	ID	方向	说明
PACKET_FIND_MATCH	10	C→S	请求匹配
PACKET_MATCH_FOUND	11	S→C	匹配成功
PACKET_MATCH_CANCEL	12	C→S	取消匹配
PACKET_ATTACK_START	13	双向	开始对战
PACKET_ATTACK_RESULT	14	双向	对战结果
PACKET_BATTLE_REPLAY	15	C→S	战斗回放
部落系统			
PACKET_CREATE_CLAN	20	C→S	创建部落
PACKET_JOIN_CLAN	21	C→S	加入部落
PACKET_LEAVE_CLAN	22	C→S	离开部落
PACKET_CLAN_LIST	23	双向	部落列表
PACKET_CLAN_MEMBERS	24	双向	部落成员
PACKET_CLAN_INFO	25	双向	部落信息
部落战争			
PACKET_CLAN_WAR_SEARCH	30	C→S	搜索部落战
PACKET_CLAN_WAR_MATCH	31	S→C	部落战匹配
PACKET_CLAN_WAR_ATTACK	32	双向	部落战攻击
PACKET_CLAN_WAR_RESULT	33	C→S	部落战结果
PACKET_CLAN_WAR_STATUS	34	S→C	战争状态
PVP & 观战			
PACKET_PVP_REQUEST	40	C→S	请求 PVP
PACKET_PVP_START	41	S→C	PVP 开始
PACKET_PVP_ACTION	42	双向	PVP 操作
PACKET_PVP_END	43	双向	PVP 结束
PACKET_SPECTATE_REQUEST	44	C→S	请求观战
PACKET_SPECTATE_JOIN	45	S→C	加入观战

消息格式示例

1. 登录 (PACKET_LOGIN = 1)

```
请求（C→S）：
  Header: type=1, length=25
  Body: "player_001|MyPlayer|500"

响应（S→C）：
  Header: type=1, length=14
  Body: "Login Success"
```

2. PVP 请求 (PACKET_PVP_REQUEST = 40)

```
请求（C→S）：
  Header: type=40, length=10
  Body: "player_002"  // 目标玩家 ID

响应（S→C）：
  Header: type=41, length=50+mapSize
  Body: "ATTACK|player_002|<base64 mapData>"  // 攻击者收到地图

      OR
  Body: "DEFEND|player_001"  // 防御者收到通知

      OR
  Body: "FAIL|OFFLINE"  // 目标离线
```

3. 用户列表 (REQ_USER_LIST = 5)

```
请求（C→S）：
  Header: type=5, length=0
  Body: "" (空)

响应（S→C）：
  Header: type=6, length=100+
  Body: "id1,name1,1,1000,1000|id2,name2,2,2000,2000|..."
```

4. 部落列表 (PACKET_CLAN_LIST = 23)

```
请求（C→S）：
  Header: type=23, length=0
  Body: "" (空)

响应（S→C）：
  Header: type=23, length=200+
  Body: "[{\"id\":\"CLAN_123\",\"name\":\"My
Clan\",\"members\":5,\"trophies\":1000,\"required\":0,\"open\":true},...] "
```

常见问题

Q1: 如何快速本地测试？

A: 按以下步骤：

1. **启动服务器**

```
cd ..\Server
# 编译并运行 Server.exe
# 应看到 "Waiting for players..."
```

2. **修改客户端连接地址**

```
socketClient.connect("127.0.0.1", 8888);  // 改为 localhost
```

3. **打包 APK/EXE 并运行至少 2 个客户端实例**
   - 可在编辑器中运行，也可独立部署
   - 每个客户端用不同的 playerId 登录
4. **在一个客户端请求玩家列表**
   - 应该看到另一个玩家
   - 点击发起 PVP

Q2: 连接超时或连接被拒绝？

A: 检查以下几点：

1. **服务器是否运行**

```
netstat -ano | findstr :8888  // 检查 8888 端口
```

2. **防火墙配置**
   - Windows Defender 防火墙可能阻止了 Server.exe
   - 手动添加入站规则：设置 → 防火墙 → 允许应用
3. **IP 地址和端口**
   - 本地：`127.0.0.1:8888`
   - 局域网：查询服务器 IP（`ipconfig`）
   - 公网：使用公网 IP 或域名

Q3: 对战中下兵操作卡顿？

A:

1. **增加网络刷新频率**

```
// 在客户端更新循环中减少间隔
schedule([this](float dt) {
    SocketClient::getInstance()->processCallbacks();
}, 0.008f);  // 8ms 而不是 16ms
```

2. **减少发送的数据量**
   - 只在关键帧发送操作，而非每帧
3. **使用 UDP 代替 TCP**（可选优化）
   - 目前实现基于 TCP，可改进为 UDP 来减少延迟

Q4: 如何扩展到支持更多玩家？

A:

1. **数据库集成**
   - 将玩家数据、部落数据保存到 SQLite/MySQL
   - 实现玩家离线后的数据持久化
2. **使用 C++ ORM 库**

```
#include "sqlitecpp/SQLiteCpp.h"  // SQLiteCpp
// 保存玩家数据到数据库
```

3. **增加服务器线程池**
   - 目前每个连接一个线程，可改为线程池管理
   - 使用 `boost::asio` 或 `asio` 库
4. **地理分布式部署**
   - 多个地区部署服务器副本
   - 使用 Redis 同步玩家状态

Q5: 如何实现战斗录制和回放？

A:

1. **记录每个单位的行动**

```
struct UnitAction {
    float timestamp;
```

```
        UnitType type;
        Vec2 position;
        int unitId;
    };
    std::vector<UnitAction> battleLog;
```

2. **保存到服务器**

```
// 对战结束时发送完整日志
socketClient.submitAttackResult(result, battleLog);
```

3. **回放实现**

```
void BattleReplayScene::update(float dt) {
    for (auto& action : battleLog) {
        if (currentTime >= action.timestamp) {
            deployUnit(action.type, action.position);
        }
    }
    currentTime += dt;
}
```

Q6: 如何处理网络断连？

A:

```
socketClient.setOnDisconnected([this]() {
    CCLOG("Disconnected from server");

    // 如果在对战中
    if (battleState == FIGHTING) {
        // 暂停对战
        pauseBattle();
        // 显示重连对话框
        showReconnectDialog();
    } else {
        // 返回主菜单
        returnToMainMenu();
    }
});

// 重连逻辑
void reconnect() {
    auto socketClient = SocketClient::getInstance();
    socketClient.connect(serverIp, serverPort);
}
```

Q7: 如何调试网络问题？

A:

1. **启用日志输出**

```
#define DEBUG_NETWORK 1

#ifdef DEBUG_NETWORK
CCLOG("[NETWORK] Sending packet type %d, length %d", type, length);
#endif
```

2. **使用网络抓包工具**

   - Wireshark：监控 8888 端口的数据包
   - 验证数据格式是否正确

3. **在服务器添加详细日志**

```
std::cout << "[DEBUG] Received packet type=" << msgType
          << " length=" << msgData.length()
          << " from " << clientSocket << std::endl;
```

总结

本文档涵盖了联网对战系统的以下方面：

? **系统架构** - 客户端-服务器通信模式
? **功能模块** - PVP、部落、战争、观战等完整功能
? **服务器部署** - 从编译到公网部署的全流程
? **客户端集成** - UI 和游戏逻辑的完整示例
? **网络协议** - 所有消息类型和格式规范
? **常见问题** - 快速排查和优化建议

如有任何问题，请参考代码注释或提交 Issue。

文档版本：v1.0
更新日期：2025/12/14