

**课 程 设 计 报 告**

**课程名称:** C#高级程序设计

**设计题目** 3D坦克大战联网对战版之服务器端

**院（系）:** 计算机学院（软件学院）

**专业年级:**软件工程（游戏开发工程师）2017级

**学 号:** 171530425

**姓 名:** 徐 可 可

**指导教师:** 彭 伟 国

**2019 年 12 月 28 日**

目 录

[1 系统需求分析 3](#_Toc27900213)

[1.1项目意义 3](#_Toc27900214)

[1.2系统需求分析 3](#_Toc27900215)

[**1.2.1**系统需求调查研究 3](#_Toc27900216)

[**1.2.2**系统需求调查结果 4](#_Toc27900217)

[**1.2.3**系统功能设计要求 5](#_Toc27900218)

[2系统概要设计 6](#_Toc27900219)

[2.1登录注册功能 6](#_Toc27900220)

[3系统详细设计 7](#_Toc27900221)

[3.1记事本 7](#_Toc27900222)

[3.1注册功能 7](#_Toc27900223)

[3.1登录功能 7](#_Toc27900224)

[3.2系统调试及解决方法 8](#_Toc27900225)

[4系统运行结果 9](#_Toc27900226)

[**5**项目评价 13](#_Toc27900227)

[参考文献 13](#_Toc27900228)

[附录：源代码 14](#_Toc27900229)

# 1 系统需求分析

## 1.1项目意义

对于我本人来说，开发调试一款较为复杂的项目，可以充分锻炼我各方面的能力。

首先锻炼的就是我对所使用的开发语言，C#掌握的能力。其次，由于该游戏基于Unity开发，也涉及到网络和数据库以及团队协同开发的操作，所以也会锻炼我对Unity、SQL、Git、TCP等的掌握和综合应用能力。

另外由于本游戏是开源的，任何人都能够获取到本游戏的源代码，并且里面不乏有一些优秀的设计之处，所以能够给其他开发者提供借鉴。

## 1.2系统需求分析

**1.2.1**系统需求调查研究

开发一款网络游戏，必不可少的要开发服务端，而开发服务端并不容易，首先就是需要有好的架构。



图1-1 百度搜索服务器端架构

软件架构的分析，可以通过不同的层面入手。比较经典的软件架构描述，包含了以下几种架构：运行时架构、逻辑架构、物理架构、数据架构、开发架构。



图1-2 常见的架构描述

**1.2.2**系统需求调查结果

服务器端软件的本质，是一个会长期运行的程序，并且它还要服务于多个不定时，不定地点的网络请求。所以这类软件的特点是要非常关注稳定性和性能。这类程序如果需要多个协作来提高承载能力，则还要关注部署和扩容的便利性；同时，还需要考虑如何实现某种程度容灾需求。由于多进程协同工作，也带来了开发的复杂度，这也是需要关注的问题。

功能约束，是架构设计决定性因素。一个万能的架构，必定是无能的架构。一个优秀的架构，则是正好把握了对应业务领域的核心功能产生的。游戏领域的功能特征，于服务器端系统来说，非常明显的表现为几个功能的需求：

* 对于游戏数据和玩家数据的存储
* 对玩家客户端进行数据广播
* 把一部分游戏逻辑在服务器上运算，便于游戏更新内容，以及防止外挂。

针对以上的需求特征，在服务器端软件开发上，我们往往会关注软件对电脑内存和CPU的使用，以求在特定业务代码下，能尽量满足承载量和响应延迟的需求。最基本的做法就是“时空转换”，用各种缓存的方式来开发程序，以求在CPU时间和内存空间上取得合适的平衡。在CPU和内存之上，是另外一个约束因素：网卡。网络带宽直接限制了服务器的处理能力，所以游戏服务器架构也必定要考虑这个因素。

对于游戏服务器架构设计来说，最重要的是利用游戏产品的需求约束，从而优化出对此特定功能最合适的“时—空”架构。并且最小化对网络带宽的占用。

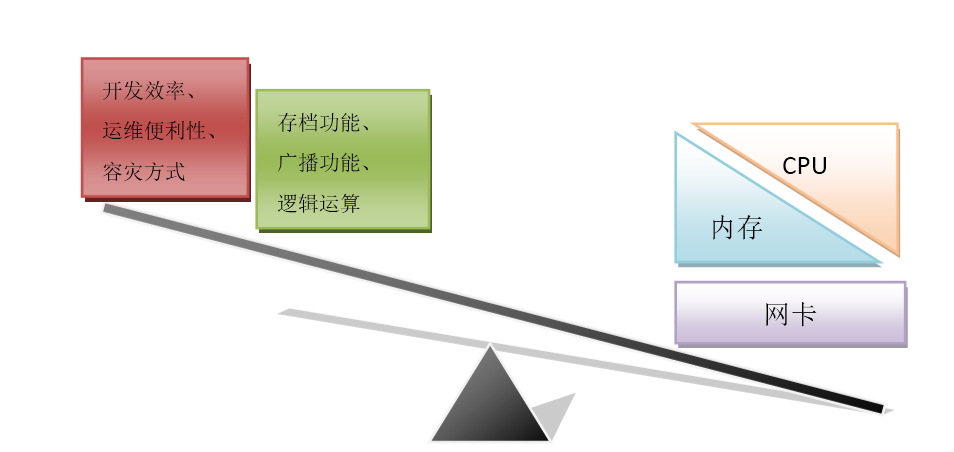


图1-3 游戏服务器的分析模型

**1.2.3**系统功能设计要求

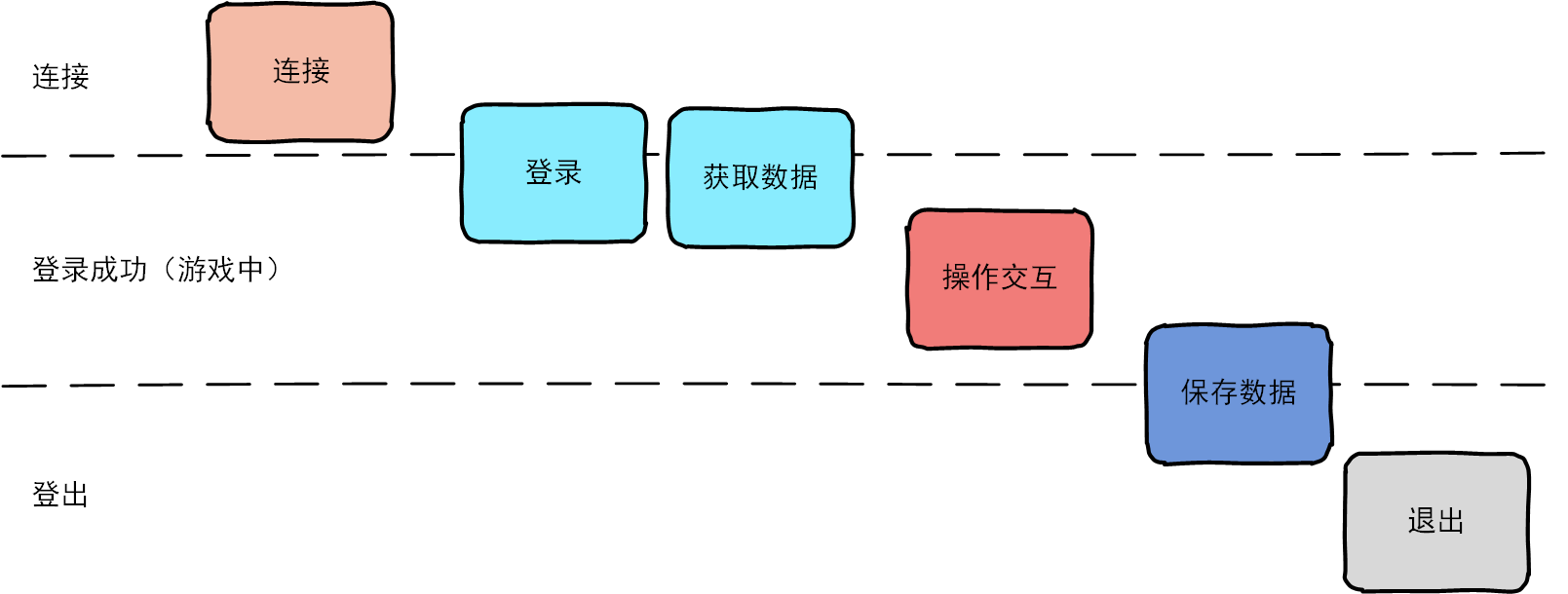
从服务端的角度看，一个玩家会经历连接、登录、获取数据、操作交互、保存数据和退出六个阶段，如图1-4所示。

图1-4 游戏流程

* **连接阶段**：客户端调用Connect连接服务端即为连接阶段。连接后双端即可通信，但服务端还不知道玩家控制的是哪个角色。于是客户端需要发送一条登录协议，协议中包含用户名、密码等信息，待检验通过后服务端会将网络连接与游戏角色对应起来，从数据库中获取该角色的数据后，才算登录成功。
* **交互阶段**：双端互通协议。MsgMove、MsgAttack，记事本程序的保存文本功能，都发生在这一阶段。
* **登出阶段**：玩家下线，服务端把玩家的数据保存到数据库中。对于保存玩家数据的时机，不同的服务端会有不同实现。有些服务端采用定时存储的方式，每隔几分钟把在线玩家的数据写回数据库；有些服务端采用下线时存储的方式，只有在玩家下线时才保存数据。上述方式各有优缺点，定时存储相对于下线时存储安全，在服务端突然挂掉的情况下，能够挽回一部分在线玩家数据，但也因为要频繁写数据库，性能较差。本系统采用玩家下线时才保存数据的方式。

对应于上述几个步骤，一个连接会有“连接但未登录”和“登录成功”两种状态，如表1-1所示。

表1-1 连接状态

# 2系统概要设计

## 2.1登录注册功能

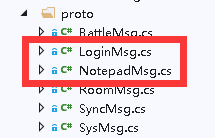
在服务端程序中为proto文件夹添加LoginMsg.cs和NotepadMsg.cs两个文件，用于定义登录和记事本相关的协议，如图2-1所示。

图2-1 添加LoginMsg.cs和NotepadMsg.cs两个文件

LoginMsg中包含了注册、登录和踢出三条协议。MsgRegister即注册协议，客户端需要发送id和pw字段，指定要注册的用户名和密码。服务端处理消息后，也会给客户端回应MsgRegister协议，如果服务端回应的result为0，代表注册成功，如果为1。

MsgLogin即登录协议，客户端也需要发送id和pw字段，指定要登录的用户名及其密码。服务端收到消息后，会判断密码是否正确，然后加载玩家数据，回应客户端。如果服务端回应的result为0,代表登录成功，如果为1，代表登录失败。

MsgKick是由服务端推送的“强制下线”协议。游戏中常有多个客户端同时登录同一个账号的情况，后登录的客户端会把早前登录客户端踢下线。服务端会给早前登录的客户端推送MsgKick协议，指明被踢下线的原因。

# 3系统详细设计

## 3.1记事本

在NotepadMsg.cs中编写读取和保存记事本的协议，客户端发送MsgGetText 协议后，服务端会返回带有test字段的同名协议，返回记事本文本。编辑完文本后，玩家点击保存按钮，客户端会发送MsgSaveText协议，并将修改后的文本以text字段发送给服务端。服务端收到后，更新文本，并返回同名协议。如果result为0，代表保存成功。

## 3.1注册功能

在服务端程序中添加LoginMsgHandle.cs和NotepadMsgHandle.cs两个文件， 用于处理登录注册和记事本的协议，如图3-1所示。

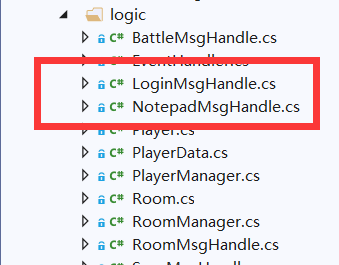


图3-1 添加LoginMsgHandle.cs和NotepadMsgHandle.cs两个文件

在LoginMsgHandle中编写MsgHandler类(partial class MsgHandler)，添加处理注册协议的方法MsgRegister。MsgRegister会调用DbManager.Register向account 表写入账号信息，再使用DbManager.CreatePlayer向game表写入默认的角色信息。最后调用NetManager. Send返回协议给客户端。

## 3.1登录功能

添加处理登录协议的方法MsgLogin,它相对复杂，因为要处理下面几项任务。

1. 验证密码：通过DbManager. CheckPassword验证用户名和密码，如果密码错误，返回result=1给客户端。
2. 状态判断：如果该客户端已经登录，不能重复登录。
3. 踢下线：通过PlayerManager.IsOnline判断该账户是否已经登录，如果已经登录，需要先把它踢下线。程序会通过PlayerManager.GetPlayer(msg.id) 获取已登录的玩家对象，给它发送MsgKick协议，通知被踢下线的客户端。最后调用NetManager.Close关闭Socket连接。
4. 读取数据：通过DbManager.GetPlayerData从数据库中读取玩家数据。
5. 构建Player：根据读取到的数据，构建player对象，并把它添加到PlayerManager的列表中，将客户端信息ClientState和player对象关联起来。

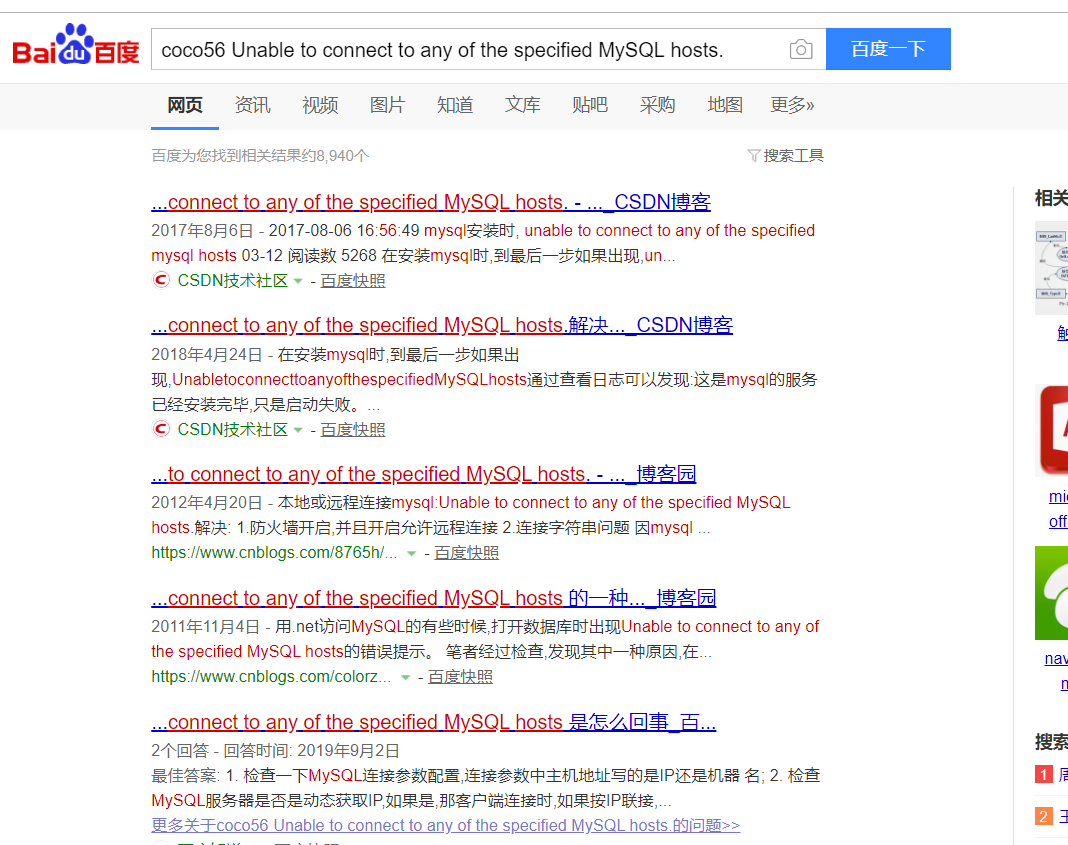
## 3.2系统调试及解决方法

实现过程中曾出现以下调试错误信息：

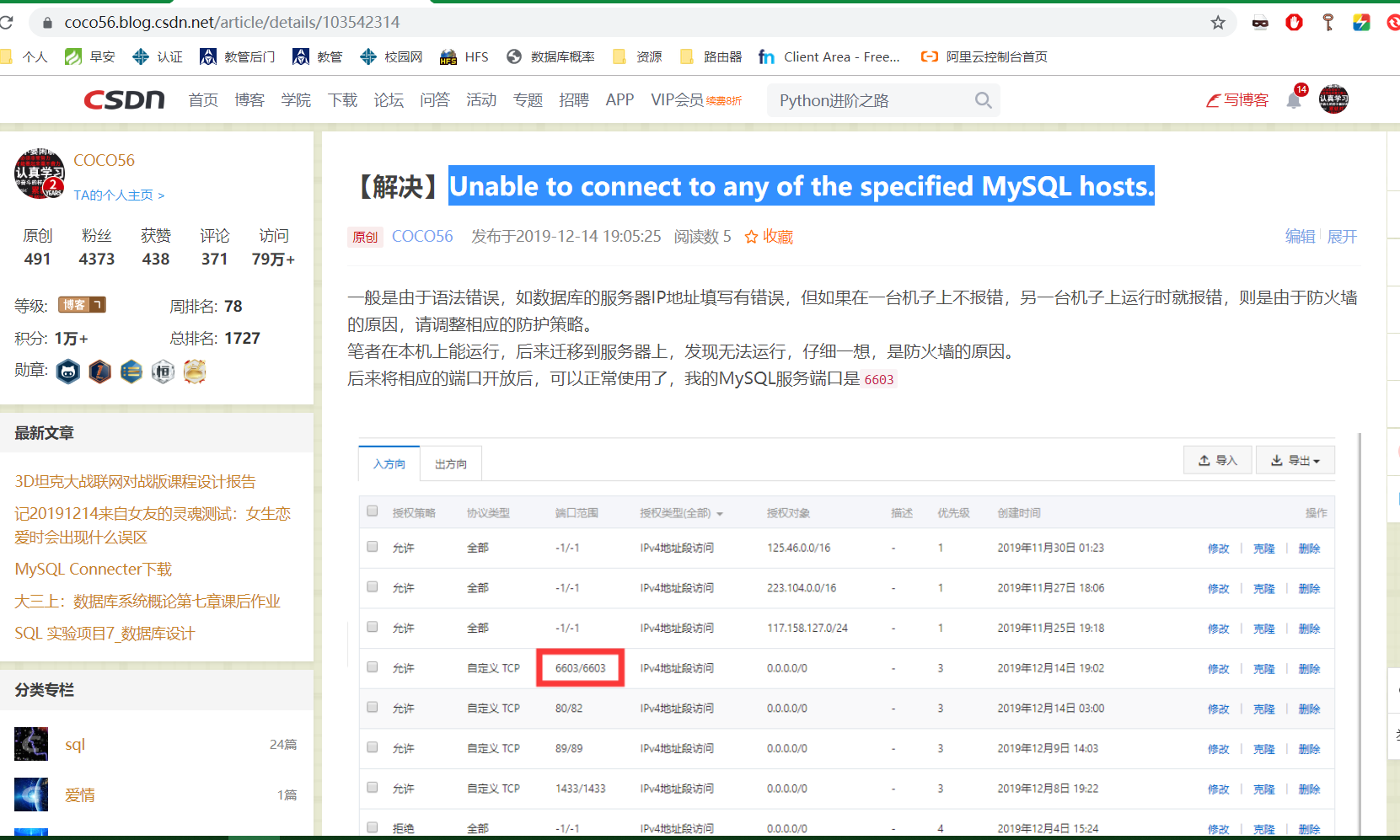
(1)、Unable to connect to any of the specified MySQL hosts.

解决方法：

通过百度搜索自己的曾经写的博文，成功解决问题。



3-2 百度搜索结果



3-3 本人自己之前写的博客

# 4系统运行结果

打开服务端应用程序，会看到如图4-1所示的界面和提示，此时代表连接数据库成功，并成功监听相应的端口，如果出错，则会闪退。

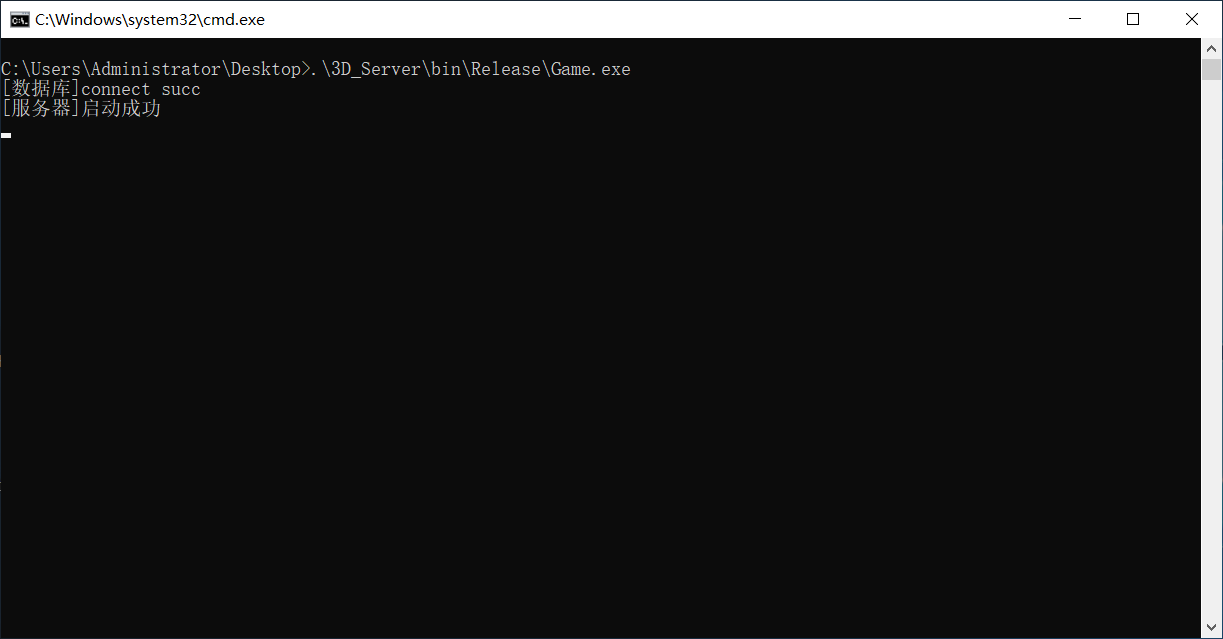


图4-1 启动成功时的提示

当有新客户端到达时，会有如图4-2所示的提示，图4-2所示的Accept 125.46.3.236:32323意为客户端的IP为125.46.3.236端口号为32323。

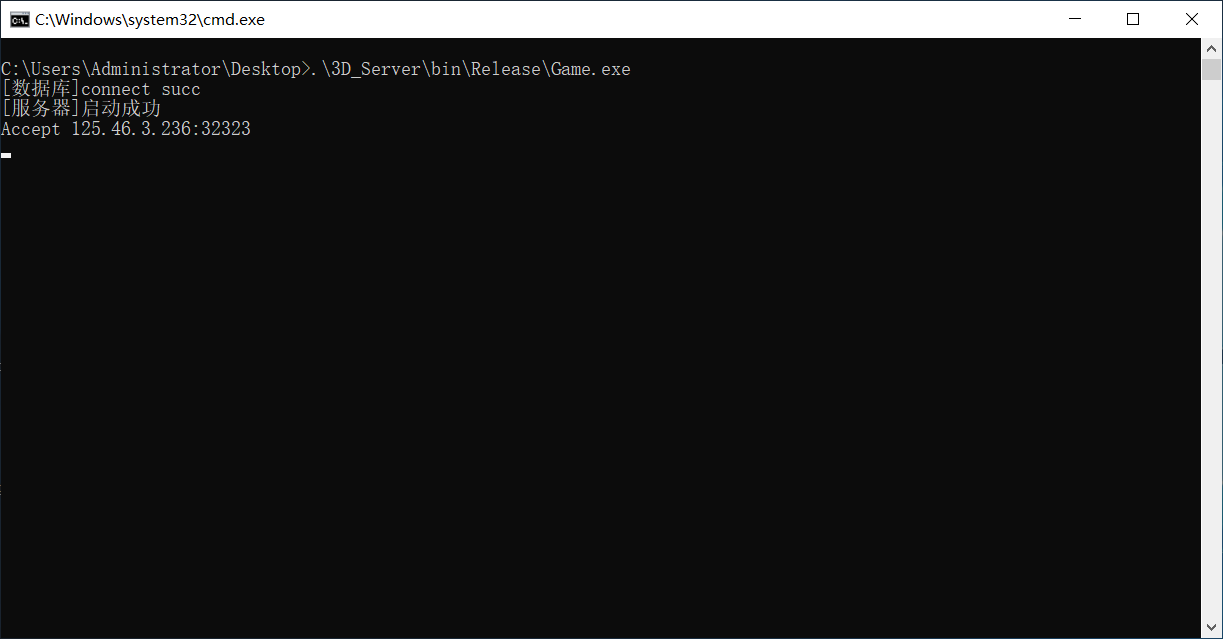


图4-2 有新客户端到达时的提示

当有客户端关闭时会有如图4-3所示的提示，图4-3所示的Socket Close 125.46.3.236:32323意为客户端的IP为125.46.3.236端口号为32323。

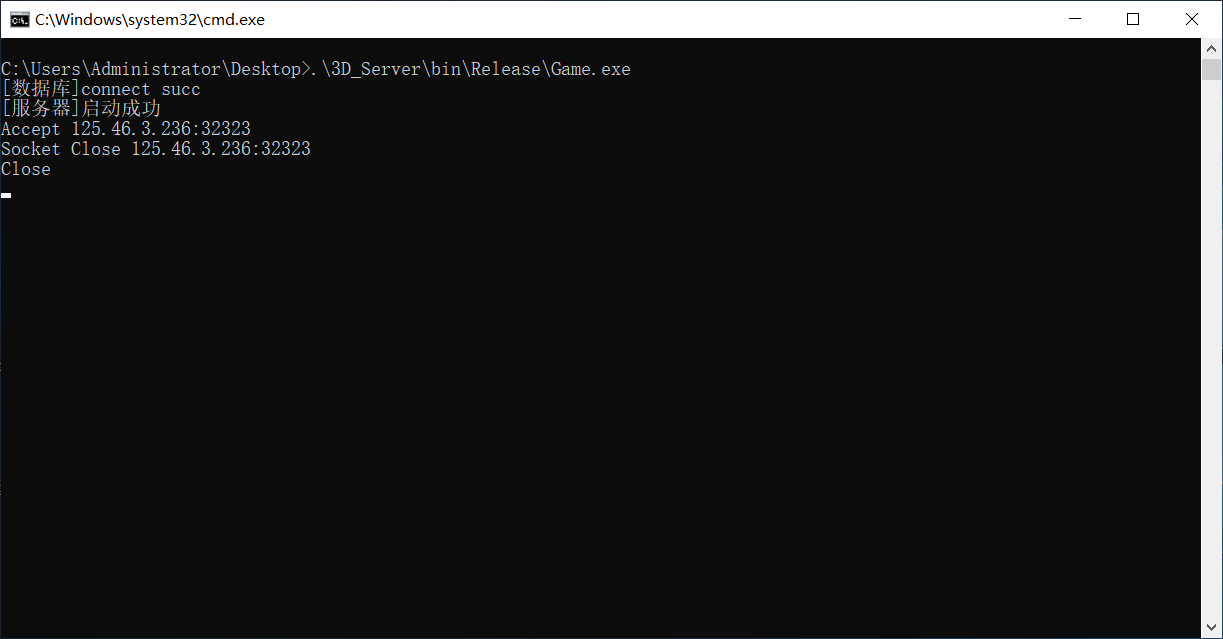


图4-3 客户端关闭时的提示

当客户端成功登录时，会自动进入房间。提示如图4-4所示。另外当客户端打开房间列表面板后，面板左侧会显示玩家的战绩（总胜利次数和总失败次数），因此进入房间时会请求查询战绩（MsgGetAchieve）；客户端面板右侧显示了房间列表，因此还会请求获取房间列表（MsgGetRoomList）；

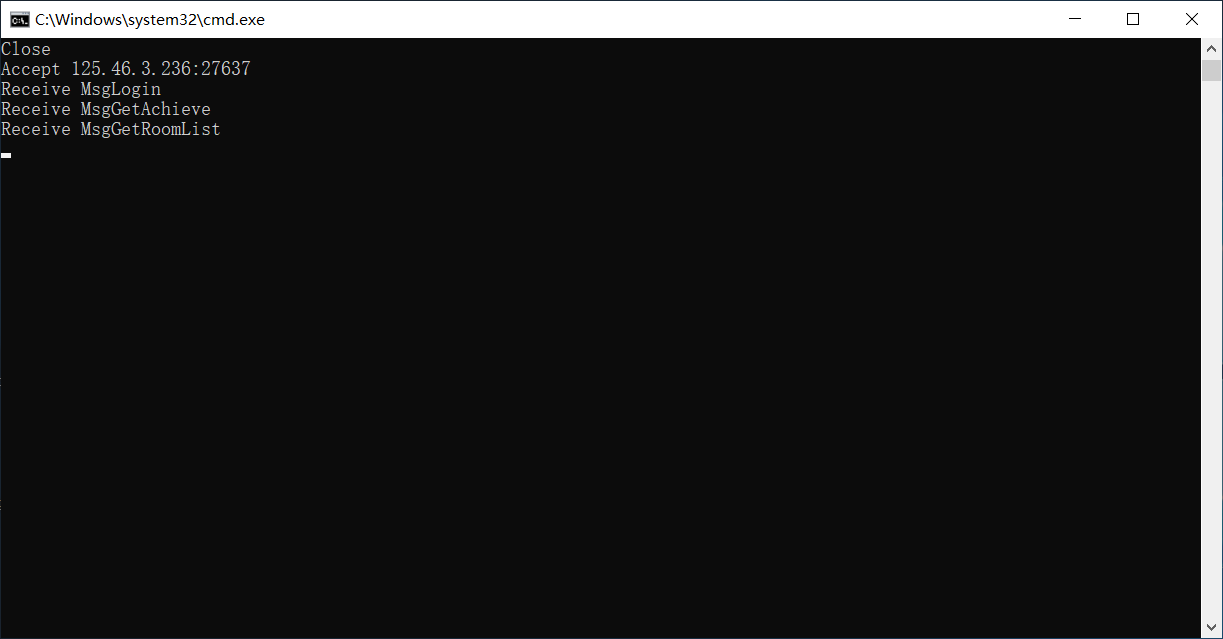


图4-4 客户端请求获取房间列表的提示

如果玩家拿着手机进入没有信号的山区，或者有人拿剪刀剪断网线，都会导致链路不通。但TCP本身的心跳机制太＂鸡肋＂，要经过2个小时的时间才能主动释放资源，游戏程序一般都会自行实现心跳机制。具体来说就是，客户端会定时（如30秒）给服务端发送PING协议，服务端收到后会回应PONG协议。正常情况下，客户端每隔一段时间（如30秒）必然会收到服务端的PONG协议（就算网络不通畅，最慢120秒也总该收到了吧）。如果客户端很长时间（如120秒）没有收到PONG协议，很大概率是网络不通畅或服务端挂掉，客户端程序可以释放Socket资源。其实对于客户端来说，释放不释放关系不大，毕竞只有一个Socket。但对服务端来说却很重要，因为服务端可能保持着数以万计的连接，当游戏在线人数很多时，只有及时释放资源，才能让玩家正常玩游戏（不然，内存爆满服务器挂掉大家都玩不了）。所以客户端会定时向服务端发送MsgPing协议，服务端收到后也会回应MsgPong协议。

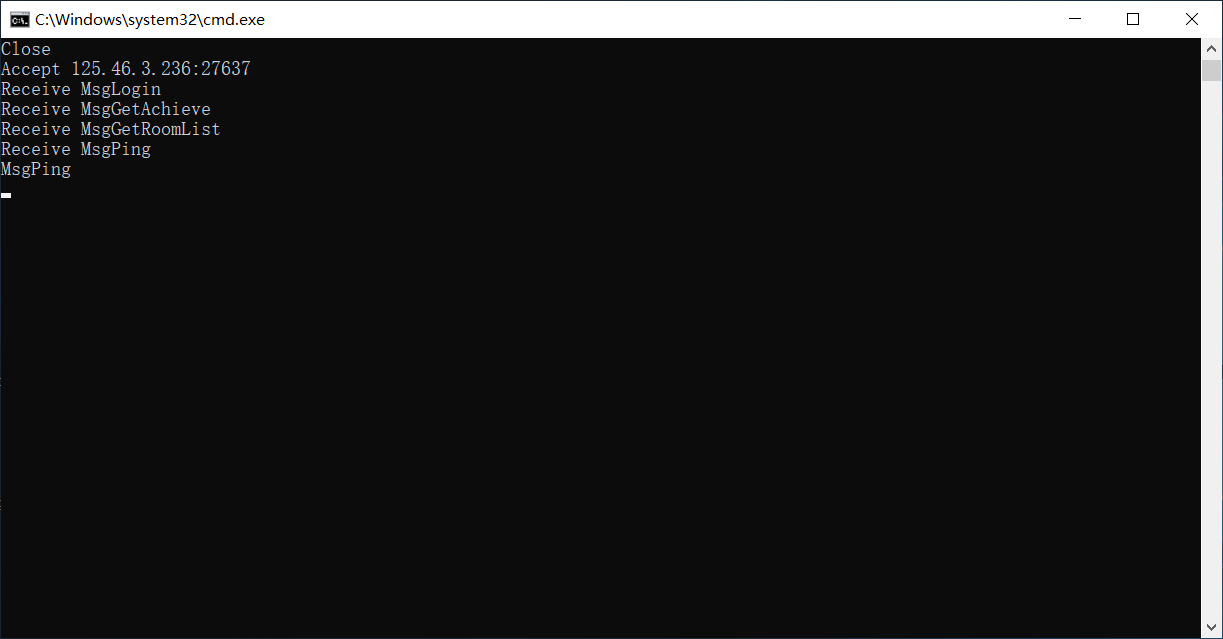


图4-5 客户端与服务端进行“乒乓”时的提示

客户端登录后面板中有“新建房间”和“加入房间”按钮，涉及MsgCreateRoom和MsgEnterRoom两条协议；若玩家加入房间，需要获取房间信息(MsgGetRoomlnfo协议）；

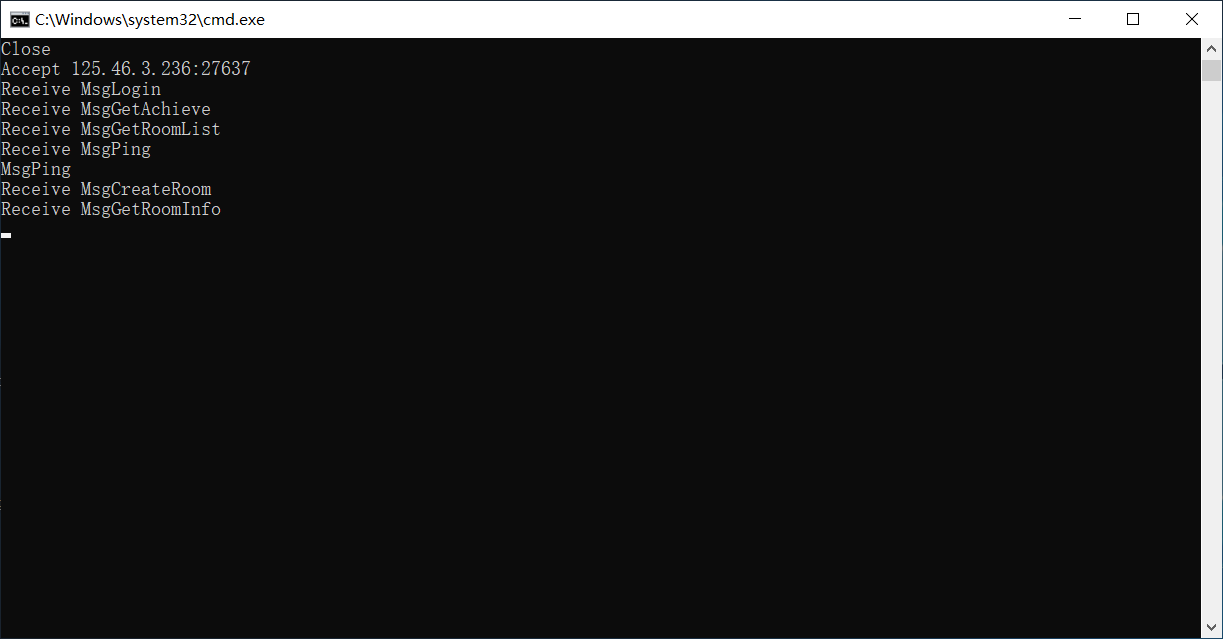


图4-6 客户端创建房间并获取房间信息的提示

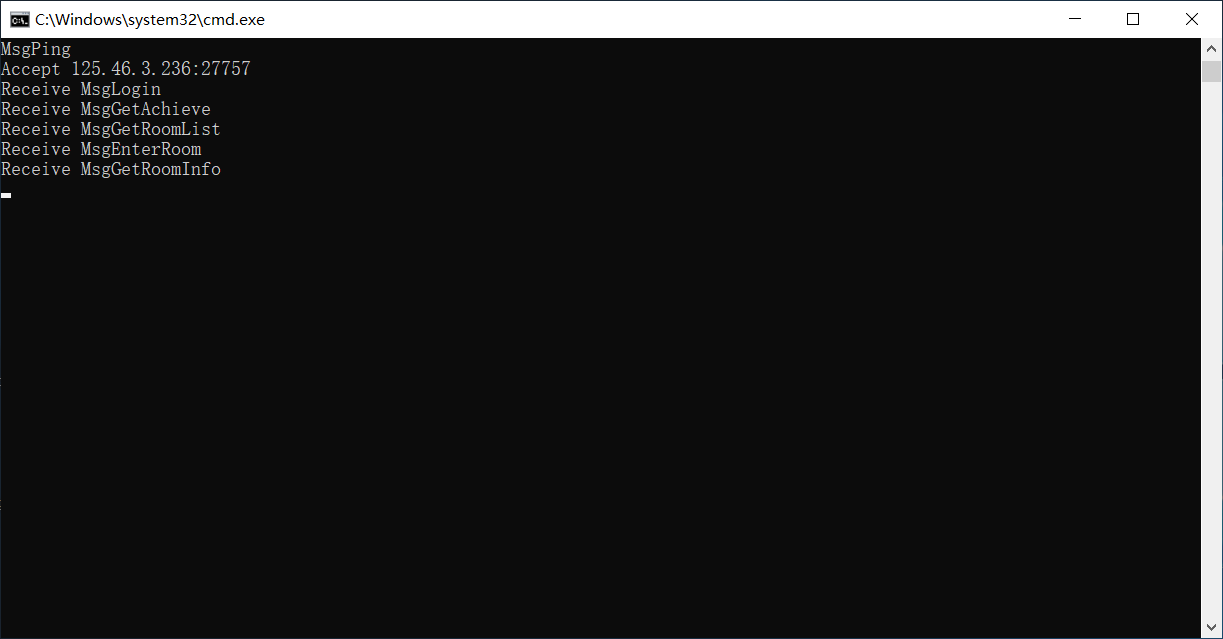


图4-7 客户端进入房间并获取房间信息的提示

战斗过程中，程序会通过MsgSyncTank,MsgFire,MsgHit等协议去同步坦克的位置、炮弹位置等信息。当某个阵营取得胜利，服务端会广播MsgBattleResult协议，通知客户端哪个阵营获得了胜利。

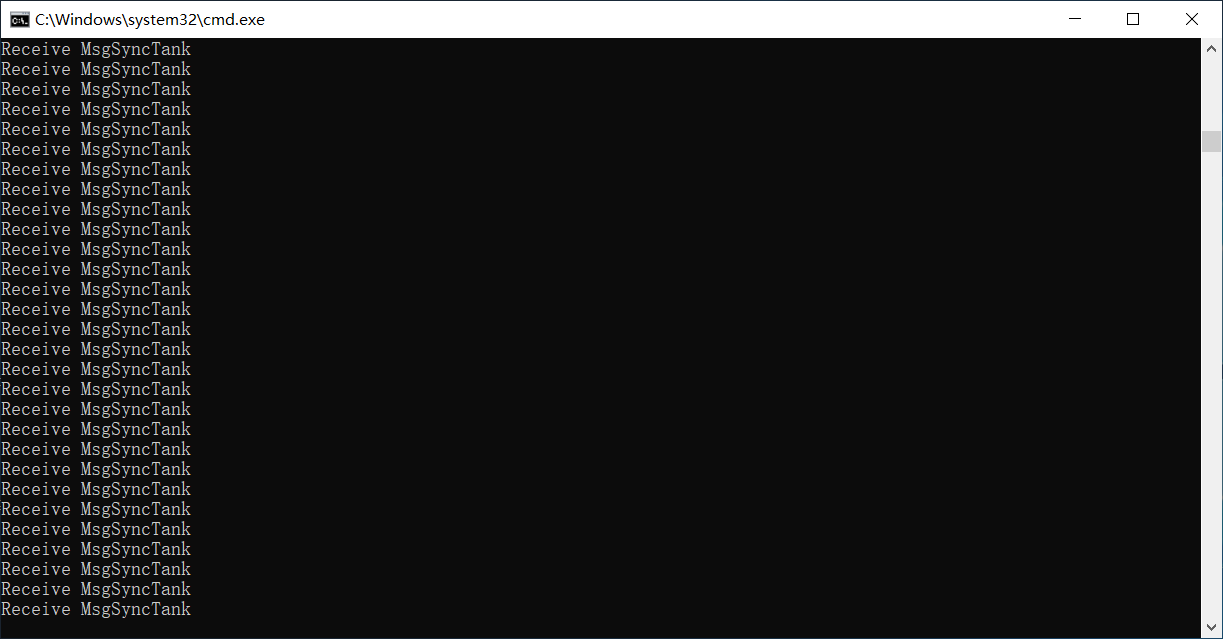


图4-8 客户端发送同步坦克信息时的提示

**5**项目评价

本游戏服务器的设计完成了基本的设计要求，但还有一些地方需要优化，比如如何处理高并发，以及如何支持在服务器集群上运行。另外为了方便用户登录，也可以接入第三方服务，比如接入QQ快捷登录，这样用户登录起来就比较方便了，无需专门注册我们的游戏账号，直接用QQ扫一下二维码就能自动登录或注册了。

具体地：

* 由于本游戏不分区不分服，所以在设计服务器的时候，应按世界服的思想去设计，即服务器是一个n多台物理机的集群。当用户登陆服务器，创建房间时，可能根据负载均衡算法，它可以在任何一台服务器上面。这样，不管用户登陆到哪一台服务器上面了，都可以获得自己的数据。可以使用redis来做数据共享。
* 在同一局游戏中，应要求所有人都在同一个房间中，可以规定在同一个房间中的用户，必须登陆到同一台物理服务器上面。在创建房间完成之后，其他人根据房间号查找房间的时候，就可以根据房间号，获取这个房间所在的服务器ip和端口，判断一个当前用户登陆的服务器ip与房间所在的服务器ip是否相同，如果相同，就不做切换，如果不一样，客户端就使用ip和端口，连接到房间所在的服务器上面。
* 创建房间成功之后，接下来的操作都要保证它的顺序性，所以房间需要有一个它自己的消息队列。可以把每个房间到达服务器的消息封装为一个任务，把这个任务放到消息队列中，然后有一个任务执行者去按顺序执行这些任务。
* 在用户登录时一般都是需要接第三方登陆（如使用QQ登录），登陆这一块是http操作，我们统一提供一个web服务，用来做登陆验证。因为在登陆时，调用第三方的http服务，这个过程可能很慢，如果放在逻辑服务器的话，可能会卡业务逻辑任务。因为可能不同的玩家业务请求可能同在一个线程中，如果有任务卡了，那么这个任务以后新来的请求请会卡住，导致消息延迟。

# 参考文献

[1] 李心蕊.独立游戏的设计艺术和文化特性研究[D].齐齐哈尔：齐齐哈尔大学，2016.

[2] 刘晓扬.基于社交网络游戏的行为干预研究[D].哈尔滨：哈尔滨工业大学,2013.

[3] 王琛瑜.移动游戏界面视觉风格的研究与设计应用[D].上海：东华大学,2017.

[4] 李绍龙.网络游戏用户行为分析和应用[D].北京：北京交通大学, 2009.

[5] 郭振兴.手机“废城赛车”游戏软件的技术研发[D].天津：天津大学,2015.

[6] 杨观.基于Unity的游戏逻辑引擎的设计与实现[D].哈尔滨：哈尔滨工业大学,2015.

[7] 周晓风.基于MAYA和VIRTOOLS的虚拟仓储物流系统的研究与实现[D].上海：上海交通大学,2011.

[8] 贺瞿. 基于动力学原理的Maya粒子特效插件[D].上海：上海交通大学,2011.

[9] 刘佳.关于photoshop应用的几个小技巧[J].信息与电脑(理论版),2013,5(03):187-188.

[10] 赵娟. 地图编辑器在苹果手机游戏设计中的开发与应用[D].北京：北京工业大学,2013.

[11] 张海藩.软件工程导论（第6版）[M].清华大学出版社,2016.

[12] 许鹏森. 游戏软件黑盒测试方法研究与应用[D].武汉：华中科技大学,2008.

附录：源代码

* 3D\_Server\DBConfiguration.cs

1. // 将此文件更名为DBConfiguration.cs，并对以下变量进行赋值
2. namespace Game
3. {
4. class DBConfiguration
5. {
6. public static string db= "game";
7. public static string ip = "127.0.0.1";
8. public static int port = 3306;
9. public static string user = "root";
10. public static string pw = "aa123bb456";
11. }
12. }

* 3D\_Server\Program.cs

1. namespace Game
2. {
3. class MainClass
4. {
5. public static void Main (string[] args)
6. {
7. if(!DbManager.Connect(DBConfiguration.db, DBConfiguration.ip, DBConfiguration.port, DBConfiguration.user, DBConfiguration.pw)){
8. return;
9. }
10. NetManager.StartLoop(82);
11. }
12. }
13. }

* 3D\_Server\script\db\DbManager.cs

1. using System;
2. using MySql.Data.MySqlClient;
3. using System.Text.RegularExpressions;
4. using System.Web.Script.Serialization;
5. public class DbManager {
6. public static MySqlConnection mysql;
7. static JavaScriptSerializer Js = new JavaScriptSerializer();
8. //连接mysql数据库
9. public static bool Connect(string db, string ip, int port, string user, string pw)
10. {
11. //创建MySqlConnection对象
12. mysql = new MySqlConnection();
13. //连接参数
14. string s = string.Format("Database={0};Data Source={1}; port={2};User Id={3}; Password={4}",
15. db, ip, port, user, pw);
16. mysql.ConnectionString = s;
17. //连接
18. try
19. {
20. mysql.Open();
21. Console.WriteLine("[数据库]connect succ ");
22. return true;
23. }
24. catch (Exception e)
25. {
26. Console.WriteLine("[数据库]connect fail, " + e.Message);
27. return false;
28. }
29. }
30. //测试并重连
31. private static void CheckAndReconnect(){
32. try{
33. if(mysql.Ping()){
34. return;
35. }
36. mysql.Close();
37. mysql.Open();
38. Console.WriteLine("[数据库] Reconnect!");
39. }
40. catch(Exception e){
41. Console.WriteLine("[数据库] CheckAndReconnect fail " + e.Message);
42. }
44. }
45. //判定安全字符串
46. private static bool IsSafeString(string str)
47. {
48. return !Regex.IsMatch(str, @"[-|;|,|\/|\(|\)|\[|\]|\}|\{|%|@|\\*|!|\']");
49. }
50. //是否存在该用户
51. public static bool IsAccountExist(string id)
52. {
53. CheckAndReconnect();
54. //防sql注入
55. if (!DbManager.IsSafeString(id)){
56. return false;
57. }
58. //sql语句
59. string s = string.Format("select \* from account where id='{0}';", id);
60. //查询
61. try
62. {
63. MySqlCommand cmd = new MySqlCommand (s, mysql);
64. MySqlDataReader dataReader = cmd.ExecuteReader ();
65. bool hasRows = dataReader.HasRows;
66. dataReader.Close();
67. return !hasRows;
68. }
69. catch(Exception e)
70. {
71. Console.WriteLine("[数据库] IsSafeString err, " + e.Message);
72. return false;
73. }
74. }
75. //注册
76. public static bool Register(string id, string pw)
77. {
78. CheckAndReconnect();
79. //防sql注入
80. if(!DbManager.IsSafeString(id)){
81. Console.WriteLine("[数据库] Register fail, id not safe");
82. return false;
83. }
84. if(!DbManager.IsSafeString(pw)){
85. Console.WriteLine("[数据库] Register fail, pw not safe");
86. return false;
87. }
88. //能否注册
89. if (!IsAccountExist(id))
90. {
91. Console.WriteLine("[数据库] Register fail, id exist");
92. return false;
93. }
94. //写入数据库User表
95. string sql = string.Format("insert into account set id ='{0}' ,pw ='{1}';", id, pw);
96. try
97. {
98. MySqlCommand cmd = new MySqlCommand(sql, mysql);
99. cmd.ExecuteNonQuery();
100. return true;
101. }
102. catch(Exception e)
103. {
104. Console.WriteLine("[数据库] Register fail " + e.Message);
105. return false;
106. }
107. }
108. //创建角色
109. public static bool CreatePlayer(string id)
110. {
111. CheckAndReconnect();
112. //防sql注入
113. if(!DbManager.IsSafeString(id)){
114. Console.WriteLine("[数据库] CreatePlayer fail, id not safe");
115. return false;
116. }
117. //序列化
118. PlayerData playerData = new PlayerData ();
119. string data = Js.Serialize(playerData);
120. //写入数据库
121. string sql = string.Format ("insert into player set id ='{0}' ,data ='{1}';", id, data);
122. try
123. {
124. MySqlCommand cmd = new MySqlCommand (sql, mysql);
125. cmd.ExecuteNonQuery ();
126. return true;
127. }
128. catch (Exception e)
129. {
130. Console.WriteLine("[数据库] CreatePlayer err, " + e.Message);
131. return false;
132. }
133. }
134. //检测用户名密码
135. public static bool CheckPassword(string id, string pw)
136. {
137. CheckAndReconnect();
138. //防sql注入
139. if(!DbManager.IsSafeString(id)){
140. Console.WriteLine("[数据库] CheckPassword fail, id not safe");
141. return false;
142. }
143. if(!DbManager.IsSafeString(pw)){
144. Console.WriteLine("[数据库] CheckPassword fail, pw not safe");
145. return false;
146. }
147. //查询
148. string sql = string.Format("select \* from account where id='{0}' and pw='{1}';", id, pw);
149. try
150. {
151. MySqlCommand cmd = new MySqlCommand (sql, mysql);
152. MySqlDataReader dataReader = cmd.ExecuteReader();
153. bool hasRows = dataReader.HasRows;
154. dataReader.Close();
155. return hasRows;
156. }
157. catch(Exception e)
158. {
159. Console.WriteLine("[数据库] CheckPassword err, " + e.Message);
160. return false;
161. }
162. }
163. //获取玩家数据
164. public static PlayerData GetPlayerData(string id)
165. {
166. CheckAndReconnect();
167. //防sql注入
168. if(!DbManager.IsSafeString(id)){
169. Console.WriteLine("[数据库] GetPlayerData fail, id not safe");
170. return null;
171. }
172. //sql
173. string sql = string.Format("select \* from player where id ='{0}';", id);
174. try
175. {
176. //查询
177. MySqlCommand cmd = new MySqlCommand (sql, mysql);
178. MySqlDataReader dataReader = cmd.ExecuteReader();
179. if(!dataReader.HasRows)
180. {
181. dataReader.Close();
182. return null;
183. }
184. //读取
185. dataReader.Read();
186. string data = dataReader.GetString("data");
187. //反序列化
188. PlayerData playerData = Js.Deserialize<PlayerData>(data);
189. dataReader.Close();
190. return playerData;
191. }
192. catch(Exception e)
193. {
194. Console.WriteLine("[数据库] GetPlayerData fail, " + e.Message);
195. return null;
196. }
197. }
198. //保存角色
199. public static bool UpdatePlayerData(string id, PlayerData playerData)
200. {
201. CheckAndReconnect();
202. //序列化
203. string data = Js.Serialize(playerData);
204. //sql
205. string sql = string.Format("update player set data='{0}' where id ='{1}';", data, id);
206. //更新
207. try
208. {
209. MySqlCommand cmd = new MySqlCommand (sql, mysql);
210. cmd.ExecuteNonQuery ();
211. return true;
212. }
213. catch (Exception e)
214. {
215. Console.WriteLine("[数据库] UpdatePlayerData err, " + e.Message);
216. return false;
217. }
218. }
219. }

* 3D\_Server\script\logic\BattleMsgHandle.cs

1. using System;
2. public partial class MsgHandler {
3. public static void MsgMove(ClientState c, MsgBase msgBase){
4. //MsgMove msgMove = (MsgMove)msgBase;
5. //Console.WriteLine(msgMove.x);
6. //msgMove.x++;
7. //NetManager.Send(c, msgMove);
8. }
9. }

* 3D\_Server\script\logic\EventHandler.cs

1. using System;
2. public partial class EventHandler
3. {
4. public static void OnDisconnect(ClientState c){
5. Console.WriteLine("Close");
6. //Player下线
7. if(c.player != null){
8. //离开战场
9. int roomId = c.player.roomId;
10. if(roomId >= 0){
11. Room room = RoomManager.GetRoom(roomId);
12. room.RemovePlayer(c.player.id);
13. }
14. //保存数据
15. DbManager.UpdatePlayerData(c.player.id, c.player.data);
16. //移除
17. PlayerManager.RemovePlayer(c.player.id);
18. }
19. }
21. public static void OnTimer(){
22. CheckPing();
23. RoomManager.Update();
24. }
25. //Ping检查
26. public static void CheckPing(){
27. //现在的时间戳
28. long timeNow = NetManager.GetTimeStamp();
29. //遍历，删除
30. foreach(ClientState s in NetManager.clients.Values){
31. if(timeNow - s.lastPingTime > NetManager.pingInterval\*4){
32. Console.WriteLine("Ping Close " + s.socket.RemoteEndPoint.ToString());
33. NetManager.Close(s);
34. return;
35. }
36. }
37. }
38. }

* 3D\_Server\script\logic\LoginMsgHandle.cs

1. using System;
2. public partial class MsgHandler {
3. //注册协议处理
4. public static void MsgRegister(ClientState c, MsgBase msgBase){
5. MsgRegister msg = (MsgRegister)msgBase;
6. //注册
7. if(DbManager.Register(msg.id, msg.pw)){
8. DbManager.CreatePlayer(msg.id);
9. msg.result = 0;
10. }
11. else{
12. msg.result = 1;
13. }
14. NetManager.Send(c, msg);
15. }
16. //登陆协议处理
17. public static void MsgLogin(ClientState c, MsgBase msgBase){
18. MsgLogin msg = (MsgLogin)msgBase;
19. //密码校验
20. if(!DbManager.CheckPassword(msg.id, msg.pw)){
21. msg.result = 1;
22. NetManager.Send(c, msg);
23. return;
24. }
25. //不允许再次登陆
26. if(c.player != null){
27. msg.result = 1;
28. NetManager.Send(c, msg);
29. return;
30. }
31. //如果已经登陆，踢下线
32. if(PlayerManager.IsOnline(msg.id)){
33. //发送踢下线协议
34. Player other = PlayerManager.GetPlayer(msg.id);
35. MsgKick msgKick = new MsgKick();
36. msgKick.reason = 0;
37. other.Send(msgKick);
38. //断开连接
39. NetManager.Close(other.state);
40. }
41. //获取玩家数据
42. PlayerData playerData = DbManager.GetPlayerData(msg.id);
43. if(playerData == null){
44. msg.result = 1;
45. NetManager.Send(c, msg);
46. return;
47. }
48. //构建Player
49. Player player = new Player(c);
50. player.id = msg.id;
51. player.data = playerData;
52. PlayerManager.AddPlayer(msg.id, player);
53. c.player = player;
54. //返回协议
55. msg.result = 0;
56. player.Send(msg);
57. }
58. }

* 3D\_Server\script\logic\NotepadMsgHandle.cs

1. public partial class MsgHandler {
3. //获取记事本内容
4. public static void MsgGetText(ClientState c, MsgBase msgBase){
5. MsgGetText msg = (MsgGetText)msgBase;
6. Player player = c.player;
7. if(player == null) return;
8. //获取text
9. msg.text = player.data.text;
10. player.Send(msg);
11. }
12. //保存记事本内容
13. public static void MsgSaveText(ClientState c, MsgBase msgBase){
14. MsgSaveText msg = (MsgSaveText)msgBase;
15. Player player = c.player;
16. if(player == null) return;
17. //获取text
18. player.data.text = msg.text;
19. player.Send(msg);
20. }
21. }

* 3D\_Server\script\logic\Player.cs

1. using System;
2. public class Player {
3. //id
4. public string id = "";
5. //指向ClientState
6. public ClientState state;
7. //构造函数
8. public Player(ClientState state){
9. this.state = state;
10. }
11. //坐标和旋转
12. public float x;
13. public float y;
14. public float z;
15. public float ex;
16. public float ey;
17. public float ez;
18. //在哪个房间
19. public int roomId = -1;
20. //阵营
21. public int camp = 1;
22. //坦克生命值
23. public int hp = 100;
24. //数据库数据
25. public PlayerData data;
26. //发送信息
27. public void Send(MsgBase msgBase){
28. NetManager.Send(state, msgBase);
29. }
30. }

* 3D\_Server\script\logic\PlayerData.cs

1. public class PlayerData{
2. //金币
3. public int coin = 0;
4. //记事本
5. public string text = "new text";
6. //胜利数
7. public int win = 0;
8. //失败数
9. public int lost = 0;
10. }

* 3D\_Server\script\logic\PlayerManager.cs

1. using System.Collections.Generic;
2. public class PlayerManager
3. {
4. //玩家列表
5. static Dictionary<string, Player> players = new Dictionary<string, Player>();
6. //玩家是否在线
7. public static bool IsOnline(string id){
8. return players.ContainsKey(id);
9. }
10. //获取玩家
11. public static Player GetPlayer(string id){
12. if(players.ContainsKey(id)){
13. return players[id];
14. }
15. return null;
16. }
17. //添加玩家
18. public static void AddPlayer(string id, Player player){
19. players.Add(id, player);
20. }
21. //删除玩家
22. public static void RemovePlayer(string id){
23. players.Remove(id);
24. }
25. }

* 3D\_Server\script\logic\Room.cs

1. using System;
2. using System.Collections.Generic;
3. public class Room {
4. //id
5. public int id = 0;
6. //最大玩家数
7. public int maxPlayer = 6;
8. //玩家列表
9. public Dictionary<string, bool> playerIds = new Dictionary<string, bool>();
10. //房主id
11. public string ownerId = "";
12. //状态
13. public enum Status {
14. PREPARE = 0,
15. FIGHT = 1 ,
16. }
17. public Status status = Status.PREPARE;
18. //出生点位置配置
19. static float[,,] birthConfig = new float[2, 3, 6] {
20. //阵营1出生点
21. {
22. {262.3f, -8.0f, 342.7f, 0, -151.0f, 0f},//出生点1
23. {229.7f, -5.5f, 354.4f, 0, -164.2f, 0f},//出生点2
24. {197.1f, -3.6f, 347.7f, 0, -193.0f, 0f},//出生点3
25. },
26. //阵营2出生点
27. {
28. {-80.3f,  9.5f, 114.6f, 0, -294.0f,  0f},//出生点1
29. {-91.1f, 15.5f, 139.1f, 0, -294.2f, 0f},//出生点2
30. {-62.3f,  1.2f, 76.1f,  0, -315.4f, 0f},//出生点3
31. },
32. };
33. //上一次判断结果的时间
34. private long lastjudgeTime = 0;
35. //添加玩家
36. public bool AddPlayer(string id){
37. //获取玩家
38. Player player = PlayerManager.GetPlayer(id);
39. if(player == null){
40. Console.WriteLine("room.AddPlayer fail, player is null");
41. return false;
42. }
43. //房间人数
44. if(playerIds.Count >= maxPlayer){
45. Console.WriteLine("room.AddPlayer fail, reach maxPlayer");
46. return false;
47. }
48. //准备状态才能加人
49. if(status != Status.PREPARE){
50. Console.WriteLine("room.AddPlayer fail, not PREPARE");
51. return false;
52. }
53. //已经在房间里
54. if(playerIds.ContainsKey(id)){
55. Console.WriteLine("room.AddPlayer fail, already in this room");
56. return false;
57. }
58. //加入列表
59. playerIds[id] = true;
60. //设置玩家数据
61. player.camp = SwitchCamp();
62. player.roomId = this.id;
63. //设置房主
64. if(ownerId == ""){
65. ownerId = player.id;
66. }
67. //广播
68. Broadcast(ToMsg());
69. return true;
70. }
71. //分配阵营
72. public int SwitchCamp() {
73. //计数
74. int count1 = 0;
75. int count2 = 0;
76. foreach(string id in playerIds.Keys) {
77. Player player = PlayerManager.GetPlayer(id);
78. if(player.camp == 1) {count1++;}
79. if(player.camp == 2) {count2++;}
80. }
81. //选择
82. if (count1 <= count2){
83. return 1;
84. }
85. else{
86. return 2;
87. }
88. }
89. //是不是房主
90. public bool isOwner(Player player){
91. return player.id == ownerId;
92. }
93. //删除玩家
94. public bool RemovePlayer(string id) {
95. //获取玩家
96. Player player = PlayerManager.GetPlayer(id);
97. if(player == null){
98. Console.WriteLine("room.RemovePlayer fail, player is null");
99. return false;
100. }
101. //没有在房间里
102. if(!playerIds.ContainsKey(id)){
103. Console.WriteLine("room.RemovePlayer fail, not in this room");
104. return false;
105. }
106. //删除列表
107. playerIds.Remove(id);
108. //设置玩家数据
109. player.camp = 0;
110. player.roomId = -1;
111. //设置房主
112. if(ownerId == player.id){
113. ownerId = SwitchOwner();
114. }
115. //战斗状态退出
116. if(status == Status.FIGHT){
117. player.data.lost++;
118. MsgLeaveBattle msg = new MsgLeaveBattle();
119. msg.id = player.id;
120. Broadcast(msg);
121. }
122. //房间为空
123. if(playerIds.Count == 0){
124. RoomManager.RemoveRoom(this.id);
125. }
126. //广播
127. Broadcast(ToMsg());
128. return true;
129. }
130. //选择房主
131. public string SwitchOwner() {
132. //选择第一个玩家
133. foreach(string id in playerIds.Keys) {
134. return id;
135. }
136. //房间没人
137. return "";
138. }
139. //广播消息
140. public void Broadcast(MsgBase msg){
141. foreach(string id in playerIds.Keys) {
142. Player player = PlayerManager.GetPlayer(id);
143. player.Send(msg);
144. }
145. }
146. //生成MsgGetRoomInfo协议
147. public MsgBase ToMsg(){
148. MsgGetRoomInfo msg = new MsgGetRoomInfo();
149. int count = playerIds.Count;
150. msg.players = new PlayerInfo[count];
151. //players
152. int i = 0;
153. foreach(string id in playerIds.Keys){
154. Player player = PlayerManager.GetPlayer(id);
155. PlayerInfo playerInfo = new PlayerInfo();
156. //赋值
157. playerInfo.id = player.id;
158. playerInfo.camp = player.camp;
159. playerInfo.win = player.data.win;
160. playerInfo.lost = player.data.lost;
161. playerInfo.isOwner = 0;
162. if(isOwner(player)){
163. playerInfo.isOwner = 1;
164. }
165. msg.players[i] = playerInfo;
166. i++;
167. }
168. return msg;
169. }
170. //能否开战
171. public bool CanStartBattle() {
172. //已经是战斗状态
173. if (status != Status.PREPARE){
174. return false;
175. }
176. //统计每个队伍的玩家数
177. int count1 = 0;
178. int count2 = 0;
179. foreach(string id in playerIds.Keys) {
180. Player player = PlayerManager.GetPlayer(id);
181. if(player.camp == 1){ count1++; }
182. else { count2++; }
183. }
184. //每个队伍至少要有1名玩家
185. if (count1 < 1 || count2 < 1){
186. return false;
187. }
188. return true;
189. }
190. //初始化位置
191. private void SetBirthPos(Player player, int index){
192. int camp = player.camp;
193. player.x  = birthConfig[camp-1, index,0];
194. player.y  = birthConfig[camp-1, index,1];
195. player.z  = birthConfig[camp-1, index,2];
196. player.ex = birthConfig[camp-1, index,3];
197. player.ey = birthConfig[camp-1, index,4];
198. player.ez = birthConfig[camp-1, index,5];
199. }
200. //玩家数据转成TankInfo
201. public TankInfo PlayerToTankInfo(Player player){
202. TankInfo tankInfo = new TankInfo();
203. tankInfo.camp = player.camp;
204. tankInfo.id = player.id;
205. tankInfo.hp = player.hp;
206. tankInfo.x  = player.x;
207. tankInfo.y  = player.y;
208. tankInfo.z  = player.z;
209. tankInfo.ex = player.ex;
210. tankInfo.ey = player.ey;
211. tankInfo.ez = player.ez;
212. return tankInfo;
213. }
214. //重置玩家战斗属性
215. private void ResetPlayers(){
216. //位置和旋转
217. int count1 = 0;
218. int count2 = 0;
219. foreach(string id in playerIds.Keys) {
220. Player player = PlayerManager.GetPlayer(id);
221. if(player.camp == 1){
222. SetBirthPos(player, count1);
223. count1++;
224. }
225. else {
226. SetBirthPos(player, count2);
227. count2++;
228. }
229. }
230. //生命值
231. foreach(string id in playerIds.Keys) {
232. Player player = PlayerManager.GetPlayer(id);
233. player.hp = 100;
234. }
235. }
236. //开战
237. public bool StartBattle() {
238. if(!CanStartBattle()){
239. return false;
240. }
241. //状态
242. status = Status.FIGHT;
243. //玩家战斗属性
244. ResetPlayers();
245. //返回数据
246. MsgEnterBattle msg = new MsgEnterBattle();
247. msg.mapId = 1;
248. msg.tanks = new TankInfo[playerIds.Count];
249. int i=0;
250. foreach(string id in playerIds.Keys) {
251. Player player = PlayerManager.GetPlayer(id);
252. msg.tanks[i] = PlayerToTankInfo(player);
253. i++;
254. }
255. Broadcast(msg);
256. return true;
257. }
258. //是否死亡
259. public bool IsDie(Player player){
260. return player.hp <= 0;
261. }
262. //定时更新
263. public void Update(){
264. //状态判断
265. if(status != Status.FIGHT){
266. return;
267. }
268. //时间判断
269. if(NetManager.GetTimeStamp() - lastjudgeTime < 10f){
270. return;
271. }
272. lastjudgeTime = NetManager.GetTimeStamp();
273. //胜负判断
274. int winCamp = Judgment();
275. //尚未分出胜负
276. if(winCamp == 0){
277. return;
278. }
279. //某一方胜利，结束战斗
280. status = Status.PREPARE;
281. //统计信息
282. foreach(string id in playerIds.Keys) {
283. Player player = PlayerManager.GetPlayer(id);
284. if(player.camp == winCamp){player.data.win++;}
285. else{player.data.lost++;}
286. }
287. //发送Result
288. MsgBattleResult msg = new MsgBattleResult();
289. msg.winCamp = winCamp;
290. Broadcast(msg);
291. }
292. //胜负判断
293. public int Judgment(){
294. //存活人数
295. int count1 = 0;
296. int count2 = 0;
297. foreach(string id in playerIds.Keys) {
298. Player player = PlayerManager.GetPlayer(id);
299. if(!IsDie(player)){
300. if(player.camp == 1){count1++;};
301. if(player.camp == 2){count2++;};
302. }
303. }
304. //判断
305. if(count1 <= 0){
306. return 2;
307. }
308. else if(count2 <= 0){
309. return 1;
310. }
311. return 0;
312. }
313. }

* 3D\_Server\script\logic\RoomManager.cs

1. using System.Collections.Generic;
2. public class RoomManager
3. {
4. //最大id
5. private static int maxId = 1;
6. //房间列表
7. public static Dictionary<int, Room> rooms = new Dictionary<int, Room>();
8. //创建房间
9. public static Room AddRoom(){
10. maxId++;
11. Room room = new Room();
12. room.id = maxId;
13. rooms.Add(room.id, room);
14. return room;
15. }
16. //删除房间
17. public static bool RemoveRoom(int id) {
18. rooms.Remove(id);
19. return true;
20. }
21. //获取房间
22. public static Room GetRoom(int id) {
23. if(rooms.ContainsKey(id)){
24. return rooms[id];
25. }
26. return null;
27. }
28. //生成MsgGetRoomList协议
29. public static MsgBase ToMsg(){
30. MsgGetRoomList msg = new MsgGetRoomList();
31. int count = rooms.Count;
32. msg.rooms = new RoomInfo[count];
33. //rooms
34. int i = 0;
35. foreach(Room room in rooms.Values){
36. RoomInfo roomInfo = new RoomInfo();
37. //赋值
38. roomInfo.id = room.id;
39. roomInfo.count = room.playerIds.Count;
40. roomInfo.status = (int)room.status;
41. msg.rooms[i] = roomInfo;
42. i++;
43. }
44. return msg;
45. }
46. //Update
47. public static void Update(){
48. foreach(Room room in rooms.Values){
49. room.Update();
50. }
51. }
52. }

* 3D\_Server\script\logic\RoomMsgHandle.cs

1. public partial class MsgHandler {
3. //查询战绩
4. public static void MsgGetAchieve(ClientState c, MsgBase msgBase){
5. MsgGetAchieve msg = (MsgGetAchieve)msgBase;
6. Player player = c.player;
7. if(player == null) return;
8. msg.win = player.data.win;
9. msg.lost = player.data.lost;
10. player.Send(msg);
11. }
12. //请求房间列表
13. public static void MsgGetRoomList(ClientState c, MsgBase msgBase){
14. MsgGetRoomList msg = (MsgGetRoomList)msgBase;
15. Player player = c.player;
16. if(player == null) return;
17. player.Send(RoomManager.ToMsg());
18. }
19. //创建房间
20. public static void MsgCreateRoom(ClientState c, MsgBase msgBase){
21. MsgCreateRoom msg = (MsgCreateRoom)msgBase;
22. Player player = c.player;
23. if(player == null) return;
24. //已经在房间里
25. if(player.roomId >=0 ){
26. msg.result = 1;
27. player.Send(msg);
28. return;
29. }
30. //创建
31. Room room = RoomManager.AddRoom();
32. room.AddPlayer(player.id);
33. msg.result = 0;
34. player.Send(msg);
35. }
36. //进入房间
37. public static void MsgEnterRoom(ClientState c, MsgBase msgBase){
38. MsgEnterRoom msg = (MsgEnterRoom)msgBase;
39. Player player = c.player;
40. if(player == null) return;
41. //已经在房间里
42. if(player.roomId >=0 ){
43. msg.result = 1;
44. player.Send(msg);
45. return;
46. }
47. //获取房间
48. Room room = RoomManager.GetRoom(msg.id);
49. if(room == null){
50. msg.result = 1;
51. player.Send(msg);
52. return;
53. }
54. //进入
55. if(!room.AddPlayer(player.id)){
56. msg.result = 1;
57. player.Send(msg);
58. return;
59. }
60. //返回协议
61. msg.result = 0;
62. player.Send(msg);
63. }
64. //获取房间信息
65. public static void MsgGetRoomInfo(ClientState c, MsgBase msgBase){
66. MsgGetRoomInfo msg = (MsgGetRoomInfo)msgBase;
67. Player player = c.player;
68. if(player == null) return;
69. Room room = RoomManager.GetRoom(player.roomId);
70. if(room == null){
71. player.Send(msg);
72. return;
73. }
74. player.Send(room.ToMsg());
75. }
76. //离开房间
77. public static void MsgLeaveRoom(ClientState c, MsgBase msgBase){
78. MsgLeaveRoom msg = (MsgLeaveRoom)msgBase;
79. Player player = c.player;
80. if(player == null) return;
81. Room room = RoomManager.GetRoom(player.roomId);
82. if(room == null){
83. msg.result = 1;
84. player.Send(msg);
85. return;
86. }
87. room.RemovePlayer(player.id);
88. //返回协议
89. msg.result = 0;
90. player.Send(msg);
91. }
92. //请求开始战斗
93. public static void MsgStartBattle(ClientState c, MsgBase msgBase){
94. MsgStartBattle msg = (MsgStartBattle)msgBase;
95. Player player = c.player;
96. if(player == null) return;
97. //room
98. Room room = RoomManager.GetRoom(player.roomId);
99. if(room == null){
100. msg.result = 1;
101. player.Send(msg);
102. return;
103. }
104. //是否是房主
105. if(!room.isOwner(player)){
106. msg.result = 1;
107. player.Send(msg);
108. return;
109. }
110. //开战
111. if(!room.StartBattle()){
112. msg.result = 1;
113. player.Send(msg);
114. return;
115. }
116. //成功
117. msg.result = 0;
118. player.Send(msg);
119. }
120. }

* 3D\_Server\script\logic\SyncMsgHandle.cs

1. using System;
2. public partial class MsgHandler {
3. //同步位置协议
4. public static void MsgSyncTank(ClientState c, MsgBase msgBase){
5. MsgSyncTank msg = (MsgSyncTank)msgBase;
6. Player player = c.player;
7. if(player == null) return;
8. //room
9. Room room = RoomManager.GetRoom(player.roomId);
10. if(room == null){
11. return;
12. }
13. //status
14. if(room.status != Room.Status.FIGHT){
15. return;
16. }
17. //是否作弊
18. if(Math.Abs(player.x - msg.x) > 5 ||
19. Math.Abs(player.y - msg.y) > 5 ||
20. Math.Abs(player.z - msg.z) > 5){
21. Console.WriteLine("疑似作弊 " + player.id);
22. }
23. //更新信息
24. player.x = msg.x;
25. player.y = msg.y;
26. player.z = msg.z;
27. player.ex = msg.ex;
28. player.ey = msg.ey;
29. player.ez = msg.ez;
30. //广播
31. msg.id = player.id;
32. room.Broadcast(msg);
33. }
34. //开火协议
35. public static void MsgFire(ClientState c, MsgBase msgBase){
36. MsgFire msg = (MsgFire)msgBase;
37. Player player = c.player;
38. if(player == null) return;
39. //room
40. Room room = RoomManager.GetRoom(player.roomId);
41. if(room == null){
42. return;
43. }
44. //status
45. if(room.status != Room.Status.FIGHT){
46. return;
47. }
48. //广播
49. msg.id = player.id;
50. room.Broadcast(msg);
51. }
52. //击中协议
53. public static void MsgHit(ClientState c, MsgBase msgBase){
54. MsgHit msg = (MsgHit)msgBase;
55. Player player = c.player;
56. if(player == null) return;
57. //targetPlayer
58. Player targetPlayer = PlayerManager.GetPlayer(msg.targetId);
59. if(targetPlayer == null){
60. return;
61. }
62. //room
63. Room room = RoomManager.GetRoom(player.roomId);
64. if(room == null){
65. return;
66. }
67. //status
68. if(room.status != Room.Status.FIGHT){
69. return;
70. }
71. //发送者校验
72. if(player.id != msg.id){
73. return;
74. }
75. //状态
76. int damage = 35;
77. targetPlayer.hp -= damage;
78. //广播
79. msg.id = player.id;
80. msg.hp = player.hp;
81. msg.damage = damage;
82. room.Broadcast(msg);
83. }
84. }

* 3D\_Server\script\logic\SysMsgHandler.cs

1. using System;
2. public partial class MsgHandler {
3. public static void MsgPing(ClientState c, MsgBase msgBase){
4. Console.WriteLine("MsgPing");
5. c.lastPingTime = NetManager.GetTimeStamp();
6. MsgPong msgPong = new MsgPong();
7. NetManager.Send(c, msgPong);
8. }
9. }

* 3D\_Server\script\net\ByteArray.cs

1. using System;
2. public class ByteArray  {
3. //默认大小
4. const int DEFAULT\_SIZE = 1024;
5. //初始大小
6. int initSize = 0;
7. //缓冲区
8. public byte[] bytes;
9. //读写位置
10. public int readIdx = 0;
11. public int writeIdx = 0;
12. //容量
13. private int capacity = 0;
14. //剩余空间
15. public int remain { get { return capacity-writeIdx; }}
16. //数据长度
17. public int length { get { return writeIdx-readIdx; }}
18. //构造函数
19. public ByteArray(int size = DEFAULT\_SIZE){
20. bytes = new byte[size];
21. capacity = size;
22. initSize = size;
23. readIdx = 0;
24. writeIdx = 0;
25. }
26. //构造函数
27. public ByteArray(byte[] defaultBytes){
28. bytes = defaultBytes;
29. capacity = defaultBytes.Length;
30. initSize = defaultBytes.Length;
31. readIdx = 0;
32. writeIdx = defaultBytes.Length;
33. }
34. //重设尺寸
35. public void ReSize(int size){
36. if(size < length) return;
37. if(size < initSize) return;
38. int n = 1;
39. while(n<size) n\*=2;
40. capacity = n;
41. byte[] newBytes = new byte[capacity];
42. Array.Copy(bytes, readIdx, newBytes, 0, writeIdx-readIdx);
43. bytes = newBytes;
44. writeIdx = length;
45. readIdx = 0;
46. }
47. //写入数据
48. public int Write(byte[] bs, int offset, int count){
49. if(remain < count){
50. ReSize(length + count);
51. }
52. Array.Copy(bs, offset, bytes, writeIdx, count);
53. writeIdx+=count;
54. return count;
55. }
56. //读取数据
57. public int Read(byte[] bs, int offset, int count){
58. count = Math.Min(count, length);
59. Array.Copy(bytes, 0, bs, offset, count);
60. readIdx+=count;
61. CheckAndMoveBytes();
62. return count;
63. }
64. //检查并移动数据
65. public void CheckAndMoveBytes(){
66. if(length < 8){
67. MoveBytes();
68. }
69. }
70. //移动数据
71. public void MoveBytes(){
72. if(length > 0) {
73. Array.Copy(bytes, readIdx, bytes, 0, length);
74. }
75. writeIdx = length;
76. readIdx = 0;
77. }
78. //打印缓冲区
79. public override string ToString(){
80. return BitConverter.ToString(bytes, readIdx, length);
81. }
82. //打印调试信息
83. public string Debug(){
84. return string.Format("readIdx({0}) writeIdx({1}) bytes({2})",
85. readIdx,
86. writeIdx,
87. BitConverter.ToString(bytes, 0, capacity)
88. );
89. }
90. }

* 3D\_Server\script\net\ClientState.cs

1. using System.Net.Sockets;
2. public class ClientState
3. {
4. public Socket socket;
5. public ByteArray readBuff = new ByteArray();
6. //Ping
7. public long lastPingTime = 0;
8. //玩家
9. public Player player;
10. }

* 3D\_Server\script\net\MsgBase.cs

1. using System;
2. using System.Web.Script.Serialization;
3. public class MsgBase{
4. public string protoName = "null";
5. //编码器
6. static JavaScriptSerializer Js = new JavaScriptSerializer();
7. //编码
8. public static byte[] Encode(MsgBase msgBase){
9. string s = Js.Serialize(msgBase);
10. return System.Text.Encoding.UTF8.GetBytes(s);
11. }
12. //解码
13. public static MsgBase Decode(string protoName, byte[] bytes, int offset, int count){
14. string s = System.Text.Encoding.UTF8.GetString(bytes, offset, count);
15. MsgBase msgBase = (MsgBase)Js.Deserialize(s, Type.GetType(protoName));
16. return msgBase;
17. }
18. //编码协议名（2字节长度+字符串）
19. public static byte[] EncodeName(MsgBase msgBase){
20. //名字bytes和长度
21. byte[] nameBytes = System.Text.Encoding.UTF8.GetBytes(msgBase.protoName);
22. Int16 len = (Int16)nameBytes.Length;
23. //申请bytes数值
24. byte[] bytes = new byte[2+len];
25. //组装2字节的长度信息
26. bytes[0] = (byte)(len%256);
27. bytes[1] = (byte)(len/256);
28. //组装名字bytes
29. Array.Copy(nameBytes, 0, bytes, 2, len);
30. return bytes;
31. }
32. //解码协议名（2字节长度+字符串）
33. public static string DecodeName(byte[] bytes, int offset, out int count){
34. count = 0;
35. //必须大于2字节
36. if(offset + 2 > bytes.Length){
37. return "";
38. }
39. //读取长度
40. Int16 len = (Int16)((bytes[offset+1] << 8 )| bytes[offset] );
41. if(len <= 0){
42. return "";
43. }
44. //长度必须足够
45. if(offset + 2 + len > bytes.Length){
46. return "";
47. }
48. //解析
49. count = 2+len;
50. string name = System.Text.Encoding.UTF8.GetString(bytes, offset+2, len);
51. return name;
52. }
53. }

* 3D\_Server\script\net\NetManager.cs

1. using System;
2. using System.Net;
3. using System.Net.Sockets;
4. using System.Collections.Generic;
5. using System.Reflection;
6. class NetManager
7. {
8. //监听Socket
9. public static Socket listenfd;
10. //客户端Socket及状态信息
11. public static Dictionary<Socket, ClientState> clients = new Dictionary<Socket, ClientState>();
12. //Select的检查列表
13. static List<Socket> checkRead = new List<Socket>();
14. //ping间隔
15. public static long pingInterval = 30;
16. public static void StartLoop(int listenPort)
17. {
18. //Socket
19. listenfd = new Socket(AddressFamily.InterNetwork,
20. SocketType.Stream, ProtocolType.Tcp);
21. //Bind
22. IPAddress ipAdr = IPAddress.Parse("0.0.0.0");
23. IPEndPoint ipEp = new IPEndPoint(ipAdr, listenPort);
24. listenfd.Bind(ipEp);
25. //Listen
26. listenfd.Listen(0);
27. Console.WriteLine("[服务器]启动成功");
28. //循环
29. while(true){
30. ResetCheckRead();  //重置checkRead
31. Socket.Select(checkRead, null, null, 1000);
32. //检查可读对象
33. for(int i = checkRead.Count-1; i>=0; i--){
34. Socket s = checkRead[i];
35. if(s == listenfd){
36. ReadListenfd(s);
37. }
38. else{
39. ReadClientfd(s);
40. }
41. }
42. //超时
43. Timer();
44. }
45. }
46. //填充checkRead列表
47. public static void ResetCheckRead(){
48. checkRead.Clear();
49. checkRead.Add(listenfd);
50. foreach (ClientState s in clients.Values){
51. checkRead.Add(s.socket);
52. }
53. }
54. //读取Listenfd
55. public static void ReadListenfd(Socket listenfd){
56. try{
57. Socket clientfd = listenfd.Accept();
58. Console.WriteLine("Accept " + clientfd.RemoteEndPoint.ToString());
59. ClientState state = new ClientState();
60. state.socket = clientfd;
61. state.lastPingTime = GetTimeStamp();
62. clients.Add(clientfd, state);
63. }catch(SocketException ex){
64. Console.WriteLine("Accept fail" + ex.ToString());
65. }
66. }
67. //关闭连接
68. public static void Close(ClientState state){
69. //消息分发
70. MethodInfo mei =  typeof(EventHandler).GetMethod("OnDisconnect");
71. object[] ob = {state};
72. mei.Invoke(null, ob);
73. //关闭
74. state.socket.Close();
75. clients.Remove(state.socket);
76. }
77. //读取Clientfd
78. public static void ReadClientfd(Socket clientfd){
79. ClientState state = clients[clientfd];
80. ByteArray readBuff = state.readBuff;
81. //接收
82. int count = 0;
83. //缓冲区不够，清除，若依旧不够，只能返回
84. //当单条协议超过缓冲区长度时会发生
85. if(readBuff.remain <=0){
86. OnReceiveData(state);
87. readBuff.MoveBytes();
88. };
89. if(readBuff.remain <=0){
90. Console.WriteLine("Receive fail , maybe msg length > buff capacity");
91. Close(state);
92. return;
93. }
94. try{
95. count = clientfd.Receive(readBuff.bytes, readBuff.writeIdx, readBuff.remain, 0);
96. }catch(SocketException ex){
97. Console.WriteLine("Receive SocketException " + ex.ToString());
98. Close(state);
99. return;
100. }
101. //客户端关闭
102. if(count <= 0 ){
103. Console.WriteLine("Socket Close " + clientfd.RemoteEndPoint.ToString());
104. Close(state);
105. return;
106. }
107. //消息处理
108. readBuff.writeIdx+=count;
109. //处理二进制消息
110. OnReceiveData(state);
111. //移动缓冲区
112. readBuff.CheckAndMoveBytes();
113. }
114. //数据处理
115. public static void OnReceiveData(ClientState state){
116. ByteArray readBuff = state.readBuff;
117. //消息长度
118. if(readBuff.length <= 2) {
119. return;
120. }
121. //消息体长度
122. int readIdx = readBuff.readIdx;
123. byte[] bytes =readBuff.bytes;
124. Int16 bodyLength = (Int16)((bytes[readIdx+1] << 8 )| bytes[readIdx]);
125. if(readBuff.length < bodyLength){
126. return;
127. }
128. readBuff.readIdx +=2;
129. //解析协议名
130. int nameCount = 0;
131. string protoName = MsgBase.DecodeName(readBuff.bytes, readBuff.readIdx, out nameCount);
132. if(protoName == ""){
133. Console.WriteLine("OnReceiveData MsgBase.DecodeName fail");
134. Close(state);
135. return;
136. }
137. readBuff.readIdx += nameCount;
138. //解析协议体
139. int bodyCount = bodyLength - nameCount;
140. if(bodyCount <= 0){
141. Console.WriteLine("OnReceiveData fail, bodyCount <=0 ");
142. Close(state);
143. return;
144. }
145. MsgBase msgBase = MsgBase.Decode(protoName, readBuff.bytes, readBuff.readIdx, bodyCount);
146. readBuff.readIdx += bodyCount;
147. readBuff.CheckAndMoveBytes();
148. //分发消息
149. MethodInfo mi =  typeof(MsgHandler).GetMethod(protoName);
150. object[] o = {state, msgBase};
151. Console.WriteLine("Receive " + protoName);
152. if(mi != null){
153. mi.Invoke(null, o);
154. }
155. else{
156. Console.WriteLine("OnReceiveData Invoke fail " + protoName);
157. }
158. //继续读取消息
159. if(readBuff.length > 2){
160. OnReceiveData(state);
161. }
162. }
163. //发送
164. public static void Send(ClientState cs, MsgBase msg){
165. //状态判断
166. if(cs == null){
167. return;
168. }
169. if(!cs.socket.Connected){
170. return;
171. }
172. //数据编码
173. byte[] nameBytes = MsgBase.EncodeName(msg);
174. byte[] bodyBytes = MsgBase.Encode(msg);
175. int len = nameBytes.Length + bodyBytes.Length;
176. byte[] sendBytes = new byte[2+len];
177. //组装长度
178. sendBytes[0] = (byte)(len%256);
179. sendBytes[1] = (byte)(len/256);
180. //组装名字
181. Array.Copy(nameBytes, 0, sendBytes, 2, nameBytes.Length);
182. //组装消息体
183. Array.Copy(bodyBytes, 0, sendBytes, 2+nameBytes.Length, bodyBytes.Length);
184. //为简化代码，不设置回调
185. try{
186. cs.socket.BeginSend(sendBytes,0, sendBytes.Length, 0, null, null);
187. }catch(SocketException ex){
188. Console.WriteLine("Socket Close on BeginSend" + ex.ToString());
189. }
190. }
191. //定时器
192. static void Timer(){
193. //消息分发
194. MethodInfo mei =  typeof(EventHandler).GetMethod("OnTimer");
195. object[] ob = {};
196. mei.Invoke(null, ob);
197. }
198. //获取时间戳
199. public static long GetTimeStamp() {
200. TimeSpan ts = DateTime.UtcNow - new DateTime(1970, 1, 1, 0, 0, 0, 0);
201. return Convert.ToInt64(ts.TotalSeconds);
202. }
203. }

* 3D\_Server\script\proto\BattleMsg.cs

1. //坦克信息
2. [System.Serializable]
3. public class TankInfo{
4. public string id = "";  //玩家id
5. public int camp = 0;    //阵营
6. public int hp = 0;      //生命值
7. public float x = 0;     //位置
8. public float y = 0;
9. public float z = 0;
10. public float ex = 0;    //旋转
11. public float ey = 0;
12. public float ez = 0;
13. }
14. //进入战场（服务端推送）
15. public class MsgEnterBattle:MsgBase {
16. public MsgEnterBattle() {protoName = "MsgEnterBattle";}
17. //服务端回
18. public TankInfo[] tanks;
19. public int mapId = 1;   //地图，只有一张
20. }
21. //战斗结果（服务端推送）
22. public class MsgBattleResult:MsgBase {
23. public MsgBattleResult() {protoName = "MsgBattleResult";}
24. //服务端回
25. public int winCamp = 0;  //获胜的阵营
26. }
27. //玩家退出（服务端推送）
28. public class MsgLeaveBattle:MsgBase {
29. public MsgLeaveBattle() {protoName = "MsgLeaveBattle";}
30. //服务端回
31. public string id = "";  //玩家id
32. }

* 3D\_Server\script\proto\LoginMsg.cs

1. //注册
2. public class MsgRegister:MsgBase {
3. public MsgRegister() {protoName = "MsgRegister";}
4. //客户端发
5. public string id = "";
6. public string pw = "";
7. //服务端回（0-成功，1-失败）
8. public int result = 0;
9. }
10. //登陆
11. public class MsgLogin:MsgBase {
12. public MsgLogin() {protoName = "MsgLogin";}
13. //客户端发
14. public string id = "";
15. public string pw = "";
16. //服务端回（0-成功，1-失败）
17. public int result = 0;
18. }
19. //踢下线（服务端推送）
20. public class MsgKick:MsgBase {
21. public MsgKick() {protoName = "MsgKick";}
22. //原因（0-其他人登陆同一账号）
23. public int reason = 0;
24. }

* 3D\_Server\script\proto\NotepadMsg.cs

1. //获取记事本内容
2. public class MsgGetText:MsgBase {
3. public MsgGetText() {protoName = "MsgGetText";}
4. //服务端回
5. public string text = "";
6. }
7. //保存记事本内容
8. public class MsgSaveText:MsgBase {
9. public MsgSaveText() {protoName = "MsgSaveText";}
10. //客户端发
11. public string text = "";
12. //服务端回（0-成功 1-文字太长）
13. public int result = 0;
14. }

* 3D\_Server\script\proto\RoomMsg.cs

1. //查询成绩
2. public class MsgGetAchieve:MsgBase {
3. public MsgGetAchieve() {protoName = "MsgGetAchieve";}
4. //服务端回
5. public int win = 0;
6. public int lost = 0;
7. }
8. //房间信息
9. public class RoomInfo{
10. public int id = 0;      //房间id
11. public int count = 0;   //人数
12. public int status = 0;  //状态0-准备中 1-战斗中
13. }
14. //请求房间列表
15. public class MsgGetRoomList:MsgBase {
16. public MsgGetRoomList() {protoName = "MsgGetRoomList";}
17. //服务端回
18. public RoomInfo[] rooms;
19. }
20. //创建房间
21. public class MsgCreateRoom:MsgBase {
22. public MsgCreateRoom() {protoName = "MsgCreateRoom";}
23. //服务端回
24. public int result = 0;
25. }
26. //进入房间
27. public class MsgEnterRoom:MsgBase {
28. public MsgEnterRoom() {protoName = "MsgEnterRoom";}
29. //客户端发
30. public int id = 0;
31. //服务端回
32. public int result = 0;
33. }
34. //玩家信息
35. public class PlayerInfo{
36. public string id = "lpy";   //账号
37. public int camp = 0;        //阵营
38. public int win = 0;         //胜利数
39. public int lost = 0;        //失败数
40. public int isOwner = 0;     //是否是房主
41. }
42. //获取房间信息
43. public class MsgGetRoomInfo:MsgBase {
44. public MsgGetRoomInfo() {protoName = "MsgGetRoomInfo";}
45. //服务端回
46. public PlayerInfo[] players;
47. }
48. //离开房间
49. public class MsgLeaveRoom:MsgBase {
50. public MsgLeaveRoom() {protoName = "MsgLeaveRoom";}
51. //服务端回
52. public int result = 0;
53. }
54. //开战
55. public class MsgStartBattle:MsgBase {
56. public MsgStartBattle() {protoName = "MsgStartBattle";}
57. //服务端回
58. public int result = 0;
59. }

* 3D\_Server\script\proto\SyncMsg.cs

1. //同步坦克信息
2. public class MsgSyncTank:MsgBase {
3. public MsgSyncTank() {protoName = "MsgSyncTank";}
4. //位置、旋转、炮塔旋转
5. public float x = 0f;
6. public float y = 0f;
7. public float z = 0f;
8. public float ex = 0f;
9. public float ey = 0f;
10. public float ez = 0f;
11. public float turretY = 0f;
12. public float gunX = 0f;
13. //服务端补充
14. public string id = "";      //哪个坦克
15. }
16. //开火
17. public class MsgFire:MsgBase {
18. public MsgFire() {protoName = "MsgFire";}
19. //炮弹初始位置、旋转
20. public float x = 0f;
21. public float y = 0f;
22. public float z = 0f;
23. public float ex = 0f;
24. public float ey = 0f;
25. public float ez = 0f;
26. //服务端补充
27. public string id = "";      //哪个坦克
28. }
29. //击中
30. public class MsgHit:MsgBase {
31. public MsgHit() {protoName = "MsgHit";}
32. //击中谁
33. public string targetId = "";
34. //击中点
35. public float x = 0f;
36. public float y = 0f;
37. public float z = 0f;
38. //服务端补充
39. public string id = "";      //哪个坦克
40. public int hp = 0;          //被击中坦克血量
41. public int damage = 0;      //受到的伤害
42. }

* 3D\_Server\script\proto\SysMsg.cs

1. public class MsgPing:MsgBase {
2. public MsgPing() {protoName = "MsgPing";}
3. }
4. public class MsgPong:MsgBase {
5. public MsgPong() {protoName = "MsgPong";}
6. }

* 3D\_Server\script\SQL\game.sql

1. drop database if exists game;
2. CREATE DATABASE IF NOT EXISTS game;
3. use game;
4. SET NAMES utf8mb4;
5. SET FOREIGN\_KEY\_CHECKS = 0;
6. -- ----------------------------
7. -- Table structure for account
8. -- ----------------------------
9. DROP TABLE IF EXISTS `account`;
10. CREATE TABLE `account`  (
11. `id` text CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci NOT NULL,
12. `pw` text CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci NULL,
13. PRIMARY KEY (`id`(20)) USING BTREE
14. ) ENGINE = InnoDB CHARACTER SET = utf8mb4 COLLATE = utf8mb4\_0900\_ai\_ci ROW\_FORMAT = Dynamic;
15. -- ----------------------------
16. -- Table structure for player
17. -- ----------------------------
18. DROP TABLE IF EXISTS `player`;
19. CREATE TABLE `player`  (
20. `id` text CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci NOT NULL,
21. `data` text CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci NULL,
22. PRIMARY KEY (`id`(20)) USING BTREE
23. ) ENGINE = InnoDB CHARACTER SET = utf8mb4 COLLATE = utf8mb4\_0900\_ai\_ci ROW\_FORMAT = Dynamic;
24. SET FOREIGN\_KEY\_CHECKS = 1;