

**GameFacade类介绍**

**简介：作为一个中转站，为了减少个脚本之间的耦合性，保证单一脚本执行单一功能，并且互相之间尽量减少引用，同时这个脚本继承与MonoBehaviour，也就是unity会帮我实例化他，在façade里会实例化一些不继承于monobehaviour的脚本**

public class GameFacade : MonoBehaviour//GameFacade挂在一个空游戏对象上，继承自MonoBehaviour，可以实例化

{

public int player = -1;

private static GameFacade \_instance;//单例模式，作为游戏主控制脚本，需要在别的很多地方都能访问到

public static GameFacade Instance

{ get

{

if (\_instance == null)

{

\_instance = GameObject.Find("GameFacade").GetComponent<GameFacade>();

}

return \_instance;

}

}

//持有这些各个模块之间的管理器，管理器们没有挂在游戏物体上，要在这里从初始

public RequestManager requestMng;

public ClientManager clientMng;

private bool isEnterPlaying = false;

void Start ()//初始化所有脚本，即先new出来，在调用他们自己内部的初始化方法

{

//这里在每个脚本的初始化方法里把自己传递过去，让每个管理器持有自己的引用

requestMng = new RequestManager(this);

clientMng = new ClientManager(this);

requestMng.OnInit();

clientMng.OnInit();

Screen.SetResolution(1366, 768, false);

}

void Update ()

{

UpdateManager();

if (isEnterPlaying)

{

}

}

private void OnDestroy()

{

requestMng.OnDestroy();

clientMng.OnDestroy();

}

private void UpdateManager()

{

requestMng.Update();

clientMng.Update();

}

public void AddRequest(ActionCode actionCode, BaseRequest request)

{

requestMng.AddRequest(actionCode, request);

}

public void RemoveRequest(ActionCode actionCode)

{

requestMng.RemoveRequest(actionCode);

}

public void SendRequest(RequestCode requestCode, ActionCode actionCode, string data)//发送数据的方法，把需要发送的请求数据传到这里，在传递给ClientManager然后按格式打包发送

{

clientMng.SendRequest(requestCode, actionCode, data);

}

public void HandleReponse(ActionCode actionCode, string data)

{

requestMng.HandleReponse(actionCode, data);

}

/\*

public void ShowMessage(string msg)

{

uiMng.ShowMessage(msg);

}

public void PlayBgSound(string soundName)

{

audioMng.PlayBgSound(soundName);

}

public void PlayNormalSound(string soundName)

{

audioMng.PlayNormalSound(soundName);

}

public void SetUserData(UserData ud)

{

playerMng.UserData = ud;

}

public UserData GetUserData()

{

return playerMng.UserData;

}

public void SetCurrentRoleType(RoleType rt)

{

playerMng.SetCurrentRoleType(rt);

}

public GameObject GetCurrentRoleGameObject()

{

return playerMng.GetCurrentRoleGameObject();

}

public void EnterPlayingSync()

{

isEnterPlaying = true;

}

private void EnterPlaying()

{

playerMng.SpawnRoles();

cameraMng.FollowRole();

}

public void StartPlaying()

{

playerMng.AddControlScript();

playerMng.CreateSyncRequest();

}

public void SendAttack(int damage)

{

playerMng.SendAttack(damage);

}

public void GameOver()

{

cameraMng.WalkthroughScene();

playerMng.GameOver();

}

public void UpdateResult(int totalCount, int winCount)

{

playerMng.UpdateResult(totalCount, winCount);

}

\*/

}

**BaseManager介绍**

**简介：所有管理类的父类**

public class BaseManager//所有管理器类的基类，提取出来的一些通用的成员和方法

{

protected GameFacade facade;

public BaseManager(GameFacade facade)//构造方法，在自己被GameFacade创建出来的时候获取他的引用

{

this.facade = facade;

}

public virtual void OnInit() { }

public virtual void Update(){}

public virtual void OnDestroy() { }

}

**ClientManager介绍**

**简介：开启socket与服务器来链接，开启线程循环接收消息，并且发送消息**

// <summary>

/// 这个是用来管理跟服务器端的Socket连接

/// </summary>

public class ClientManager :BaseManager {

private const string IP = "127.0.0.1";

private const int PORT = 6688;

private Socket clientSocket;

private Message msg = new Message();//帮助解析信息的类

public ClientManager(GameFacade facade) : base(facade) { }

public override void OnInit()//当ClientManager被GameFacade创建出来的时候调用这个方法

{

base.OnInit();

clientSocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

try

{

clientSocket.Connect(IP, PORT);

clientSocket.BeginReceive(msg.Data, msg.StartIndex, msg.RemainSize, SocketFlags.None, ReceiveCallback, null);//开始循环接收消息

}

catch (Exception e)

{

Debug.LogWarning("无法连接到服务器端，请检查您的网络！！" + e);

}

}

private void ReceiveCallback(IAsyncResult ar)

{

try

{

if (clientSocket == null || clientSocket.Connected == false) return;

int count = clientSocket.EndReceive(ar);

msg.ReadMessage(count, OnProcessDataCallback);//需要两个参数，一个是这次读取新信息要增加的startIndex值，第二个是解析完消息调用的回调函数

clientSocket.BeginReceive(msg.Data, msg.StartIndex, msg.RemainSize, SocketFlags.None, ReceiveCallback, null); //循环接收消息

}

catch(Exception e)

{

Debug.Log(e);

}

}

private void OnProcessDataCallback(ActionCode actionCode,string data)//解析完从服务器发回来的消息后，把actionCode和data通过GameFacade传递给RequestManager处理

{

facade.HandleReponse(actionCode, data);

}

public void SendRequest(RequestCode requestCode, ActionCode actionCode, string data)//发送数据的方法，把需要发送的请求数据传到这里，然后按格式打包发送，需要通过GameFacade中介来调用这个方法

{

byte[] bytes = Message.PackData(requestCode, actionCode, data);

clientSocket.Send(bytes);

}

public override void OnDestroy()

{

base.OnDestroy();

try

{

clientSocket.Close();

}

catch (Exception e)

{

Debug.LogWarning("无法关闭跟服务器端的连接！！" + e);

}

}

**RequestManager介绍**

**简介：处理分发消息，根据服务器返回的ActionCode来找到对应的发送RequestCode的类，然后处理返回的消息**

public class RequestManager : BaseManager

{

public RequestManager(GameFacade facade) : base(facade) { }

private Dictionary<ActionCode, BaseRequest> requestDict = new Dictionary<ActionCode, BaseRequest>();

public void AddRequest(ActionCode actionCode,BaseRequest request)//向字典中添加request对象

{

requestDict.Add(actionCode, request);

}

public void RemoveRequest(ActionCode actionCode)//移除字典里的request对象

{

requestDict.Remove(actionCode);

}

public void HandleReponse(ActionCode actionCode, string data)//这里处理从ClientManager接收完的返回消息中间经过了中介facade

{

BaseRequest request;

requestDict.TryGetValue(actionCode,out request);//通过ActionCode来把消息传送给对应的Response对象里面的OnResponse方法来处理

if (request == null)

{

Debug.LogWarning("无法得到ActionCode[" + actionCode + "]对应的Request类");return;

}

request.OnResponse(data);

}

}

**LoginRequest介绍**

**简介：负责向服务器发送登录请求，把输入的用户名和密码发送给服务器进行校验，并且接受返回的结果ReturnCode来做出相应的反应**

public class LoginRequest : BaseRequest

{

bool islogin = false;

bool ReturnCodeisfail = false;

public Text nametext;

public Text passwordtext;

public TiShi tishi;

public GameObject LoginPlane;

public GameObject RoomListPlane;

public GameObject RoomItem;

public override void Awake()

{

requestCode = RequestCode.User;//这个RequestCode对应着服务器那边的UserController

actionCode = ActionCode.Login;//这个ActionCode对应着UserController里面的Login方法

base.Awake();//base里面的语句：facade.AddRequest(actionCode, this) 将自己添加进字典里，所以上面三条语句要在这之前执行

RoomListPlane.SetActive (false);

RoomItem.SetActive(false);

}

private void Update()

{

if (islogin)

{

islogin = false;

RoomListPlane.SetActive(true);//显示房间列表

RoomListPlane.GetComponent<RoomListRequest>().FlashRoomList();//刷新房间列表

LoginPlane.SetActive(false);//隐藏自己

}

if (ReturnCodeisfail)

{

ReturnCodeisfail = false;

tishi.ShowXinXi("登录失败，请检查密码用户名和网络");

}

}

public void ClickToSendRequest()

{

string username = nametext.text;

string password = passwordtext.text;

string msg = "";

if (string.IsNullOrEmpty(username))

{

msg += "用户名不能为空 ";

}

if (string.IsNullOrEmpty(password))

{

msg += "密码不能为空 ";

}

if (msg != "")

{

tishi.ShowXinXi(msg); return;

}

string data = username + "," + password;//把用户名密码按一定规则组合，发送给服务器

base.SendRequest(data);

passwordtext.text = "";

nametext.text = "";

}

public override void OnResponse(string data)//这里来处理返回的信息，消息来源是：ReadMessage方法的回调函数OnProcessDataCallback——>GameFacade里的HandleReponse——>RequestManager里的HandleReponse将消息分发给Request对象

{

ReturnCode returnCode = (ReturnCode)int.Parse(data);

if (returnCode == ReturnCode.Success)

{

islogin = true;

}

else if(returnCode == ReturnCode.Fail)

{

ReturnCodeisfail = true;

}

}

**RoomListRequest介绍**

**简介：RoomListRequest用于处理第一次进入到房间列表页面的刷新请求，和接下来的手动刷新请求的发送以及响应**

public class RoomListRequest : BaseRequest

{

bool ReturnCodeisfail = false;

bool ReturnCodeissuccess = false;

string roomstate;

public GameObject roomplane;

public TiShi tishi;

public Image battle;

public Image waitingbattle;

public Image waitingjoin;

public override void Awake()//重写父类中的方法

{

requestCode = RequestCode.Room;//这个RequestCode对应着服务器那边的UserController

actionCode = ActionCode.ListRoom;//这个ActionCode对应着UserController里面的Login方法

base.Awake();//base里面的语句：facade.AddRequest(actionCode, this) 将自己添加进字典里，所以上面三条语句要在这之前执行

}

private void Update()

{

if (ReturnCodeisfail)

{

ReturnCodeisfail = false;

roomplane.SetActive(false);

tishi.ShowXinXi("没有房间！");

}

if (ReturnCodeissuccess)

{

ReturnCodeissuccess = false;

roomplane.SetActive(true);

if (roomstate == "Battle")

{

battle.enabled = true;

waitingbattle.enabled = false;

waitingjoin.enabled = false;

}

else if (roomstate == "WaitingBattle")

{

battle.enabled = false;

waitingbattle.enabled = true;

waitingjoin.enabled = false;

}

else if (roomstate == "WaitingJoin")

{

battle.enabled = false;

waitingbattle.enabled = false;

waitingjoin.enabled = true;

}

}

}

//这里发送的数据没有实际含义，只是把自己的RequestCode和ActionCode发送给服务器而已

//这个方法会在UI界面里面注册，当点击按钮的时候进行调用

public void FlashRoomList()

{

base.SendRequest("Flash");

}

public override void OnResponse(string data)

{

string[] strs = data.Split(',');

ReturnCode returnCode = (ReturnCode)int.Parse(strs[0]);

if (returnCode == ReturnCode.Success)

{

ReturnCodeissuccess = true;

if (strs[1] == "Battle")

{

roomstate = "Battle";

}

else if (strs[1] == "WaitingBattle")

{

roomstate = "WaitingBattle";

}

else if (strs[1] == "WaitingJoin")

{

roomstate = "WaitingJoin";

}

}

else if (returnCode == ReturnCode.Fail)

{

ReturnCodeisfail = true;

}

}

}

**CreateRoomRequest介绍**

**简介：CreateRoomRequest主要用于处理发送创建房间的请求以及处理是否创建成功的响应**

public class CreateRoomRequest : BaseRequest

{

bool ReturnCodeisfail = false;

bool ReturnCodeissuccess = false;

public TiShi tishi;

public GameObject RoomItem;

public GameObject RoomList;

public Image player1;

public Text kaishibuttontext;

public GameObject kaishibutton;

public override void Awake()

{

requestCode = RequestCode.Room;

actionCode = ActionCode.CreateRoom;

base.Awake();

}

private void Update()

{

if (ReturnCodeisfail)

{

ReturnCodeisfail = false;

tishi.ShowXinXi("现阶段只能存在一个房间！");

}

if (ReturnCodeissuccess)

{

ReturnCodeissuccess = false;

RoomItem.SetActive(true);

RoomList.SetActive(false);

player1.enabled = true;

kaishibutton.SetActive(true);

kaishibuttontext.text = "开始游戏";//自己是房主，所以开始游戏按钮

}

}

//这里发送的数据没有实际含义，只是把自己的RequestCode和ActionCode发送给服务器而已

//这个方法会在UI界面里面注册，当点击按钮的时候进行调用

public void CreatRoom()

{

base.SendRequest("r");

}

public override void OnResponse(string data)

{

string[] strs = data.Split(',');

ReturnCode returnCode = (ReturnCode)int.Parse(strs[0]);

if (returnCode == ReturnCode.Success)

{

GameFacade.Instance.player = int.Parse(strs[1]);

ReturnCodeissuccess = true;

}

else if (returnCode == ReturnCode.Fail)

{

ReturnCodeisfail = true;

}

}

}

**JoinRequest介绍**

**简介：JoinRequest处理发送加入房间的请求，并处理服务端给予的响应**

public class JoinRequest : BaseRequest

{

bool ReturnCodeisfail = false;

bool ReturnCodeissuccess = false;

public GameObject RoomList;

public GameObject RoomItem;

public Image player1;

public Image player2;

public TiShi tishi;

public GameObject kaishibutton;

private string datastrq\_playertype;

public override void Awake()

{

requestCode = RequestCode.Room;

actionCode = ActionCode.JoinRoom;

base.Awake();

}

private void Update()

{

if (ReturnCodeisfail)

{

ReturnCodeisfail = false;

tishi.ShowXinXi("无法加入房间！");

}

if (ReturnCodeissuccess)

{

ReturnCodeissuccess = false;

GameFacade.Instance.player = int.Parse(datastrq\_playertype);

RoomItem.SetActive(true);

RoomList.SetActive(false);

player1.enabled = true;

player2.enabled = true;

kaishibutton.SetActive(false);

}

}

public void JoinRoom()

{

base.SendRequest("r");

}

public override void OnResponse(string data)

{

string[] strs = data.Split(',');

ReturnCode returnCode = (ReturnCode)int.Parse(strs[0]);

datastrq\_playertype = strs[1];

if (returnCode == ReturnCode.Success)

{

ReturnCodeissuccess = true;

}

else if (returnCode == ReturnCode.Fail)

{

ReturnCodeisfail = true;

}

}

}

**QuitRoomRequest介绍**

**简介：QuitRoomRequest处理发送退出房间的请求，并处理服务端给予的响应，在服务端会判断本客户端是否是房主来做出答复**

public class QuitRoomRequest : BaseRequest

{

bool istuichu = false;

public GameObject RoomItem;

public GameObject RoomList;

public Image player1;

public Image player2;

public override void Awake()

{

requestCode = RequestCode.Room;

actionCode = ActionCode.QuitRoom;

base.Awake();

}

private void Update()

{

if (istuichu)

{

istuichu = false;

player1.enabled = false;

player2.enabled = false;

RoomList.SetActive(true);

RoomList.GetComponent<RoomListRequest>().FlashRoomList();//刷新房间列表

RoomItem.SetActive(false);

}

}

public void TuiChu()

{

base.SendRequest("q");

}

public override void OnResponse(string data)

{

string[] strs = data.Split(',');

ReturnCode returnCode = (ReturnCode)int.Parse(strs[0]);

if(returnCode == ReturnCode.Success)

{

istuichu = true;

}

}

**UpdateRoomRequest介绍**

**简介：UpdateRoomRequest处理刷新房间内部信息的请求，比如有玩家退出房间，需要及时更新对方玩家头像，或者是房主退出房间要处理让剑销毁，并让另一名玩家也强制退出房间的操作**

public class UpdateRoomRequest : BaseRequest

{

public Image player2;

public override void Awake()

{

requestCode = RequestCode.Room;

actionCode = ActionCode.UpdateRoom;

base.Awake();

}

public override void OnResponse(string data)

{

if (data == "1")

{

player2.enabled = true;

}

else if (data == "0")

{

player2.enabled = false;

}

}

}

**StartGameRequest介绍**

**简介：StartGameRequest处理开始游戏的请求，会发送达服务器，服务器判断是否达到开始游戏的标准，若达到则开始游戏，并且广播给房间里面其他的客户端来开始游戏，并且根据服务器的返回来做出具体响应的操作，比如动态的添加控制移动的脚本给本地客户端需要控制的角色，激活对应的本地角色的摄像机等**

public class StartGameRequest : BaseRequest

{

public GameObject roomitem;

public TiShi tishi;

bool ReturnCodeisfail = false;

bool ReturnCodeissuccess = false;

public GameObject camera0;

public GameObject camera1;

public GameObject nothingcamera;

public GameObject gamecontrol;

public override void Awake()

{

requestCode = RequestCode.Game;

actionCode = ActionCode.StartGame;

base.Awake();

}

private void Update()

{

if (ReturnCodeisfail)

{

ReturnCodeisfail = false;

tishi.ShowXinXi("无法开始游戏，人数不足");

}

if (ReturnCodeissuccess)

{

ReturnCodeissuccess = false;

nothingcamera.SetActive(false);

roomitem.SetActive(false);

gamecontrol.AddComponent<TestPlayerControl>();//添加控制移动的脚本

gamecontrol.AddComponent<MoveRequest>();//添加同步移动的脚本

gamecontrol.AddComponent<NowWeaponRequest>();//添加同步武器的脚本

if (GameFacade.Instance.player == 0)//激活摄像机

{

camera0.SetActive(true);

}else if(GameFacade.Instance.player == 1)

{

camera1.SetActive(true);

}

}

}

public void ClickStartGame()

{

base.SendRequest("r");

}

public override void OnResponse(string data)

{

ReturnCode returnCode = (ReturnCode)int.Parse(data);

if (returnCode == ReturnCode.Success)

{

ReturnCodeissuccess = true;

}

else if(returnCode == ReturnCode.Fail)

{

ReturnCodeisfail = true;

}

}

}

**MoveRequest介绍**

**简介：同步远端玩家的移动位置，旋转信息，每秒24次**

public class MoveRequest : BaseRequest

{

private Transform localPlayerTransform;

private Transform remotePlayerTransform;

private int setRate = 24;

private Vector3 pos;

private Vector3 rotation;

private bool issetRemotePlayer = false;

public override void Awake()

{

requestCode = RequestCode.Game;

actionCode = ActionCode.Move;

base.Awake();

}

private void FixedUpdate()

{

if (issetRemotePlayer)

{

setRemotePlayer();

issetRemotePlayer = false;

}

}

private void OnEnable()

{

if (GameFacade.Instance.player == 0)

{

Debug.Log("角色0");

localPlayerTransform = GameObject.Find("Player0").GetComponent<Transform>();

remotePlayerTransform = GameObject.Find("Player1").GetComponent<Transform>();

}

else if (GameFacade.Instance.player == 1)

{

Debug.Log("角色1");

localPlayerTransform = GameObject.Find("Player1").GetComponent<Transform>();

remotePlayerTransform = GameObject.Find("Player0").GetComponent<Transform>();

}

else if (GameFacade.Instance.player == -1)

{

Debug.Log("MoveRequest中角色未指定");

}

InvokeRepeating("setLocalPlayerToRemote", 1f, 1f / setRate);

}

void setLocalPlayerToRemote()

{

float x = localPlayerTransform.position.x;

float y = localPlayerTransform.position.y;

float z = localPlayerTransform.position.z;

float rotationX = localPlayerTransform.eulerAngles.x;

float rotationY = localPlayerTransform.eulerAngles.y;

float rotationZ = localPlayerTransform.eulerAngles.z;

string data = string.Format("{0},{1},{2}|{3},{4},{5}", x, y, z, rotationX, rotationY, rotationZ);

base.SendRequest(data);

}

void setRemotePlayer()

{

remotePlayerTransform.position = pos;

remotePlayerTransform.eulerAngles = rotation;

}

public override void OnResponse(string data)

{

string[] strs = data.Split('|');

pos = UnityTools.ParseVector3(strs[0]);

rotation = UnityTools.ParseVector3(strs[1]);

issetRemotePlayer = true;

}

}