ForestBattle开发流程

2017年8月2日

19:46

一、游戏服务器端开发

服务器架构图：

DAO 
Controller 
ControllerManager 
Conn Helper 
Client 
Client 
Message 

在该框架下，主要分为几个大类：Servers、Controller、Tool（ConnHelper）、Model、DAO

Servers大类主要包含客户端与服务器端的搭建、对服务器端和客户端相互通信时相互发送消息的打包发送和解析、游戏房间的建立；

Controller大类主要对从客户端发送过来的各种请求命令做处理，其中一个request对应一个controller，比如客户端向服务器端请求验证登陆时就要从服务器端获取用户信息，于是就放在UserController中处理;

Tool（ConnHelper）大类主要包含服务器端和客户端与数据库的连接；

DAO大类主要是通过对数据库的访问完成相关操作，比如验证登陆信息，注册用户等等；

Model大类主要是与数据库相关的游戏中需要的信息，比如说注册用户的基本信息，包括用户名、密码及对应用户相关信息；

现对整个游戏服务器端框架的说明如下：

1.Servers

包含Clients、Server、Message三个类。

Server类包括对服务器搭建的一系列操作，如绑定IP和端口号，监听客户端的连接，其主要功能是实现服务器端与客户端的通信协议正常工作，其中包括，客户端向服务器端发送请求命令后服务器端接收并解析请求之后发送给客户端，房间的创建等；

Clients类主要功能是配合Server类实现整个通信协议的正常运行，当客户端与服务器端建立连接之后，可以通过实例化Clients对象，完成与客户端之间的交互；

Message类主要负责解析从客户端发送过来的消息消息指令以及对服务器端需要发送给客户端的消息进行打包处理；

Room类则主要负责房间创建相关的操作；

2.Controller

包括ControllerManager、BaseController、DefaultController、GameController、RoomController、UserController几个类。

ControllerManager统一管理所有Controller，主要使用字典类来管理各种Controller，一个request对应一个controller,根据相应的requestCode找到controller，然后根据ActionCode找到controller里面的方法进行调用;

BaseController为各种Controller的基类，里面包含了一些默认的初始化操作；

DefaultController为默认Controller类；

GameController为与游戏相关的Controller，主要处理在对战过程中的服务器与客户端的同步请求。其中包括：StartGame（服务器端转发开始游戏的请求）、Move（服务器端向其他客户端转发关于同步移动的请求）、Shot（服务器端向其他客户端转发关于同步箭的请求）、Attack（在服务器端处理来自客户端的伤害处理）、QuitBattle（服务器端针对客户端离开战斗的请求处理）。

GameController为与游戏用户相关Controller,主要处理登录和注册的请求。其中包括：Login（服务器端处理登陆请求并返回状态,以及取得当前登录成功用户相关信息）、Register（服务器端处理注册请求）。

RoomController为游戏中与房间相关的Controller。主要处理房间相关请求。其中包括：CreateRoom（创建房间）、ListRoom（取得房间列表）、JoinRoom（加入房间）、QuitRoom（退出房间）

3.Tool（ConnHelper）

ConnHelper为数据库连接类，主要实现数据库的连接和关闭；

4.DAO

包括UserDAO、ResultDAO两个类。

UserDAO主要是在数据库中对用户的信息进行查找或者验证，主要放在登录和注册中使用；

ResultDAO主要是在数据库中对对应的用户的战绩信息查询或者更新，在UI界面中显示；

5.Model

包括User、Result两个类。

User类主要是对数据库中User表的各个参数的描述，方便取值；

Result类是对数据库中User对应的战绩信息的参数的描述，方便取值；

二、游戏客户端框架

客户端框架图：

LIIManager 
CameraManager 
AudioManager 
GameFacade 
aaseManager 
PlayerManager 
RequestManager 
aaseRequest 
ClientManager 

在该框架下：主要分为以下几类文件：Net、Requests、Managers、UIPanel、Tools、Player、Model、Camera、GameFacade

1.Net

包括ClientManager、Message两个类。

ClientManager类用来管理跟服务器端的Socket连接；

Message类用来对客户端接收请求命令的解析以及对需要发送到服务器端的指令进行打包处理；

2.Requests

包括BaseRequest、AttackRequest、CreateRoomRequest、GameOverRequest、JoinRoomRequest、ListRoomRequest、LoginRequest、MoveRequest、QuitBattleRequest、QuitRoomRequest、RegisterRequest、ShotRequest、ShowTimerRequest、StartGameRequest、StartPlayRequest、UpdateResultRequest、UpdateRoomRequest类。

BaseRequest类作为Requests基类，包含几个可供子类重写的方法，主要包括Awake、OnResponse、SendRequest、Destroy几个方法；

AttackRequest类为与服务器同步伤害的Request，直接将造成的伤害发送到服务器端进行同步；

MoveRequest类为同步玩家的位置动作信息的Request，并且取得其他客户端的位置动作信息；

ShotRequest类为同步玩家射击行为以及同步实例化出来的箭的位置信息；

LoginRequest类为登录请求，主要发送登录的用户名和密码到服务器端进行验证，如果用户存在，通过服务器端发过来的状态码进行下一步操作，如果成功，则登录成功，并且取得用户信息；

RegisterRequest类为注册请求，主要将需要注册的用户名以及密码发送到服务器端进行注册和得到服务器的响应，根据服务器的响应做出处理；

ListRoomRequest类为列出房间列表请求，服务器端响应之后，显示现有房间列表；

CreateRoomRequest类为创建房间请求，根据服务器具体响应执行相关操作；

JoinRoomRequest类为加入房间请求，根据服务器具体响应执行相关操作；

UpdateRoomRequest类为更新房间去请求，根据服务器具体响应执行相关操作；

QuitRoomRequest类为退出房间请求，根据服务器具体响应执行相关操作；

StartGameRequest类为开始游戏请求，根据服务器具体响应执行相关操作；

StartPlayRequest类为开始游玩请求，根据服务器具体响应执行相关操作；

GameOverRequest类为游戏结束请求，根据服务器具体响应执行相关操作；

UpdateResultRequest类为更新游戏战绩请求，根据服务器具体响应执行相关操作；

3.Managers

包括BaseManager、RequestManager、PlayerManager、UIManager、CameraManager、AudioManager几大类。

BaseManager类作为各个Manager类的基类，包含了一些基本的方法；

RequestManager类作为管理各种Request类的Manager，定义一个字典来存储所有的request，并且提供添加和删除Request的方法，以及根据ActionCode得到对应的Request；

PlayerManager类主要用来管理各种角色相关的操作，比如初始化角色，生产角色等等；

UIManager类则是对各种UI界面的管理，其中包含UI界面的Push和Pop等等；

CameraManager类是对游戏视角切换的管理；

AudioManager类是对游戏音效的管理；

4.UIPanel

主要包括各种UI界面所挂载的脚本；

5.Tools

主要包含在游戏脚本中会经常使用到的几个方法；

6.Player

主要包含与玩家相关的一系列脚本，包括玩家攻击、移动、以及玩家相关信息等等；

7.Model

主要是玩家数据库中的相关数据，包括战绩、密码等；

8.GameFacade

该游戏设计模式的核心。通过使用GameFacade来对各个子系统进行有效的管理，使整个游戏系统结构更加高效清晰。其中主要包含一些实现游戏功能的方法，作为一个中介来调用其他子系统中的各种方法，降低系统的耦合性。

三、服务器端与客户端通信流程

流程图：

RegisterRequest 
Log nRequest 
OnResponse 
Request 
小 … RequestCode … ActlonCode … 
亠 ActlonCOde:ä% 
us CO 彐 = e 「 
CO 彐 日 = e 「 

其通信的流程描述如下：

一个RequestCode对应一个Controller，一个ActionCode对应一个Request，ControllerManager管理所有Controller，RequestManager管理所有的Request；

客户端和服务器端建立连接之后。

1.客户端用户进行操作时向服务器端发送相应操作的请求，其请求命令格式为：数据长度+RequestCode+ActionCode+数据。

服务器端接收到命令请求之后，通过Message类的ReadMessage对命令进行解析后执行回调函数OnProcessMessage，通过该回调函数调用Server类中的HandleRequest方法，然后再在次方法中调用ControllerManager类中的HandleRequest方法（至此整个服务器端接收请求的过程就是Client.RecieveCallBack.message.ReadMessage(OnProcessMessage)->Client.Onprocess.server.HandleRequest->Sever.HandleRequest.controllerManager.HandleRequest->ControllerManager.HandleRequest

,其目的是降低整个工程项目的耦合性）通过其来处理从客户端发送过来的请求命令，根据相应的ActionCode找到与之对应的方法，得到相关方法信息之后通过Invoke去调用，返回一个object对象。

然后通过Server.SendResponse向客户端做出响应（其过程就是ControllerManager.server.SendResponse->Server.cllient.Send->Client.clientSocket.send,其目的性还是为了降低耦合性），将格式为（数据长度+ActionCode+数据（这里的数据就是上面得到的object对象））发送给客户端；

2.服务器端做出响应之后，客户端根据服务器的ActionCode响应找到对应的Request中的OnResponse方法做出响应；对于那些需要向服务器确认消息的客户端操作而言，可以通过Request中的SendRequest来向服务器端再次发出请求，待得到服务器端发出的响应之后再进行下一步的操作；

其具体执行过程为：

首先客户端发送请求：Request.SendRequest()->facade.SendRequest(requestCode, actionCode, data)->clientManager.SendRequest(requestCode, actionCode, data)->clientSocket.Send()；

服务器端接收请求：Clinets.message.ReadMessage()->Clients.OnProcessCallBack->Server.HandleRequest->ControllerManager.HandleRequest->执行ActionCode对应的方法，将方法的返回值返回->server.SendResponse()发送到客户端->客户端接收，解析之后通过ActionCode找到对应的Request，然后执行request.OnResponse()即客户端对服务器端的响应做出响应;