Photon Network

由 Htc Vive 开发者联盟 胡良云（CloudHu）特别翻译

群名称：Htc Vive开发者联盟
群　号：566189328

自己为自给带盐的创意广告：

阿猿/媛，我看你骨骼惊奇，是万中无一的编程奇才，维护宇宙和谐的重任就靠你啦！

…… 

我这里有本祖传的开发秘籍，少年请看：

…… 

「母猴依稀」，拿错了，「歹势」，「歹势啦」！是这本：

…… 

所以，喜欢的同学可以买来看看，不喜欢的同学也可以买来送给喜欢的同学，如果不喜欢的同学还没有认识的喜欢的同学，也可以买来收藏着，等待遇见一个喜欢的同学！购买之前请记得和卖家确认是不是正版，请给正版五星好评，谢谢！

…… 


HTC Vive VR
游戏开发实战
一本关于HTC Vive在VR开发的实战教程书籍
一本技术开发书
亦是一本励志书
免费下载
实例的源代码与素材文件
HTC Vive VR
游戏开发实战
从基础到实战，掌握VR开发必备技能
胡良云 编著
清华大学出版社

编辑推荐

现在VR行业火爆，很多开发者都想开发自己的VR游戏或者应用，本书将深入浅出的讲述**如何开发一款VR游戏**，并且带着读者一步一步地**做游戏案例**，让读者也能够轻松地开发出自己的游戏。

本书跳脱游戏引擎繁杂的操作，由经验丰富的VR游戏设计者，带领读者**在各类型游戏案例中熟悉HTC Vive游戏设计的流程与技巧**，亲手打造创意VR游戏，成就VR开发者。

- ☑ 提供计算机VR游戏开发实战
- ☑ 从创意发想的规划草图开始，用丰富图解说明游戏的制作流程
- ☑ 详解VR各种开发套件
- ☑ HTC Vive的基本配置
- ☑ Unity 3d和Unreal4虚幻双引擎制动


内容简介
本书是目前HTC Vive在VR开发方面解析较为全面的书，也是一本实战教程书籍。如果你不会编程，不会游戏引擎，没关系，本书在附录中为你准备了教学资料和视频分享目录。读者完全可以从零开始，只要坚持学习，就可以开发出VR游戏应用。
01
本书以HTC Vive VR游戏开发实例教学为主线
循序渐进地介绍
针对HTC Vive设备在VR游戏开发方面的整套流程
02
本书还提供了所有实例的源代码与素材文件
供读者上机练习使用
读者可从网上下载本书资源文件
03
本书适用于
广大游戏开发人员、游戏开发爱好者、
软件培训机构以及计算机专业的学生等


作者简介
胡良云 VR公司开发主管
负责HTC Vive开发，也是游戏论坛的专栏作家和译员
工作之余会发布一些专栏文章或者翻译一些国外的教程技术文章
组织成立了HTC Vive开发者联盟，意在促进国内VR行业的技术发展


VRTK传送的另一种方式案例解析
VRTK
案例场景
传输方式
using UnityEngine;
using VRTK;
public class ModelVillage_TeleportLocation : VRTK_DestinationMarker
{
[Tooltip("终点")]public Transform destination;
private bool lastUsePressedState = false; //上一个使用按下状态
/// <summary>
/// 当检测到碰撞时触发终点标记事件
/// </summary>
/// <param name="collider"></param>
vate void OnTriggerStay(Collider collider)
var controller = (collider.GetComponent<VRTK_ControllerEvents>() ? collider.
GetComponent<VRTK_ControllerEvents>() : collider.GetComponentInParent
<VRTK_ControllerEvents>());
if (controller)
{
if (lastUsePressedState == true && !controller.usePressed)
{
var distance = Vector3.Distance(transform.position, destination.position);
OnDestinationMarkerSet(SetDestinationMarkerEvent(distance, destination,
destination.position, (uint)controller.GetComponent<SteamVR_
TrackedObject>().index));
}
lastUsePressedState = controller.usePressed;
}
}
}

由 Htc Vive 开发者联盟 胡良云（CloudHu）特别翻译

VR中国象棋游戏介绍

6.6 VR 中国象棋

本节是一个 VR 策划文案，融入的是对 VR 游戏的理解和经验，同时也想将 VR 带来的可能变革表达出来。VR 改变的不仅仅是游戏或者虚拟的应用，也是我们的思维方式，因为更多的可能也代表希望和机遇。

6.6.1 游戏简介

游戏平台：HTC Vive + Windows 7、10

游戏愿景：作为一个中国象棋爱好者，我希望能够将中国象棋发扬光大，这促使我想要开发一个 VR 版本的中国象棋，这个版本将继承中国象棋原有的精髓，结合 VR 现有的表现能力，同时在游戏中融入楚汉文化，使每一

游戏特色：每一颗棋子都会变成

中军帐，与他对应的红子“帅”变成

中国象棋游戏框架：菜单UI风格是水墨八卦盘，中间是太极图案，随着玩家的点击，这个八卦会旋转改变卦象展示新的层级。这里取的是道家阴阳相生相克，无中生有，一生二，二生三，三生万物，可以衍生出无穷变化。

沿用象棋的规则，当棋子被选中时不但会触发唤醒动画，还会用绿色的路径和点位 标明棋子可选走位，当玩家点击可选的位置时会执行位移并触发移动动画，所以这里没有任何新手教学，只有一个胜利条件提示——击杀敌方首领即可获得胜利。

由 Htc Vive 开发者联盟 胡良云（CloudHu）特别翻译

Photon Network 目录

第一章 主页 25

Introduction | 简介 25

First Steps | 第一步 26

第二章 综合文档 28

2.1 Photon | 概述 28

Exit Games Cloud | 云端 28

Photon Server SDK | Photon 服务器软件开发工具集 29

PUN - First steps | 第一步 30

2.1.1 Master Server And Lobby | 主服务器和大厅 31

2.1.2 Remote Procedure Calls | 远程过程调用 38

Various topics | 多个主题 40

2.1.3 Instantiating Networked Objects | 实例化网络对象 42

Limitations | 限制 46

F.A.Q. | 常见问题 47

将 UNet 项目转化到 Photon 48

第三章 网络模拟 GUI 50

第四章 网络统计界面 51

第 5 章 公共的接口模块 53

第 6 章 模块文档 54

6.1 Public API | 公共 API 54

Classes | 类 54

6.1.1 Detailed Description | 详细说明 60

6.1.2 Enumeration Type Documentation | 枚举类型文档 60

第 7 章 命名空间文档 77

7.1 Package ExitGames | ExitGames 包 77

7.2 Package ExitGames.Client | 客户端包 77

7.3 Package ExitGames.Client.GUI | 客户端 UI 包 77

7.3.1 Enumeration Type Documentation | 枚举类型文档 77

7.4 Package Photon | Photon 包 78

7.4.1 Typedef Documentation | 类型定义文档 78

7.5 Package UnityEngine | Unity 引擎包 78

7.6 Package UnityEngine.SceneManagement 78

第 8 章 类文档 79

8.1 ActorProperties Class Reference | 属性类参考 79

8.1.1 Detailed Description | 详细描述 79

8.1.2 Member Data Documentation | 成员数据文档 79

8.2 AuthenticationValues Class Reference | 类参考 80

8.2.1 Detailed Description | 详细描述 81

8.2.2 Constructor & Destructor Documentation | 构造和析构函数文档 82

8.2.4 Property Documentation | 属性文档 83

8.3 EncryptionDataParameters Class Reference 参考....84
8.3.1 Member Data Documentation | 成员数据文档....85
8.4 ErrorCode Class Reference | 错误代码类参考....85
8.4.1 Detailed Description | 详细描述....88
8.4.2 Member Data Documentation | 成员数据文档....88
8.5 EventCode Class Reference | 事件代码类参考....91
8.5.1 Detailed Description | 详细描述....93
8.5.2 Member Data Documentation | 成员数据文档....93
8.6 Extensions Class Reference | 扩张类参考....93
8.6.1 Detailed Description | 详细描述....95
8.6.2 Member Function Documentation | 成员函数文档....95
8.7 FriendInfo Class Reference | 好友信息类参考....98
8.8 GameObjectExtensions Class Reference....98
8.9 GamePropertyKey Class Reference | 类参考....99
8.9.1 Detailed Description | 详细描述....101
8.9.2 Member Data Documentation | 成员数据文档....101
8.10 ExitGames.Client.GUI.GizmoTypeDrawer....101
8.10.1 Member Function Documentation | 成员函数文档....101
8.11 HelpURL Class Reference | 帮助超链接类参考....101
8.11.1 Detailed Description | 详细描述....102
8.11.2 Constructor & Destructor Documentation | 构造与析构文档102

8.12 IPunCallbacks Interface Reference | 接口参考 102
8.12.1 Detailed Description | 详细描述 104
8.12.2 Member Function Documentation | 成员函数文档 104
8.13 IPunObservable Interface Reference | 接口参考 106
8.14 IPunPrefabPool Interface Reference | 接口参考 107
8.14.1 Detailed Description | 详细描述 107
8.14.2 Member Function Documentation | 成员函数文档 108
8.15 Photon.MonoBehaviour Class Reference 108
8.15.1 Detailed Description | 详细描述 109
8.15.2 Property Documentation | 属性文档 109
8.16 OperationCode Class Reference | 类参考 110
8.16.1 Detailed Description | 详细描述 112
8.17 ParameterCode Class Reference | 类参考 114
8.17.1 Detailed Description | 详细描述 122
8.17.2 Member Data Documentation | 成员数据文档 122
8.18 PhotonAnimatorView Class Reference | 类参考 125
8.18.1 Detailed Description | 详细描述 127
8.18.2 Member Enumeration Documentation | 成员枚举文档 127
8.18.3 Member Function Documentation | 成员函数文档 128
8.19 PhotonLagSimulationGui Class Reference 131
8.19.1 Detailed Description | 详细描述 132

8.19.2 Member Function Documentation | 成员函数文档 132
8.19.3 Member Data Documentation | 成员数据文档 132
8.19.4 Property Documentation | 属性文档 133
8.20 PhotonMessageInfo Struct Reference | 结构 133
8.20.1 Detailed Description | 详细描述 133
8.20.2 Constructor & Destructor Documentation | 构造&析构 134
8.20.3 Member Function Documentation | 成员函数文档 134
8.20.4 Member Data Documentation | 成员数据文档 134
8.20.5 Property Documentation | 属性文档 134
8.21 PhotonNetwork Class Reference | 类参考 134
8.21.1 Detailed Description | 详细描述 149
8.21.2 Member Function Documentation | 成员函数文档 150
8.21.3 Member Data Documentation | 成员数据文档 191
8.21.4 Property Documentation | 属性文档 195
8.22 PhotonPingManager Class Reference | 参考 206
8.22.1 Member Function Documentation | 成员函数文档 207
8.22.2 Member Data Documentation | 成员数据文档 207
8.22.3 Property Documentation | 属性文档 208
8.23 PhotonPlayer Class Reference | 玩家类参考 208
8.23.1 Detailed Description | 详细描述 211
8.23.2 Constructor & Destructor Documentation | 构造和析构文档 211

8.23.3 Member Function Documentation | 成员函数文档 211

8.23.4 Member Data Documentation | 成员数据文档 214

8.23.5 Property Documentation | 属性文档 215

8.24 PhotonRigidbody2DView Class Reference 216

8.24.1 Detailed Description | 详细描述 216

8.24.2 Member Function Documentation | 成员函数文档 217

8.25 PhotonRigidbodyView Class Reference | 参考 218

8.25.1 Detailed Description | 详细描述 218

8.25.2 Member Function Documentation | 成员函数文档 219

8.26 PhotonStatsGui Class Reference | 类参考 220

8.26.1 Detailed Description | 详细描述 221

8.26.2 Member Function Documentation | 成员函数文档 221

8.26.3 Member Data Documentation | 成员数据文档 221

8.27 PhotonStream Class Reference | 类参考 222

8.27.1 Detailed Description | 详细描述 224

8.27.2 Constructor & Destructor Documentation | 构造与析构文档 225

8.27.3 Member Function Documentation | 成员函数文档 225

8.27.4 Property Documentation | 属性文档 226

8.28 PhotonStreamQueue Class Reference | 类参考 227

8.28.1 Detailed Description | 详细描述 228

8.28.2 Constructor & Destructor Documentation | 构造与析构文档 228

8.28.3 Member Function Documentation | 成员函数文档.....228

8.29 PhotonTransformView Class Reference | 参考.....230

8.29.1 Detailed Description | 详细描述.....230

8.29.2 Member Function Documentation | 成员函数文档.....230

8.30 PhotonTransformViewPositionControl Class Reference | 类参考232

8.30.1 Constructor & Destructor Documentation | 构造与析构文档233

8.30.2 Member Function Documentation | 成员函数文档.....233

8.31 PhotonTransformViewPositionModel Class Reference | 类参考..234

8.31.1 Member Enumeration Documentation | 成员枚举文档.....235

8.31.2 Member Data Documentation | 成员数据文档.....236

8.32 PhotonTransformViewRotationControl Class Reference | 类参考237

8.32.1 Constructor & Destructor Documentation | 构造与析构文档238

8.32.2 Member Function Documentation | 成员函数文档.....238

8.33 PhotonTransformViewRotationModel Class Reference | 类参考.238

8.33.1 Member Enumeration Documentation | 成员枚举文档.....239

8.33.2 Member Data Documentation | 成员数据文档.....239

8.34 PhotonTransformViewScaleControl Class Reference | 类参考.....240

8.34.1 Constructor & Destructor Documentation | 构造与析构文档240

8.34.2 Member Function Documentation | 成员函数文档.....240

8.35 PhotonTransformViewScaleModel Class Reference | 类参考.....241

8.35.1 Member Enumeration Documentation | 成员枚举文档.....241

8.35.2 Member Data Documentation | 成员数据文档 241

8.36 PhotonView Class Reference | 类参考 242

8.36.1 Detailed Description | 详细描述 245

8.36.2 Member Function Documentation | 成员函数文档 245

8.36.3 Member Data Documentation | 成员数据文档 250

8.36.4 Property Documentation | 属性文档 251

8.37 PingMonoEditor Class Reference | 类参考 252

8.37.1 Detailed Description | 详细描述 253

8.37.2 Member Function Documentation | 成员函数文档 253

8.38 Photon.PunBehaviour Class Reference | 类参考 253

8.38.1 Detailed Description | 详细描述 257

8.38.2 Member Function Documentation | 成员函数文档 257

8.39 PunRPC Class Reference | PunRPC 类参考 270

8.39.1 Detailed Description | 详细描述 270

8.40 RaiseEventOptions Class Reference | 类参考 270

8.40.1 Detailed Description | 详细描述 271

8.40.2 Member Data Documentation | 成员数据文档 271

8.41 Region Class Reference | 区域类参考 272

8.41.1 Member Function Documentation | 成员函数文档 273

8.41.2 Member Data Documentation | 成员数据文档 273

8.42 Room Class Reference | 房间类参考 273

8.42.1 Detailed Description | 详细描述.....275

8.42.2 Member Function Documentation | 成员函数文档.....275

8.42.3 Property Documentation | 属性文档.....278

8.43 RoomInfo Class Reference | RoomInfo 类参考.....280

8.43.1 Detailed Description | 详细描述.....282

8.43.2 Member Function Documentation | 成员函数文档.....282

8.43.3 Member Data Documentation | 成员数据文档.....283

8.43.4 Property Documentation | 属性文档.....284

8.44 RoomOptions Class Reference | 房间选项类参考.....285

8.44.1 Detailed Description | 详细描述.....287

8.44.2 Member Data Documentation | 成员数据文档.....287

8.44.3 Property Documentation | 属性文档.....289

8.45 UnityEngine.SceneManagement.SceneManager Class Reference | 类参考.....290

8.45.1 Detailed Description | 详细描述.....291

8.45.2 Member Function Documentation | 成员函数文档.....291

8.46 SceneManagerHelper Class Reference | 类参考.....291

8.46.1 Property Documentation | 属性文档.....291

8.47 ServerSettings Class Reference | 类参考.....292

8.47.1 Detailed Description | 详细描述.....293

8.47.2 Member Enumeration Documentation | 成员枚举文档.....293

8.47.3 Member Function Documentation | 成员函数文档 293

8.47.4 Member Data Documentation | 成员数据文档 294

8.48 PhotonAnimatorView.SynchronizedLayer Class Reference | 类参考 295

8.48.1 Member Data Documentation | 成员数据文档 295

8.49 PhotonAnimatorView.SynchronizedParameter Class Reference | 类参考 295

8.49.1 Member Data Documentation | 成员数据文档 295

8.50 TypedLobby Class Reference | 类参考 296

8.50.1 Detailed Description | 详细描述 297

8.50.2 Constructor & Destructor Documentation | 构造与析构文档 297

8.50.3 Member Function Documentation | 成员函数文档 297

8.50.4 Member Data Documentation | 成员数据文档 297

8.50.5 Property Documentation | 属性文档 297

8.51 TypedLobbyInfo Class Reference | 类参考 298

8.51.1 Member Function Documentation | 成员函数文档 298

8.51.2 Member Data Documentation | 成员数据文档 298

8.52 WebRpcResponse Class Reference | 类参考 298

8.52.1 Detailed Description | 详细描述 299

8.52.2 Constructor & Destructor Documentation | 构造与析构文档 299

8.52.3 Member Function Documentation | 成员函数文档 299

8.52.4 Property Documentation | 属性文档 300

第 9 章文件文档 301

9.1 general.md 文件参考 301

9.2 main.md 文件参考 301

9.3 optionalGui.md 文件参考 301

9.4 photonStatsGui.md 文件参考 301

9.5 publicApi.md 文件参考 301

9.6 CustomTypes.cs 文件参考 302

9.6.1 Detailed Description | 详细描述 302

9.7 Enums.cs 文件参考 302

9.7.1 Detailed Description | 详细描述 305

9.7.2 Enumeration Type Documentation | 枚举类型文档 306

9.8 Extensions.cs 文件参考 308

9.8.1 Typedef Documentation | 类型定义文档 308

9.9 FriendInfo.cs 文件参考 309

9.10 GizmoType.cs 文件参考 309

9.11 LoadbalancingPeer.cs 文件参考 310

9.11.1 Enumeration Type Documentation | 枚举类型文档 313

9.12 NetworkingPeer.cs 文件参考 318

9.12.1 Typedef Documentation | 类型定义文档 320

9.12.2 Enumeration Type Documentation | 枚举类型文档 320

9.13 PhotonClasses.cs 文件参考 321

9.13.1 Detailed Description | 详细描述 323
9.13.2 Typedef Documentation | 类型定义文档 323
9.14 PhotonHandler.cs 文件参考 323
9.14.1 Typedef Documentation | 类型定义文档 324
9.15 PhotonLagSimulationGui.cs 文件参考 324
9.15.1 Detailed Description | 详细描述 324
9.16 PhotonNetwork.cs 文件参考 324
9.16.1 Typedef Documentation | 类型定义文档 325
9.17 PhotonPlayer.cs 文件参考 325
9.17.1 Typedef Documentation | 类型定义文档 326
9.18 PhotonStatsGui.cs 文件参考 326
9.18.1 Detailed Description | 详细描述 326
9.19 PhotonStreamQueue.cs 文件参考 326
9.20 PhotonView.cs 文件参考 327
9.20.1 Enumeration Type Documentation | 枚举类型文档 328
9.21 PingCloudRegions.cs 文件参考 329
9.21.1 Typedef Documentation | 类型定义文档 330
9.22 Room.cs 文件参考 330
9.23 RoomInfo.cs 文件参考 330
9.24 RPC.cs 文件参考 331
9.24.1 Detailed Description | 详细描述 331

9.25 RpcIndexComponent.cs 文件参考 331

9.25.1 Detailed Description | 详细描述 332

9.26 ServerSettings.cs 文件参考 332

9.26.1 Detailed Description | 详细描述 332

9.27 SocketWebTcp.cs 文件参考 332

9.28 PhotonAnimatorView.cs 文件参考 333

9.29 PhotonRigidbody2DView.cs 文件参考 333

9.30 PhotonRigidbodyView.cs 文件参考 334

9.31 PhotonTransformView.cs 文件参考 334

9.32 PhotonTransformViewPositionControl.cs 文件参考 335

9.33 PhotonTransformViewPositionModel.cs 文件参考 335

9.34 PhotonTransformViewRotationControl.cs 文件参考 336

9.35 PhotonTransformViewRotationModel.cs 文件参考 336

9.36 PhotonTransformViewScaleControl.cs 文件参考 336

9.37 PhotonTransformViewScaleModel.cs 文件参考 337

由 Htc Vive 开发者联盟 胡良云（CloudHu）特别翻译

Photon Network

Photon Unity Networking

v1.78

Generated by Doxygen 1.8.7

Tue Nov 1 2016 09:53:23

由 Htc Vive 开发者联盟 胡良云（CloudHu）特别翻译

原文：https://www.photonengine.com/zh-CN/PUN

第一章 主页

Introduction | 简介

Photon 是一个快速、简洁和灵活的实时多人游戏开发框架。Photon 包含一个服务器和几个客户端针对主要平台的 SDKs。

Photon Unity Network (简称 PUN)是一个特别的客户端框架，它的目标是重新实现和优化 Unity 内置的网络。在引擎盖下，它使用 Photon 的通信和匹配玩家功能。如图 1-0-0 所示，Unity 中的 Asset Store 有 Exit Games 提供的免费版本下载。

图 1-0-0 PUN

由于 PhotonNetwork 的 API 和 Unity 的内置解决方案是非常类似的，有 Unet

使用经验的用户应该立即有如鱼得水的感觉。一个自动转换器可以帮助你把现有的多玩家项目等效地移植到 Photon。

提供完整的源代码，所以你可以扩展这个插件来支持任何你需要的类型的多玩家游戏。

这个插件是和托管的 Photon Cloud 服务兼容的,Photon 云服务器为你运行 Photon 服务器。 插件里面有一个安装窗口为你在一分钟内免费注册。

最显著的特征：

- 简单得要死的 API
- 提供的服务器可以作为托管服务（开发免费）或作为“企业预置”
- 跨服负载均衡工作流程表（无需额外的开发）
- 性能卓越的 Photon 服务器
- 没有直接的 P2P 和没有 NAT 穿透需要
- 离线模式：在单人游戏模式中重用你的多玩家代码

First Steps | 第一步

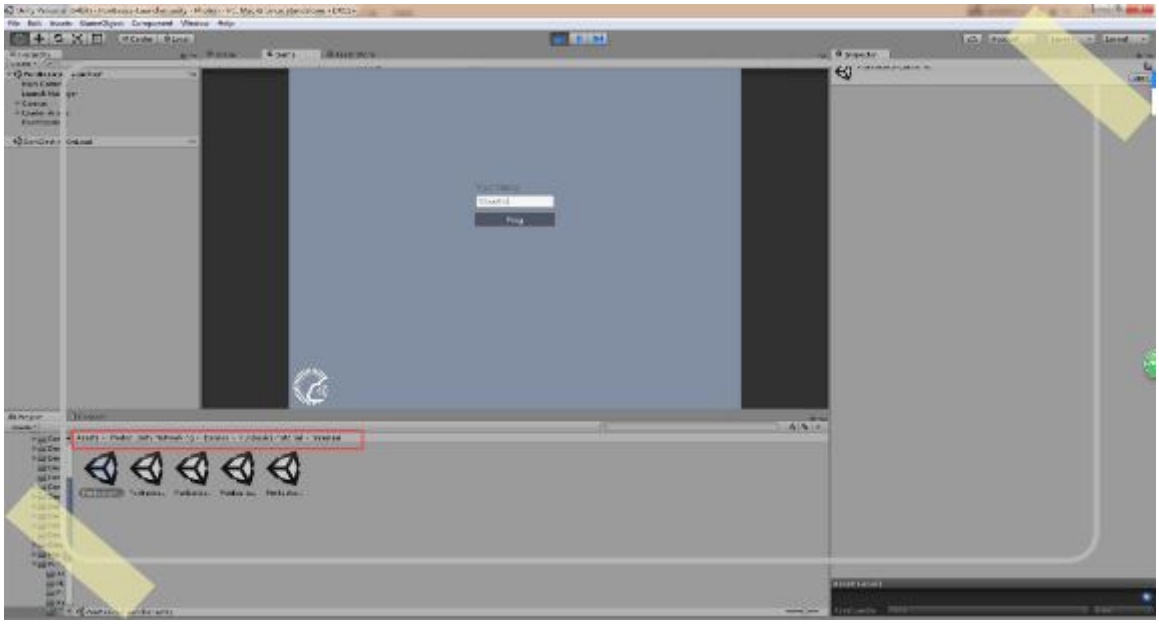
如果你知道怎样使用 Unity 的网络，你应该会觉得如鱼得水。你也许想要在你的一个项目中运行转换器（快捷启动向导：ALT+P）并潜入代码和参考。

General Documentation (API) | 综合文档

继续阅读 General Documentation.

Marco Polo Tutorial | 马可波罗教程

或你可以先做 Marco Polo Tutorial。然后只需参考 General Documentation 即可。



如图 1-0-2 PunBasics-Tutorial

第二章 综合文档

Photon、订阅、托管选项和如何开始的概述。

2.1 Photon | 概述

不像 Unity 的内建网络，PUN 总是连接一个为玩家提供房间、匹配玩家并提供同房间通信的专用服务器。Photon Unity Networking 在后台不止使用一个服务器:

几个“游戏服务器”运行特定的房间（匹配的），同时一个“主服务器”保持追踪各个房间并匹配玩家。

服务器端你有两个选项。

Exit Games Cloud | 云端

Exit Games Cloud 是一个为你提供托管和负载均衡的 Photon 服务器（该服务器由 Exit Games 公司全局管理）的服务。提供免费试用，而且商业通途的订阅成本是相对低廉的。

服务运行一个固定的逻辑，所以你不能实现你自己的服务器端游戏逻辑。相反，客户端需要被授权。

客户端通过“application id”分隔，这关系到您的游戏标题和“游戏版本”。有了这个，你的玩家不会与另一个开发或旧的游戏迭代发生冲突。

Subscriptions bought in Asset Store | 在资源商店购买的订阅

如图 1-0-1 所示，如果你是在资源商店里购买的一个带 Photon 云服务订阅的插件包，跟着这些步骤：

- 注册 Photon 云端账户：exitgames.com/en/Account/SignUp
- 从仪表盘创建一个应用并获取你的 AppID
- 发送一封带如下类容的邮件到 developer@exitgames.com：
  - 你的姓名和公司（如果有的话）
  - 资源商店的发票/购买 ID（购买后会收到带发票的邮件）
  - Photon Cloud AppID（仪表盘获得）

图 1-0-1 PUN+

Photon Server SDK | Photon 服务器软件开发工具集

作为替代的 Photon 云服务，你可以运行自己的服务器和开发服务器端逻辑在我们的“负载平衡”C#解决方案上。这将为您提供服务器逻辑的完全控制。

Photon 服务器 SDK 可以在这里下载：

www.exitgames.com/en/OnPremise/Download

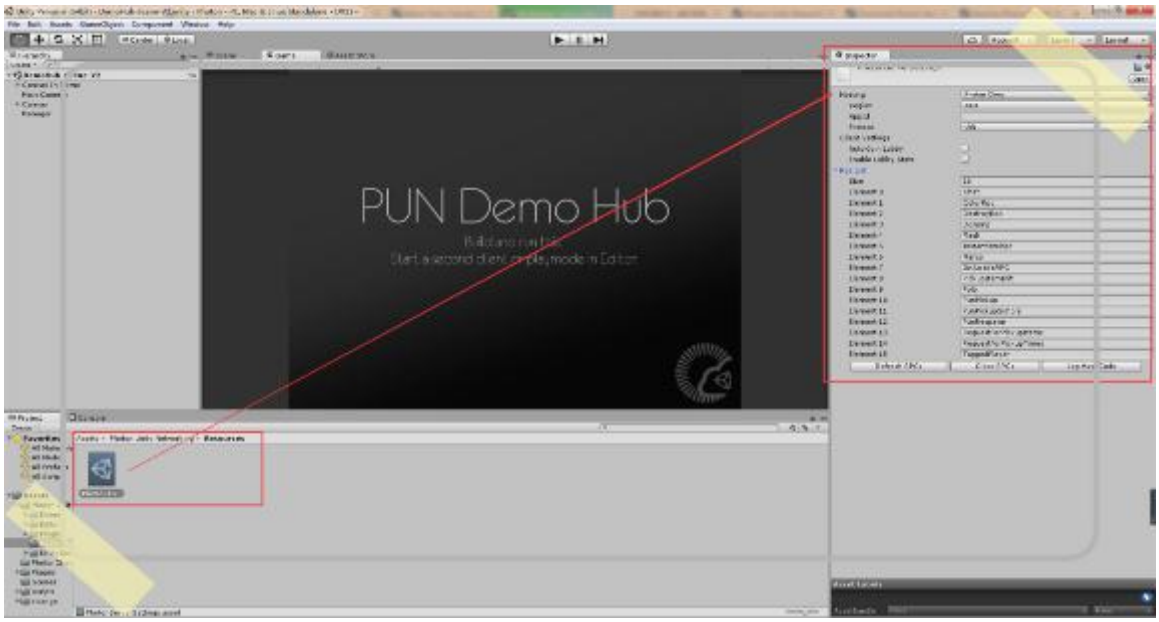
五分钟内启动服务器教程:

doc.exitgames.com/en/onpremise/current/getting-started/photon-server-in-5min

PUN - First steps | 第一步

当你导入 PUN 时，“向导”窗口会弹出。输入您的电子邮件地址登记到云端，跳过这一步输入一个现有的帐户的 AppID 或切换到“自托管” Photon 输入您的服务器的地址。

如下图所示，这将在项目中（在 PhotonServerSettings 里）创建一个云服务或您自己的 Photon 服务器的配置。



PUN 由相当多的文件组成，但只有一个真正重要：**PhotonNetwork**。这个类包含所需的所有函数和变量。如果你有定制要求，你也可以修改源文件-毕竟这个插件只是 Photon 的实现。

要从 UnityScript 中使用 PUN，把"PhotonNetwork" 文件夹和"UtilityScripts"文件夹都移动到 Assets\文件夹下。

为了向你展示这个 API 是如何工作的，这里马上就有几个例子。

2.1.1 Master Server And Lobby | 主服务器和大厅

PUN 总是使用一个主服务器和一个或多个游戏服务器。主服务器管理当前运行在不同的游戏服务器上的游戏，并在当你加入或创建一个房间时将提供一个游戏服务器地址。PUN（客户端）自动切换到游戏服务器。

个人匹配被称为房间。他们是相互独立的，并以名称为标记 ID。房间被分组成一个或多个大厅。大堂是匹配的可选部分。如果你明确地不使用自定义大厅，PUN 将使用一个由所有房间组成的单一的大堂。

默认情况下，在连接后 PUN 将加入默认的大厅。这个大厅向客户发送一个现有的房间列表，所以玩家可以选择一个房间（通过名称或一些列出的属性）。使用 PhotonNetwork.GetRoomList()方法来进入当前列表。列表在时间间隔内被更新来保持低流量。

客户端不必加入一个大厅来加入或创建房间。如果你不想在您的客户端显示房间列表，在你连接之前设置 PhotonNetwork.autoJoinLobby = false，你的客户端将会跳过大堂。

你可以使用一个以上的大厅来组织你游戏所需的房间列表。PhotonNetwork.JoinLobby 是加入一个特定的大厅的方法。你可以在客户端上完成操作-服务器将会保持跟踪它们。只要名称和类型相同，对应的 TypedLobby 将对于所有的客户端都一样。

一个客户总是只在一个大厅，而在一个大厅里，创造一个房间也将涉及到这个大厅。多个大厅意味着客户端得到更短的房间列表，这是很好的。房间列表是无限的。

在 JoinRoom ，JoinRandomRoom 和 CreateRoom 中的一个参数使你可以无需加入就可

以选择大厅。

玩家不会注意到对方在大厅里，也不能发送数据（以防止当它会变得拥挤时发生问题）。

服务器都在专用的机器上运行-没有所谓的玩家托管的“服务器”。您不必费心记住服务器的组织，因为 API 为你都隐藏了。

PhotonNetwork.ConnectUsingSettings(“v1.0“);

上面的代码是任何使用 PhotonNetwork 功能的应用所必须的。它设置你的游戏客户端游戏版本和使用安装向导配置（保存在：PhotonServerSettings，上文中有截图）。当你自己托管 Photon 服务器时也可以使用该向导。另外，使用 Connect()方法连接服务器时你可以忽略 PhotonServerSettings 文件。

Versioning | 版本控制

Photon 的负载均衡逻辑是使用 AppID 区分你的和别人的玩家。同样的游戏版本将新客户端的和旧客户端的玩家区分开来。由于我们不能保证不同 PUN 版本之间相互兼容，我们在你的游戏版本发送之前添加 PUN 版本到你的游戏版本（从 PUN v1.7 版本开始）。

Creating and Joining Games | 创建和加入游戏

下一步，你会想加入或创建一个房间。下面的代码展示了一些必要的功能：

```
//Join a room | 加入一个房间
PhotonNetwork.JoinRoom(roomName);
//Create this room. | 创建这个房间
PhotonNetwork.CreateRoom(roomName);
// Fails if it already exists and calls: OnPhotonCreateGameFailed
```

```
//如果该房间已经存在就会失败，并调用：OnPhotonCreateGameFailed
//Tries to join any random game: | 尝试加入任何随机游戏
PhotonNetwork.JoinRandomRoom();
//Fails if there are no matching games: OnPhotonRandomJoinFailed
//如果没有匹配的游戏就会失败并调用： OnPhotonRandomJoinFailed
```

目前正在运行的游戏列表是由主服务器的大厅提供的。它可以像其他房间一样被加入，但只提供和更新的房间列表。PhotonNetwork 插件会在连接后自动加入大堂。

当你加入一个房间时，列表将不再更新。

显示房间列表（在大厅里）：

```
foreach (RoomInfo game in PhotonNetwork.GetRoomList())
{
    GUILayout.Label(game.name + " " + game.playerCount + "/" + game.maxPlayers);
}
```

另外，游戏可以使用随机配对：它会尝试加入任何房间，并且如果没有一个房间可以容纳该玩家就会失败。在这种情况下：创建一个没有名字的房间，并等待直到其他玩家随机加入。

Advanced Matchmaking & Room Properties | 高级匹配和 Room 属性

完全随机的配对并不总是玩家喜欢的东西。有时你只想玩一个特定的地图或只是 2vs2。

在 Photon 云和负载均衡里，你可以任意设定房间的属性和那些在 JoinRandom 加入的玩家过滤器。

Room Properties and the Lobby | 房间属性和大厅

房间属性是同步到所有房间中的玩家，并在跟踪当前的地图、回合、开始时间等信息时很有用。这些信息被当作字符串键值的散列表来处理。优选短键。

您也可以将选定的属性发送到大厅。这使得它们可以被列表列出，也可以被随机配对。并不是所有大厅里的房间属性都是有趣的，所以你在创建房间的时候定义一系列的属性。

```
Hashtable roomProps = new Hashtable() { { "map", 1 } }; //房间属性哈希表

string[] roomPropsInLobby = { "map", "ai" }; //房间属性字符串数组

RoomOptions roomOptions = new RoomOptions() { customRoomProperties = roomProps,customRoomPropertiesForLobby = roomPropsInLobby } //房间选项

CreateRoom(roomName, roomOptions, TypedLobby.Default) //创建房间
```

请注意“ai”还不是房间属性里的一个键值。它不会出现在大厅，直到它在游戏里通过Room.SetCustomProperties()方法被设置。当你改变“map”或“ai”的值时，他们也会在大厅里在一个很短的延迟后被更新。

保持列表短以确保性能不会受到加载列表影响。

Filtering Room Properties in JoinRandom | 在 JoinRandom 中过滤房间属性

在 JoinRandom 方法中,你可以传递一个包含该方法所期望的房间属性和最大玩家值的哈希表。当服务器为你选择一个“合适的”房间时这些作为过滤器。

```
Hashtable expectedCustomRoomProperties = new Hashtable() { { "map", 1 } };

JoinRandomRoom(expectedCustomRoomProperties, 4);
```

如果你传递更多的过滤器属性，一个房间与它们相匹配的机会较低。最好限制选项。

确保你从不过滤那些大厅没有的属性（见上文）。

MonoBehaviour Callbacks | MonoBehaviour 回调函数

PUN 使用几个回调让你的游戏知道状态的变化，如“连接”或“加入游戏”。所有您需要做的是在任何 MonoBehaviour 中实现合适的方法，并在事件发生时被调用。

为了获得一个良好的可用回调函数的总览，可以在 Photon.PunBehaviour 类中看一看。如果你让你的脚本继承 PunBehaviour（代替 MonoBehaviour），您可以很容易地重写个人回调函数。如果你开始键入“override”关键字，你的编码 IDE 应该为您列出了回调，所以它们在编码时也很容易找到。

这包括建立游戏房间的基本知识。接下来是游戏中的实际交流。

Sending messages in rooms | 在房间里发送信息

在一个房间里，你可以发送网络信息给其他连接的玩家。此外，你能够发送缓冲的消息，缓存消息也将被发送到未来连接的玩家（以生成玩家为例）。

发送消息可以用两种方法来完成。无论是 RPCs 还是利用 PhotonView（Photon 消息视图，详见后文）属性 OnSerializePhotonView。虽然有更多的网络互动。你可以监听回调函数来检测特定的网络事件（如 OnPhotonInstantiate, OnPhotonPlayerConnected）并且你可以触发它们中的一些事件(也就是说会调用实例化方法 PhotonNetwork.Instantiate）。不要担心，如果你困惑的最后一段，下一段我们将解释每一个主题。

Using Groups in PUN | 在 PUN 里使用组

当组在任何 PhotonView 上被改变时是不被同步的。保持 photonviews 在所有客户端上在同一组里是由开发人员决定的，如果需要这样的话。对同样的 photonviews 使用不同的组数在几个客户端会导致一些不一致的行为。

一些网络消息只在接收端检查他们的接收组，即：

- RPCS 是针对单人（或主客户端）
- RPCS 被缓冲(AllBuffered 全部缓冲/OthersBuffered 其他缓冲)
- 这包括 PhotonNetwork.Instantiate（因为它被缓冲）。

这是技术原因：Photon 服务器只支持不被缓存且不是针对特定的演员的信息群体。这在未来可能会改变。

PhotonView | Photon 视图

PhotonView 是用于发送消息(消息是指 RPCs 和 OnSerializePhotonView)的脚本组件。你需要将该脚本挂在你的游戏中的游戏对象上。请注意，PhotonView 和 Unity 里的 NetworkView 很相似。

为了发送消息和可选的实例化/分配其他的 PhotonViews，在你的游戏中每时每刻都需要至少一个 PhotonView.

要添加一个 PhotonView 脚本组件到一个游戏对象上，简单地选择一个游戏对象并使用: “Components/Miscellaneous/PhotonView” 。

Observe Transform | 观察变换

如果你将 Transform 挂到 PhotonView 的观察属性上 ,你可以选择同步 Position、Rotation 和 Scale 或者在玩家身上的这些字段组合。这对那些原型或小游戏可以是一个伟大的帮助。注：任何观察到的值的一个变化将发送所有观察到的值-不仅是单独改变了的值。此外，更新不是平滑的或以内插值替换的。

Observe MonoBehaviour | 观察 MonoBehaviour

一个 PhotonView 可以被设置来观察一个 MonoBehaviour。在这种情况下，该脚本的 OnPhotonSerializeView 方法将会被调用。这种方法被调用来根据该脚本是否被本地玩家控制来读写一个对象的状态。

下面简单的代码显示了如何在几行代码中添加字符状态同步：

```
void OnPhotonSerializeView(PhotonStream stream,PhotonMessageInfo info)
{
    //如果正在写入说明是本地玩家，否则就是网络玩家
    if (stream.isWriting)
    {
        //We own this player: send the others our data
        //我们拥有这个玩家：把我们的数据发送给其他玩家
        stream.SendNext((int)controllerScript._characterState);
        stream.SendNext(transform.position);
        stream.SendNext(transform.rotation);
    }
    else
    {
        //Network player, receive data | 网络玩家，接收信息
        controllerScript._characterState = (CharacterState)(int)stream.ReceiveNext();
        correctPlayerPos = (Vector3)stream.ReceiveNext();
```

```
            correctPlayerRot = (Quaternion)stream.ReceiveNext();
        }
    }
```

如果你发送”ReliableDeltaCompressed”的东西，确保总是以同样的顺序写入数据到二进制流。如果你没有写入数据到 PhotonStream，更新不发送。这在暂停时可能有用。现在，以另一种方式交流：RPCs。

2.1.2 Remote Procedure Calls | 远程过程调用

远程过程调用（RPCs）正是顾名思义：方法可以在同房间的远程客户端上被调用。要启用远程对 MonoBehaviour 中某个方法的调用，你必须应用注解：[PunRPC]。

一个 PhotonView 实例需要在同一游戏对象上，来调用标记功能。

```
[PunRPC]    //在需要远程过程调用的方法前加该注解
void ChatMessage(string a, string b)
{
    Debug.Log("ChatMessage " + a + " " + b);
}
```

要在任何脚本中调用的该方法，你需要获取一个 PhotonView 对象。如果你的脚本是从 Photon. MonoBehaviour 派生的，它就会有一个 photonView 字段。任何 MonoBehaviour 或游戏对象可以使用：PhotonView。Get(this)方法来获得它的 PhotonView 组件，然后在它上面调用 RPC。

```
//获取 PhotonView 组件
```

```
PhotonView photonView = PhotonView.Get(this);
//调用 RPC 方法来远程过程调用上面注解的 ChatMessage 方法
photonView.RPC("ChatMessage", PhotonTargets.All, "jup", "and jup!");
```

因此，不是直接调用目标方法，而是在一个 PhotonView 上调用 RPC()方法。提供调用方法的名称，哪些玩家应该调用该方法，然后提供一个参数列表。

注意：用于 RPC()方法的参数列表必须符合预期的参数个数！如果接收客户端找不到匹配的方法，则会记录一个错误。这个规则有一个例外：RPC 方法的最后一个参数的类型可以是 PhotonMessageInfo 类型，这将为每个调用提供一些语境。

```
[PunRPC]
void ChatMessage(string a, string b, PhotonMessageInfo info)
{
    Debug.Log(String.Format("Info: {0} {1} {2}", info.sender, info.photonView,
info.timestamp));
}
```

Timing for RPCs and Loading Levels | RPCs 和负载水平的时机

RPCs 在远程客户端的具体 PhotonViews 上被调用 ,并且始终以与之匹配的客户端为目标。如果远程客户端不知道匹配的 PhotonView，RPC 丢失。

丢失 RPCs 一个典型的原因是当客户端加载和设置等级时，一个客户更快或在房间里更长一段时间，并发送重要的 RPCs 到那些在其他客户端上还没有加载好的目标对象。同样的情况发生在 RPCs 缓冲时。

解决方法是在加载场景的时候暂停消息队列。下面的代码展示了你可以怎样操作：

```
private IEnumerator MoveToGameScene()
{
    // Temporary disable processing of futher network messages
    //临时禁用远程网络消息的进程
    PhotonNetwork.isMessageQueueRunning = false;
    Application.LoadLevel(levelName);
}
```

或者你可以使用 PhotonNetwork.LoadLevel. 它也会临时禁用消息队列。

禁用消息队列将会延迟进来的和输出的消息，直到消息队列被解锁。显然，当你准备好继续的时候解锁消息队列就非常重要了。

属于上一个加载场景但是仍然到达的远程过程调用 RPCs 将会在这个时候被丢弃。但是你应该能够用 RPC 来在两个场景间定义一个间断。

Various topics | 多个主题

Differences to Unity Networking | 和 Unity 网络之间的区别

1. 主机模型

- Unity 网络是基于服务器-客户端（而不是 P2P！）服务器通过 Unity 客户端运行（所以是通过其中一个玩家，也就是说客户端也可以是服务器，服务器也可以是客户端）
- Photon 也是基于服务器-客户端，但是有一个专用的服务器；不再有由于主机离开而连接中断。

2. 连通性

- Unity 网络与 NAT 穿透协作来提高连接：由于玩家托管网络服务器，常常连接失败是因为防火墙/路由器等。连接永远不能被保证，有一个较低的成功率。
- Photon 有一个专用的服务器，也不需要 NAT 穿透或其他概念。连接是 100%保证。

如果，在罕见的情况下，一个连接失败，它必须是由于一个非常严格的客户端网络（例如一个商业 VPN）。

3. 性能

- Photon 击败 Unity 网络性能。我们还没有数字来证明这一点，但库已经优化了几年了。此外，由于 Unity 服务器是玩家托管，延迟/信号查验通常是更糟；你依靠玩家作为服务器的连接。这些连接永远不会比你的专用 Photon 服务器的连接更好。

4. 价格

- 像 Unity 网络解决方案一样，PUN 插件也是免费的。您可以为您的游戏订阅使用 Photon 云托管服务。或者，你可以租用自己的服务器并在上面运行 Photon。免费许可证可以使多达 100 个玩家同时在线。其他许可证是一次性费用（因为您做托管）并且解除并发用户限制。

5. 特点与维修

- Unity 似乎并没有给他们的网络实现的太多优先。很少有功能改进和错误修正也是很少。Photon 的解决方案是积极维护和部分是可用的源代码。此外，Photon 已经提供了比 Unity 更多的功能，如内置的负载平衡和离线模式。

6. 主服务器

- Photon 主服务器和朴素的 UNet 主服务器有一点不同的是：在我们的例子中，它是一个在所谓的“游戏大厅”里列出当前正在进行的游戏房间名称的 Photon 服务器。像 Unity 的主服务器，它将客户端转化成游戏服务器，实际的游戏在这些客户端服务器中完成。

2.1.3 Instantiating Networked Objects | 实例化网络对象

在每一场游戏你需要为每一个玩家实例化一个或更多的玩家对象。下面列出各种要做的选项。

PhotonNetwork.Instantiate | 实例化方法

PUN 可以通过传递一个起始位置、旋转和预设名到 PhotonNetwork.Instantiate 方法中来自动完成孵化一个对象。要求：预制应可以直接在 Resources/文件夹下找到，这样才可以在运行时加载预置。注意网络播放器：在资源文件夹中的一切都会从默认的第一个场景加载为二进制流。在网络播放器设置下，您可以通过使用“First streamed level”指定第一级通过使用 Resources 文件夹中的资产。如果你为你的第一个游戏场景设置这个，如果你的预加载器和主菜单不使用 Resources 文件夹下的资源，它们将不会被减慢。

```
void SpawnMyPlayerEverywhere()
{
    //实例化方法，参数 1 是 Resources 文件夹下的预设名，2 是 Position，3 是 Rotation
    PhotonNetwork.Instantiate(“MyPrefabName”, new Vector3(0,0,0),
Quaternion.identity, 0);
    //The last argument is an optional group number, feel free to ignore it for now.
```

```
    //最后一个参数是可选组数，现在可以暂时忽略它
}
```

Gain more control: Manually instantiate | 获得更多的控制：手动实例化

如果不想依赖 Resources 文件夹在网络上实例化对象，你必须手动实例化对象，就像这部分最后的的例子所示一样。

想手动实例化的主要原因是为了获得网络播放器所下载的二进制流资源的掌控。关于二进制流和 Unity 的 Resources 文件夹的详细信息可以在这里找到.

如果你手动生成，你必须自己指定一个 PhotonViewID，这些 viewID 是网络消息找到正确的对象/脚本的关键。想要拥有和生成一个新的对象的玩家应该通过使用方法 PhotonNetwork.AllocateViewID();分配一个新的 viewID。这个 PhotonViewID 应该在分配后被发送到所有其他使用一个已经被设置好的 PhotonView 的玩家（例如一个现有的场景 PhotonView）。你将必须要记住这个 RPC 需要被缓冲，这样任何客户端连接后也将获得孵化的指示。然后用来生成的对象的 RPC 消息将需要一个您所需的预制的引用并使用 Unity 的实例化方法 GameObject.Instantiate。最后，你需要通过指定所有 PhotonViews 上的 PhotonViewID 来设置挂在预设上的 PhotonViews。

```
void SpawnMyPlayerEverywhere()
{
    //Manually allocate PhotonViewID | 手动分配 PhotonViewID
    PhotonViewID id1 = PhotonNetwork.AllocateViewID();
    photonView.RPC("SpawnOnNetwork", PhotonTargets.AllBuffered,
```

```
transform.position,transform.rotation, id1, PhotonNetwork.player);
}
public Transform playerPrefab; //set this in the inspector | 设置这个在视窗里
[PunRPC]
void SpawnOnNetwork(Vector3 pos, Quaternion rot, PhotonViewID id1,
PhotonPlayer np)
{
Transform go = Instantiate(playerPrefab, pos, rot) as Transform;
//Set the PhotonView | 设置 PhotonView
PhotonView[] nViews = go.GetComponentsInChildren<PhotonView>();
nViews[0].viewID = id1;
}
```

如果你想使用资源包来加载您的网络对象，所有你要做的就是添加自己的资源包加载代码并用你资源包里的预设替换例子里的”playerPrefab”。

Offline mode | 离线模式

离线模式的一个特点是在单人游戏模式也能够重用你的多人游戏代码。

Mike Hergaarden: 在 M2H 我们不得不重建我们的游戏好几次，因为游戏门户通常要求您完全删除多人游戏功能。此外，在单人和多人游戏中能够使用相同的代码节省自己大量的工作。

你会想要在单人游戏中能够使用最常见的功能是发送 RPCs 和使用

PhotonNetwork.Instantiate。离线模式的主要目标是在使用 PhotonNetwork 功能且未联网时杜绝空引用和其他错误。你仍然需要追踪的事实是你正在运行一个单人游戏，需要设置游戏等。然而，在运行游戏时，所有代码都应该是可重复使用的。

你需要手动启用离线模式，因为 PhotonNetwork 需要能够从预期的行为中区分出错误。启用此功能是非常容易的：

```
PhotonNetwork.offlineMode = true;  //启用离线模式
```

现在，您可以重用某些多人方法，而不产生任何连接和错误。此外，没有明显的开销。下面的列出 PhotonNetwork 函数、变量和它们在离线模式下的结果：

- PhotonNetwork.player 的玩家 ID 总是 -1
- PhotonNetwork.playerName 玩家名称
- PhotonNetwork.playerList 只包含本地玩家
- PhotonNetwork.otherPlayers 总是为空，因为是单人模式
- PhotonNetwork.time 返回 Time.time;
- PhotonNetwork.isMasterClient 总是为真
- PhotonNetwork.AllocateViewID() 分配 ViewID
- PhotonNetwork.Instantiate 实例化方法
- PhotonNetwork.Destroy 摧毁方法
- PhotonNetwork.RemoveRPCs/RemoveRPCsInGroup/SetReceivingEnabled/SetSendingEnabled/SetLevelPrefix 这些方法在单人模式下没什么卵用
- PhotonView.RPC 远程过程调用

请注意，使用其他方法比上面的其他方法可以产生意想不到的结果，有些方法干脆什么都不做。

例如 PhotonNetwork.room 将会明显地返回空 null。如果你打算以单人模式开始游戏，但在游戏后期移到多玩家 ,你可能想要考虑托管一个一个玩家的多人游戏 ,这将保留缓冲的 RPC 和实例化调用，而离线模式的实例不会在连接后自动进行。

无论设置 PhotonNetwork.offlineMode = false; 或简单地调用 Connect()方法来停止离线模式。

Limitations | 限制

Views and players | 视图和玩家

出于性能原因 ,PhotonNetwork API 支持最多 1000 PhotonViews/玩家和最多 2147483 个玩家（注意，这是远高于你的硬件能支持的！）。你可以容易地允许更多 PhotonViews/玩家 ,以减少最大玩家数量为代价。工作方式如下 :PhotonViews 为每一个网络消息发出 viewID。这个 viewID 是一个整数，它是由玩家的 ID 和玩家的视图 ID 组成。

一个整形的最大尺寸是 2147483647，除以我们的 MAX_VIEW_IDS（1000），允许超过 200 万的玩家，各有 1000 个视图 ID。显而易见，你可以通过减少 MAX_VIEW_IDS 来很容易地增加玩家。相反，你可以用更小的最大玩家数量为代价给与所有玩家更多的 VIEW_IDS。重要的是要注意，大多数游戏将永远不需要超过 1000VIEW_IDS（角色只需要一两个，通常是这样的）。如果你需要更多的话，你可能正在做一些错误的事情！将 PhotonView 和 ID 指派到每一颗你的武器发射的子弹是效率极低的，与其通过玩家来跟踪你的子弹，不如直接通过武器的 PhotonView。

通过减少 int 到 short（取值范围：32768 到 32768）来提高你的带宽性能。通过把 MAX_VIEW_IDS 设置成 32，然后你可以仍然支持 1023 个玩家搜索“//LIMITS NETWORKVIEWS&PLAYERS”来寻找所以出现的 viewID。此外，目前的 API 是不使用 uint/ushort，但只有正数范围。这样做是为了简单和 viewIDs 的使用不在大多数情况下不是一个关键的性能问题。

Groups and Scoping | 组和作用域

PhotonNetwork 插件并非全力支持网络组。详见上面的: "Using Groups in PUN".

Unity 的“scope”作用域功能没有实现。

Feedback | 反馈

我们期待你的反馈，因为这个解决方案对我们来说是一个正在进行的项目。如果有什么隐藏、丢失或不工作请告诉我们。要让我们知道，可以在我们的论坛发帖 :forum.exitgames.com

F.A.Q. | 常见问题

我可以每个游戏对象使用多个 PhotonViews 吗？为什么？

是的，完全没有问题。如果你需要观察 2 个或更多的目标你将需要多个 PhotonViews；每个 PhotonView 你只能观察到一个目标。对于你的 RPC 您将只需要一个 PhotonView，并且这可以是相同的已经在观察某物的 PhotonView。RPC 是绝不会与观察到的目标发生冲突。

我能使用 UnityScript / Javascript 吗?

要从 UnityScript 中使用 PUN ,需要把"PhotonNetwork"和"UtilityScripts"文件夹移动到 Assets\ 文件夹下。现在 PUN 会在 UnityScript 之前编译，这样使得它可以从 UnityScript 编码规则中变得可用。

将 UNet 项目转化到 Photon

把你的 UNet 项目转换成 Photon 可以在一天内完成。只是要确定，转换前请先备份您的项目，因为我们的自动转换器将更改您的脚本。这样做后，从 Photon 编辑器窗口（Window -> Photon Unity Networking -> Converter -> Start）运行转换器。自动转换需要在 30 秒到 10 分钟之间，这取决于你的项目的大小和您的计算机的性能。

自动转换包含以下方面:

- 所有的 NetworkViews 都被替换成完全相同设置的 PhotonViews。这是适用于所有的场景和预设。
- 所有调用网络 API 的脚本（JS /BOO/ C #）都会被扫描，这些调用会替换成 PhotonNetwork 调用。

There are some minor differences, therefore you will need to manually fix a few script conversion bugs. After conversion, you will most likely see some compile errors. You’ ll have to fix these first. Most common compile errors:有一些小的差异，因此你将需要手动修复几个脚本转换错误。转换后，您将最有可能看到一些编译错误。你将必须先解决这些问题。最常见的编译错误：

PhotonNetwork.RemoveRPCs(player);PhotonNetwork.DestroyPlayerObjects(player);这些都不存在，并可以安全地被删除。当玩家离开时 Photon 会自动清理玩家（即使你可以禁用这个功能并自己手动清理，如果你想要这么做的话）..

CloseConnection 方法需要两个参数..删除这个调用的参数中的第二个布尔值。

PhotonNetwork.GetPing(player); GetPing 不需要任何参数，你只能请求到 photon 服务器

的连接检测，而不是 ping 给其他玩家。

myPlayerClass.transform.photonView.XXX 错误，你将需要将脚本中这样的代码：myPlayerClass.transform.GetComponent<PhotonView>().XXX ,您可以使用 photonView 获得附加的 PhotonView 组件。然而，你不能在一个外部的 transform 上直接调用这个。

RegisterServer 也不再需要登记你游戏到主服务器，Photon 会自动完成。

你应该能够在 5-30 分钟内解决所有编译错误。大多数的错误都源自主菜单/ GUI 代码，涉及 IPs/端口/大堂 GUI。

这是 Photon 和 Unity 的解决方案最不同的地方：

只有一个 Photon 服务器，并且您使用的房间名称连接。因此，所有涉及 IPs /端口可以从代码中删除（通常是 GUI 代码）。PhotonNetwork.JoinRoom(string room)只需要一个房间名参数，你需要删除旧的 IP /端口/ NAT 参数。如果你已经使用了 M2H 的“Ultimate Unity networking project”，你应该删除 MultiplayerFunctions 类。

最后，所有的旧主服务器调用可以删除。你不需要登记服务器，并且获取房间列表和调用 PhotonNetwork.GetRoomList()方法一样简单。这份列表总是最新的（没有需要 fetch/poll 等多余的操作）。重写的房间列表可以是大部分工作，如果你的 GUI 需要重做，从零开始编写 GUI 可能更简单。

第三章 网络模拟 GUI

简单的 GUI 元素来控制内置网络状态模拟。

Photon 客户端库可以模拟滞后（消息延迟）和损失的网络条件，当测试本地服务器或在接近完美的网络条件时，这对开发者来说是一个很好的工具。

要使用它，在你的场景中添加 PhotonNetSimSettingsGui 组件到一个启用的游戏对象上。在运行时，在屏幕的左上方显示当前的往返时间（RTT，Round Trip Time 首字母缩写，意思是往返时间）和网络模拟控制:

- RTT：往返时间是直到消息被服务器确认的平均毫秒。方差值（在+ / -之后）表明 RTT 是有多稳定（一个较低的值是更好的）。
- "Sim"开关：Simulation 的缩写，启用和禁用模拟。网络条件突然、大的变化可能导致断开。
- "Lag"延迟滑动条：向所有传出和传入消息添加一个固定的延迟。以毫秒为单位。
- "Jit"滑动条：添加一个随机延迟“高达 X 毫秒”每条消息。
- "Loss"丢失滑动条：丢失信息的百分比。在现在的互联网条件下你可以预期不到 2% 的丢失。

第四章 网络统计界面

PhotonStatsGui 是一个简单的 GUI 组件，用来在运行时跟踪并显示的网络指标。

Usage | 用法

只需添加 PhotonStatsGui 组件到层次结构中的任何活动的游戏对象上。一个窗口出现(在运行时），并显示消息计数。

几个开关可以让您配置窗口：

- 按钮：显示“统计”、“复位统计”和“日志”按钮
- 通信量：显示较低级别的网络流量（每个方向的字节数）
- 健康：显示定时发送，调度和最长的差距

Message Statistics | 信息统计

最顶部的值显示为“信息”计数器。任何操作、响应和事件都会被计数。显示的是输出、输入的总计数，以及这些信息总量和平均追踪的 timespan。

Traffic Statistics | 流量统计

这些是字节和数据包计数器。通过网络离开或到达的任何东西都被计算在这里。即使很少有消息，它们可能意外的巨大并仍然造成不太强大的客户端丢失连接。你也可以看到当你不发送消息时也有发包。它们用于保持连接。

Health Statistics | 健康统计

以“最长的三角关系”开始的块是关于您的客户端的性能。我们测量了连续调用发送和调

度之间的时间间隔。通常他们应该被每秒调用十次。

If these values go beyond one second, you should check why Update() calls are delayed.如果这些值超过一秒，你应该检查一下为什么 Update()调用延迟。

Button "Reset" | 重置按钮

这重置统计却一直跟踪他们。跟踪不同情况下的消息计数时很有用。

Button "To Log" | 日志按钮

按这个按钮就会简单地记录当前的属性值。要有一个总览事情是如何演变或只是作为参考时这个按钮可以很有用。

Button "Stats On" (Enabling Traffic Stats) | 启用流量统计按钮

Photon 库可以跟踪各种网络统计，但通常此功能被关闭了。PhotonStatsGui 将启用跟踪和显示这些值。

如果流量统计被全部收集"stats on"会在 Gui 控制里切换。在 Inspector 里的“Traffic Stats On“复选框值是相同的。

第 5 章 公共的接口模块

公共 API 模块入围最常用的 PUN 类。

这些类被组成一个“模块”，以使在 PUN 中更容易找到重要的东西。像 PhotonNetwork 和 Photon.PunBehaviour 类是很好的切入点来学习如何用 PUN 写代码。

与之相反，有几个类是为 PUN 框架内部使用。甚至一些内部使用的类是公共的。这是为了便于使用和配合 Unity 工作的结果。

下一章将详细讲解公共 API 模块。

第 6 章 模块文档

6.1 Public API | 公共 API

组织你需要尽早知道的最重要的类。

Classes | 类

- 接口 interface IPunObservable

定义了 OnPhotonSerializeView 方法来使正确地实现观察脚本更容易。

- 接口 interface IPunCallbacks

这个接口是作为 PUN 的所有回调方法的定义，除了 OnPhotonSerializeView。优选地，单独地实现它们。

- 类 class Photon.PunBehaviour

这个类提供了一个.photonView 字段和所有 PUN 可以调用的回调/事件。重写你想使用的事件/方法。

- 结构体 struct PhotonMessageInfo

关于一个特定的消息、RPC 或更新的信息的容器类。

- 类 class PhotonStream

该容器用于 OnPhotonSerializeView() 函数中，要么提供一个 PhotonView 的输入数据，要么你来提供。

- 类 class PhotonNetwork

使用 PhotonNetwork 插件的主类。这个类是静态的。

- 类 class PhotonPlayer

总结了房间内的一个“玩家”，通过 actorID 来验证身份（在房间里）。

- 类 class PhotonView

PUN 的 NetworkView，网络的替代类。像一个 NetworkView 类一样来使用它。

- 类 class Room

这个类类似于一个 PUN 加入（或已加入）的房间。和 RoomInfo 相反这些属性是可设置的，并且你可以关闭或隐藏的“你的”房间。

- 类 class RoomInfo

一个简化的房间，只需要列表和加入的信息，用于在大厅的房间列表。属性不可设置(open, MaxPlayers，等）。

Enumerations | 枚举

- 网络消息枚举 enum PhotonNetworkingMessage {

PhotonNetworkingMessage.OnConnectedToPhoton,

PhotonNetworkingMessage.OnLeftRoom,

PhotonNetworkingMessage.OnMasterClientSwitched,

PhotonNetworkingMessage.OnPhotonCreateRoomFailed,

PhotonNetworkingMessage.OnPhotonJoinRoomFailed,

PhotonNetworkingMessage.OnCreatedRoom,

PhotonNetworkingMessage.OnJoinedLobby,

PhotonNetworkingMessage.OnLeftLobby,

PhotonNetworkingMessage.OnDisconnectedFromPhoton,

PhotonNetworkingMessage.OnConnectionFail,

PhotonNetworkingMessage.OnFailedToConnectToPhoton,

PhotonNetworkingMessage.OnReceivedRoomListUpdate,

PhotonNetworkingMessage.OnJoinedRoom,

PhotonNetworkingMessage.OnPhotonPlayerConnected,

PhotonNetworkingMessage.OnPhotonPlayerDisconnected,

PhotonNetworkingMessage.OnPhotonRandomJoinFailed,

PhotonNetworkingMessage.OnConnectedToMaster,

PhotonNetworkingMessage.OnPhotonSerializeView,

PhotonNetworkingMessage.OnPhotonInstantiate,

PhotonNetworkingMessage.OnPhotonMaxCccuReached,

PhotonNetworkingMessage.OnPhotonCustomRoomPropertiesChanged,

PhotonNetworkingMessage.OnPhotonPlayerPropertiesChanged,

PhotonNetworkingMessage.OnUpdatedFriendList,

PhotonNetworkingMessage.OnCustomAuthenticationFailed,

PhotonNetworkingMessage.OnCustomAuthenticationResponse,

PhotonNetworkingMessage.OnWebRpcResponse,

PhotonNetworkingMessage.OnOwnershipRequest,

PhotonNetworkingMessage.OnLobbyStatisticsUpdate }

这个枚举定义了 PUN 正在作为回调使用的 MonoMessages。通过 PunBehaviour 实现。

- 日志等级枚举 enum PhotonLogLevel {

PhotonLogLevel.ErrorsOnly,

PhotonLogLevel.Informational,

PhotonLogLevel.Full }

用于定义由 PUN 类创建的日志记录输出的级别。无论是日志错误，信息（更多）或全部。

- 目标枚举 enum PhotonTargets {

PhotonTargets.All,

PhotonTargets.Others,

PhotonTargets.MasterClient,

PhotonTargets.AllBuffered,

PhotonTargets.OthersBuffered,

PhotonTargets.AllViaServer,

PhotonTargets.AllBufferedViaServer }

RPCs 的“目标”选项的枚举。这些定义哪些远程客户端会得到你的 RPC 调用。

- 客户端状态枚举 enum ClientState {

ClientState.Uninitialized,

ClientState.PeerCreated,

ClientState.Queued,

ClientState.Authenticated,

ClientState.JoinedLobby,

ClientState.DisconnectingFromMasterserver,

ClientState.ConnectingToGameserver,

ClientState.ConnectedToGameserver,

ClientState.Joining,

ClientState.Joined,

ClientState.Leaving,

ClientState.DisconnectingFromGameserver,

ClientState.ConnectingToMasterserver,

ClientState.QueuedComingFromGameserver,

ClientState. Disconnecting,

ClientState.Disconnected,

ClientState.ConnectedToMaster,

ClientState.ConnectingToNameServer,

ClientState.ConnectedToNameServer,

ClientState.DisconnectingFromNameServer,

ClientState.Authenticating }

详细连接/网络节点状态。PUN“在幕后”实现了负载均衡和认证流程，所以一些状态将自动前往一些接下来的状态。这些状态被注释为”（即将改变）”。如果不懂英文不用捉急，这

些客户端状态将在后文中详细解释。

- 断连原因枚举 enum DisconnectCause {

DisconnectCause.DisconnectByServerUserLimit =

StatusCode.DisconnectByServerUserLimit,

DisconnectCause.ExceptionOnConnect = StatusCode.ExceptionOnConnect,

DisconnectCause.DisconnectByServerTimeout = StatusCode.DisconnectByServer,

DisconnectCause.DisconnectByServerLogic =

StatusCode.DisconnectByServerLogic,

DisconnectCause.Exception = StatusCode.Exception,

DisconnectCause.InvalidAuthentication = ErrorCode.InvalidAuthentication,

DisconnectCause.MaxCcuReached = ErrorCode.MaxCcuReached,

DisconnectCause.InvalidRegion = ErrorCode.InvalidRegion,

DisconnectCause.SecurityExceptionOnConnect =

StatusCode.SecurityExceptionOnConnect,

DisconnectCause.DisconnectByClientTimeout = StatusCode.TimeoutDisconnect,

DisconnectCause.InternalReceiveException = StatusCode.ExceptionOnReceive,

DisconnectCause.AuthenticationTicketExpired = 32753 }

总结断连的原因。被用于：OnConnectionFail 和 OnFailedToConnectToPhoton

Functions | 函数

- void IPunObservable.OnPhotonSerializeView (PhotonStream stream,

PhotonMessageInfo info)

每秒会被 PUN 调用几次，这样你的脚本可以为 PhotonView 读写同步数据。

6.1.1 Detailed Description | 详细说明

你需要早期了解的最重要类组。

6.1.2 Enumeration Type Documentation | 枚举类型文档

6.1.2.1 enum ClientState | 枚举客户端状态

详细连接/网络节点状态。PUN“在幕后”实现了负载均衡和认证流程，所以一些状态将自动前往一些接下来的状态。这些状态被注释为”（即将改变）”。

Enumerator | 枚举

Uninitialized | 未初始化 不运行。只在初始化和第一次使用之前设置。

PeerCreated | 节点已创建 已创建和可用的连接。

Queued | 已排队 目前不使用。

Authenticated | 已认证 该应用程序已被验证。PUN 通常这个时候加入大堂。（即将改变）除非 AutoJoinLobby 是 false。

JoinedLobby | 已加入的大厅客户在主服务器的大厅里并获取房间列表。使用 Join,、Create 或 JoinRandom 方法来进入一个房间进行游戏。

DisconnectingFromMasterserver | 正在从主服务器断连 正在断连。（即将改变）

ConnectingToGameserver | 正在连接游戏服务器 连接游戏服务器（加入/创建一个房间并进行游戏）（将改变）

ConnectedToGameserver | 已连接游戏服务器 类似于已连接状态，但在游戏服务器上。仍然在过程中加入/创建房间（将改变）

Joining | 正在加入 在加入/创建房间进程中（在游戏服务器上）（将改变）

Joined | 已加入 一个房间加入/创建序列的最终状态。这个客户端现在可以和其他客户端交换事件/调用 RPCs 了。

Leaving | 正在离开 正在离开一个房间（会改变）

DisconnectingFromGameserver | 正在从游戏服务器断开连接 工作流是正在离开游戏服务器，并将重新连接到主服务器。（将更改）

ConnectingToMasterserver 正在连接主服务器 工作流是已经连接到主服务器 ,并将建立加密和验证您的应用程序。（将更改）

QueuedComingFromGameserver | 来自游戏服务器的队列 相同的队列，但来自游戏服务器（将更改）

Disconnecting | 正在断开连接 PUN 正在断开连接。这导致断开连接（将改变）

Disconnected | 已断开连接 没有连接被设置，准备连接。类似 PeerCreated。

ConnectedToMaster | 已连接到主服务器 正在连接到主服务器的最终状态，没有加入大厅的情况下（AutoJoinLobby 是 false）。

ConnectingToNameServer | 正在连接到指定名称的服务器 客户端连接到指定名称的服务器。这个过程包括低级别的连接和设置加密。完成时，状态变为 ConnectedToNameServer。

ConnectedToNameServer | 已连接到指定名称的服务器 客户端被连接到

NameServer 并已经建立的加密。你应该调用 OpGetRegions 或 ConnectToRegionMaster。

DisconnectingFromNameServer | 正在与指定名称服务器断开连接 当从一个 Photon 域名服务器断开连接时（将改变）。

Authenticating | 正在授权 当连接到 Photon 服务器时，这个状态是中间的，在你可以调用任何操作之前（将改变）

6.1.2.2 enum DisconnectCause | 断连原因枚举

总结断开的原因。用于：OnConnectionFail 和 OnFailedToConnectToPhoton。

从 ExitGames.Client.Photon.StatusCode 中提取的状态代码。

See also | 请参阅

PhotonNetworkingMessage

Enumerator | 枚举

DisconnectByServerUserLimit | 由于服务器用户限制 服务器主动断开此客户端。可能的原因：加入的客户端达到了服务器的用户限制，客户被迫断开（正在连接）。

ExceptionOnConnect | 连接时异常 无法建立连接。可能的原因：本地服务器没有运行。

DisconnectByServerTimeout | 服务器超时 服务器超时断连（决定一个 ACK 失踪太久）。

DisconnectByServerLogic | 服务器逻辑断连 服务器主动断开此客户端。可能的原因：服务器的发送缓冲区已满（太多的客户端数据）。

Exception | 异常 一些异常导致连接关闭。

InvalidAuthentication | 无效验证 (32767)Photon 云拒绝派发 AppId。检查你的仪表

板，确保你使用的 AppId 是完整和正确的。

MaxCcuReached | 到达最大在线人数 (32757)因为并发用户（CCU，concurrent users 的缩写，意思是同时上线的用户，并发用户）达到了应用程序的订阅极限，导致 Photon 云授权失。

InvalidRegion | 无效区域 (32756)Photon 云的授权失败，因为应用程序的订阅不允许使用特定区域的服务器。

SecurityExceptionOnConnect | 连接时安全异常 客户端或服务器的安全设置不允许连接（见备注）。发生这种情况的一个常见原因是浏览器客户端从服务器读取一个“跨域”的文件。如果该文件不可用或没有被配置为让客户端连接，则抛出此异常。Photon 常常为 Unity 提供这个跨域文件。如果它失败了，详见：

http://doc.exitgames.com/photon-server/PolicyApp

DisconnectByClientTimeout 客户端超时 客户端超时断开(这决定了一个 ACK 失踪太久）。

InternalReceiveException | 内部接收异常 接收循环中的异常。可能的原因：Socket 故障。

AuthenticationTicketExpired | 身份验证过期 (32753) 身份验证过期了。通过再次连接来处理（会再次进行身份验证以获得一个新的标签）。

6.1.2.3 enum PhotonLogLevel | Photon 日志等级枚举

用于定义由 PUN 类创建的日志记录输出的级别。无论是错误日志，信息（更多）或全部。

Enumerator | 枚举

ErrorsOnly | 只输出错误 只显示错误。最小输出。注意：有些可能是你所期望的“运行时错误”。

Informational | 信息 记录一些工作流、调用和结果。

Full | 全部 每一个可用的日志调用都会进入控制台/日志。只用于调试。

6.1.2.4 enum PhotonNetworkingMessage | Photon 网络消息枚举

这个枚举定义了一整套 PUN 用来作为回调的 MonoMessages。通过 PunBehaviour 实现。

就像 Unity 里的“Update()”一样，PUN 会在特定情况下调用的方法。通常，当网络操作完成时这些方法被触发（例如：当加入一个房间时）。

所有这些方法都在这个枚举中被定义和描述，并通过 PunBehaviour 实现（这使得通过重写来实现变得容易）。

每个条目都是这样一种方法的名称，并且描述告诉你它什么时候会被 PUN 使用。

确保读取每个条目的备注，因为一些方法有可选参数。

Enumerator | 枚举

OnConnectedToPhoton | 处于已连接到 Photon 的状态 当初始连接已被建立时调用，但在您可以使用服务器之前。当 PUN 准备好时调用 OnJoinedLobby()或 OnConnectedToMaster()。这个回调函数只在检测服务器是否可以被完全连接时有用（技术上）。通常，实现 OnFailedToConnectToPhoton()和 OnDisconnectedFromPhoton()就够了。

当 PUN 准备好时调用 OnJoinedLobby()或 OnConnectedToMaster()。

当被调用时，低层次的连接被建立起来并且 PUN 会在后台发送你的 AppId，用户信息等。从主服务器向游戏服务器转换时不会被调用。

例如: void OnConnectedToPhoton() { ... }

OnLeftRoom | 正在离开房间 当本地用户/客户离开房间时调用。当离开一个房间时，PUN 将你带回主服务器。在您可以使用游戏大厅和创建/加入房间之前，OnJoinedLobby()或OnConnectedToMaster()会再次被调用。

例如: void OnLeftRoom() { ... }

OnMasterClientSwitched | 已切换到主客户端 在切换到一个新的主客户端后，且在当前客户端离开时调用。当这个客户进入某个房间时，这是不被调用的。当这个方法被调用时前主客户端仍在玩家列表。

例如: void OnMasterClientSwitched(PhotonPlayer newMasterClient) { ... }

OnPhotonCreateRoomFailed | Photon 创建房间失败 当一个 CreateRoom()方法调用失败时调用。ErrorCode 和消息是可选参数。最有可能是因为房间的名称已经在使用（其他客户端比你更快）。如果 PhotonNetwork.logLevel >= PhotonLogLevel.Informational 为真，PUN 会记录一些信息。

例如: void OnPhotonCreateRoomFailed() { ... }

又例如: void OnPhotonCreateRoomFailed(object[ ] codeAndMsg) {

// codeAndMsg[0]是 short ErrorCode，codeAndMsg[1]是调试消息字符串 }

OnPhotonJoinRoomFailed | 加入房间失败 顾名思义，当一个 JoinRoom()调用失败时被调用。可选参数提供错误代码 ErrorCode 和消息。最有可能的错误是房间不存在或房间已满（一些其他的客户端比你更快）。如果满足条件 PhotonNetwork.logLevel >= PhotonLogLevel.Informational，PUN 会记录一些信息。

例子: void OnPhotonJoinRoomFailed() { ... }

可选参数例子: void OnPhotonJoinRoomFailed(object[ ] codeAndMsg) {

// codeAndMsg[0]是 short 类型的，ErrorCode.codeAndMsg[1] 是 string 类型的调试信息}

OnCreatedRoom | 创建房间 当这个客户端创建了一个房间并进入它时调用。OnJoinedRoom()也会被调用。这个回调只在创建房间的客户端调用（详见 PhotonNetwork.CreateRoom ）。

由于任何客户端在任何时候都可能会关闭（或断开连接），一个房间的创造者有一定的几率不执行 OnCreatedRoom。

如果你需要特定的房间属性或一个“开始信号”，实现 OnMasterClientSwitched()并使新的主客户端检查房间的状态更加安全。

案例: void OnCreatedRoom() { ... }

OnJoinedLobby | 加入大厅 在主服务器上进入一个大厅时调用。实际的房间列表的更新会调用 OnReceivedRoomListUpdate()。注意：当 PhotonNetwork.autoJoinLobby 是 false 时，OnConnectedToMaster()将会被调用并且房间列表将不可用。

而在大堂的房间列表是在固定的时间间隔内自动更新（这是你不能修改的）。当 OnReceivedRoomListUpdate()在 OnJoinedLobby()之后被调用后，房间列表变得可用。

例子: void OnJoinedLobby() { ... }

OnLeftLobby | 离开大厅 离开大厅后被调用。当你离开大厅时，CreateRoom 和 JoinRandomRoom 自动引用默认的大堂。

例如: void OnLeftLobby() { ... }

OnDisconnectedFromPhoton | 从 Photon 断连 从 Photon 服务器断开连接后被调用。在某些情况下，其他回调函数在 OnDisconnectedFromPhoton 被调用之前被调用。例如：OnConnectionFail()和 OnFailedToConnectToPhoton()。

示例: void OnDisconnectedFromPhoton() { ... }

OnConnectionFail | 连接失败 当未知因素导致连接失败（在建立连接之后）时调用，接着调用 OnDisconnectedFromPhoton()。如果服务器不能立即连接，就会调用 OnFailedToConnectToPhoton。错误的原因会以 StatusCode 的形式提供。

例子: void OnConnectionFail(DisconnectCause cause) { ... }

OnFailedToConnectToPhoton | 连接 Photon 失败 如果一个连接到 Photon 服务器请求失败（在连接被建立之前）被调用，接着会调用 OnDisconnectedFromPhoton()。当一个到 Photon 服务器的连接起初就被建立时才会调用 OnConnectionFail。

例如: void OnFailedToConnectToPhoton(DisconnectCause cause) { ... }

OnReceivedRoomListUpdate | 收到房间列表更新 在主服务器上的大厅内（PhotonNetwork.insideLobby）房间列表的任何更新都会调用该函数。PUN 通过 PhotonNetwork.GetRoomList()提供房间列表。

每一项都是一个 RoomInfo，其中可能包括自定义属性（提供你在创建一个房间时定义的那些 lobbylisted）。

不是所有类型的游戏大厅都会提供一系列的房间给客户端。有些游戏大厅是沉默的并且专门做服务器端的匹配（例如英雄联盟的游戏大厅就没有房间列表，所有的房间都是通过服务器

匹配的，自定义则是会提供房间列表）。

例如: void OnReceivedRoomListUpdate() { ... }

OnJoinedRoom | 已加入房间 当进入一个房间（通过创建或加入）时被调用。在所有客户端（包括主客户端）上被调用。这种方法通常用于实例化玩家角色。如果一场比赛必须“积极地”被开始，你也可以调用一个由用户的按键或定时器触发的 PunRPC 。

当这个被调用时，你通常可以通过 PhotonNetwork.playerList 访问在房间里现有的玩家。同时，所有自定义属性 Room.customProperties 应该已经可用。检查 Room.playerCount 就知道房间里是否有足够的玩家来开始游戏。

例子: void OnJoinedRoom() { ... }

OnPhotonPlayerConnected | Photon 玩家已连接 当一个远程玩家进入房间时调用。这个 PhotonPlayer 在这个时候已经被添加 playerlist 玩家列表。如果你的游戏开始时就有一定数量的玩家，这个回调在检查 Room.playerCount 并发现你是否可以开始游戏时会很有用。

用例: void OnPhotonPlayerConnected(PhotonPlayer newPlayer) { ... }

OnPhotonPlayerDisconnected | Photon 玩家掉线 当一个远程玩家离开房间时调用。这个 PhotonPlayer 此时已经从 playerlist 玩家列表删除。当你的客户端调用 PhotonNetwork.leaveRoom 时，PUN 将在现有的客户端上调用此方法。当远程客户端关闭连接或被关闭时，这个回调函数会在经过几秒钟的暂停后被执行。

用例: void OnPhotonPlayerDisconnected(PhotonPlayer otherPlayer) { ... }

OnPhotonRandomJoinFailed | 随机加入房间失败 在一个 JoinRandom()请求失败后调用。可选参数提供 ErrorCode 错误代码和消息。最有可能所有的房间都是满的或没有房间是可用的。当使用多个大厅（通过 JoinLobby 或 TypedLobby），另一个大厅可能有更多/合适的房间。如果 PhotonNetwork.logLevel >= PhotonLogLevel.Informational 为真，PUN 记录一些信息。

用例: void OnPhotonRandomJoinFailed() { ... }

可选参数: void OnPhotonRandomJoinFailed(object[ ] codeAndMsg) {

// codeAndMsg[0] 是 short 类型的 ErrorCode， codeAndMsg[1] 是 string 类型的调试信息 debug msg. }

OnConnectedToMaster | 连接到主服务器 在到主服务器连接被建立和认证后调用，但是只有当 PhotonNetwork.autoJoinLobby 是 false 时才调用。如果你设置 PhotonNetwork.autoJoinLobby 为 true，取而代之调用的是 OnJoinedLobby()。

即使没有在游戏大厅内，你也可以加入房间和创建房间。在这种情况下使用了默认的大厅。

可用房间列表将不可用，除非你通过 PhotonNetwork.joinLobby 加入一个游戏大厅。

用例: void OnConnectedToMaster() { ... }

OnPhotonSerializeView | 序列化视图 实现自定义一个 PhotonView 定期同步的数据。当被 PhotonView 观察时每个“网络更新”调用。这个方法会被那些指派为一个 PhotonView 的观察组件的脚本调用。PhotonNetwork.sendRateOnSerialize 影响调用此方法的频率。PhotonNetwork.sendRate 影响客户端发包的频率。

实现这一方法，你可以自定义哪些数据是需要 PhotonView 定期同步的。您的代码定义了

哪些是正在被发送的（内容）和接收数据的客户端如何使用数据。

不像其他的回调函数，OnPhotonSerializeView 只有在它被分配到一个 PhotonView 作为 PhotonView.observed 脚本时才会被调用。

要使用此方法，PhotonStream 是必不可少的。它将会在控制一个 PhotonView (PhotonStream.isWriting == true)的客户端上的“写入模式”里和在只需接收控制客户端发送数据的远程客户端的“阅读模式”里。

如果您跳过将任何值写入数据流中，PUN 则将跳过更新。仔细使用，这可以节省带宽和消息（每个房间每秒都有一个限制）。

注意当发送者没有发送任何更新时，OnPhotonSerializeView 不会在远程客户端上被调用。这不能作为“每秒 x 次 Update()”使用。

用例: void OnPhotonSerializeView(PhotonStream stream, PhotonMessageInfo info)

{ //这里写你要控制的内容}

OnPhotonInstantiate | Photon 实例化 在一个游戏对象（及其子类）通过使用 PhotonNetwork.Instantiate 方法被实例化时在任何脚本上被调用。

PhotonMessageInfo 参数提供关于谁创建的对象和什么时候创建的（基于 PhotonNetworking.time）的信息。

用例: void OnPhotonInstantiate(PhotonMessageInfo info) { ... }

OnPhotonMaxCccuReached | 到达 Photon 最大在线人数限制 由于并发用户限制(暂时）到达了，该客户端会被服务器拒绝并断连。当这种情况发生时，用户可能会等一会儿再试一次。在 OnPhotonMaxCcuReached()触发时你不能创建或加入房间，因为客户端会断开连

接。你可以用一个新的许可证提高 CCU 的限制（当你使用自己的主机时）或扩展订阅（当使用 Photon 云服务时）。Photon 云会在达到 CCU 极限时给你发邮件。这也是在仪表板（网页）中可见的。

用例: void OnPhotonMaxCccuReached() { ... }

OnPhotonCustomRoomPropertiesChanged | 自定义房间属性改变 当一个房间的自定义属性更改时被调用。propertiesThatChanged 改变的属性包含所有通过 Room.SetCustomProperties 设置的。自从 v1.25 版本这个方法就有一个参数：Hashtable propertiesThatChanged。更改属性必须由 Room.SetCustomProperties 完成，导致这个回调函数局限在本地。

用例: void OnPhotonCustomRoomPropertiesChanged(Hashtable propertiesThatChanged) { ... }

OnPhotonPlayerPropertiesChanged | 玩家属性改变 当自定义玩家属性更改时调用。玩家和更改的属性被传递为对象数组 object[]。自从 v1.25 版本这个方法就有一个参数 :object[] playerAndUpdatedProps，其中包含两个条目。

[0]是被影响的 PhotonPlayer 玩家。

[1]是那些改变了的属性的哈希表。

我们使用一个对象数组 object[]是由于 Unity 的 GameObject.SendMessage 方法的限制（该方法只有一个可选的参数）。

更改属性必须由 PhotonPlayer.SetCustomProperties 完成，导致这个回调局限在本地。

用例:

```
void OnPhotonPlayerPropertiesChanged(object[] playerAndUpdatedProps) {
    PhotonPlayer player = playerAndUpdatedProps[0] as PhotonPlayer;
    Hashtable props = playerAndUpdatedProps[1] as Hashtable;
    //分别将传入的对象数组参数赋值给临时变量，方便实现需要的操作
}
```

OnUpdatedFriendList | 更新好友列表 当服务器发送一个对 FindFriends 请求的响应并更新 PhotonNetwork.Friends 时调用。好友列表可作为 PhotonNetwork.Friends、列出的姓名、在线状态和玩家所在的房间（如果有）。

用例: void OnUpdatedFriendList() { ... }

OnCustomAuthenticationFailed | 自定义授权失败 当自定义身份验证失败时调用。接下来就会断开连接！

自定义身份验证可能会失败，由于用户输入、坏令牌/密匙。如果身份验证是成功的，这种方法不被调用。实现 OnJoinedLobby()或 OnConnectedToMaster()（像往常一样）。

在游戏开发过程中，它也可能由于服务器端的配置错误而失败。在这些情况下，记录 debugMessage 调试消息是非常重要的。

除非您为应用程序（在仪表板）设置一个自定义的身份验证服务，否则将不会被调用！

用例: void OnCustomAuthenticationFailed(string debugMessage) { ... }

OnCustomAuthenticationResponse | 自定义授权回应 当您的自定义身份验证服务用附加数据响应时调用。自定义身份验证服务可以在响应中包含一些自定义数据。当存在时，该数据是在回调函数中作为字典使其可用。而你的数据的键值必须是字符串，值类型可以是字

符串或数字（以 JSON 形式）。你需要额外确定，这个值类型是你期望的。数字成为（目前）Int64。

用例: void OnCustomAuthenticationResponse(Dictionary<string, object> data)

{ ... }

OnWebRpcResponse | 网页远程过程调用回应 当一个 WebRPC 的回应可用时被 PUN 调用。详见 PhotonNetwork.WebRPC。重要：如果 Photon 可以使用你的网页服务 response.ReturnCode 是 0。回应的内容是你的网页服务发送的。你可以从中创建一个 WebResponse 实例。

用例: WebRpcResponse webResponse = new WebRpcResponse(operationResponse);

请注意:OperationResponse 类是在需要被使用的一个命名空间里，即 using ExitGames.Client.Photon; //包含 OperationResponse（和其他的类）。Photon 的 OperationResponse.ReturnCode 返回代码：0 代表“OK”；-3 代表“网页服务没有配置”（详见仪表盘/ 网页钩子）；-5 代表“网页服务现在有 RPC 路径/名称”（至少是 Azure）

用例: void OnWebRpcResponse(OperationResponse response) { ... }

OnOwnershipRequest | 所有权请求 当另一个玩家从你（现在的所有者）这里请求一个 PhotonView 的所有权时调用。参数 viewAndPlayer 包含：

PhotonView view = viewAndPlayer[0] as PhotonView;

PhotonPlayer requestingPlayer = viewAndPlayer[1] as PhotonPlayer;

void OnOwnershipRequest(object[] viewAndPlayer) {} //

OnLobbyStatisticsUpdate | 大厅统计更新 当主服务器发送一个游戏大厅统计更新、更

新 PhotonNetwork.LobbyStatistics 时调用。这个回调有两个前提条件：

EnableLobbyStatistics 必须设置为 true，在客户端连接之前。并且客户端必须连接到主服务器，提供关于大堂的信息。

6.1.2.5 enum PhotonTargets | 枚举 Photon 目标

RPCs 的“目标”选项枚举。这些定义哪个远程客户端得到你的 RPC 调用。

Enumerator | 枚举

All | 所有 发送 RPC 所有其他客户端并在这个客户端上立即执行。后面加入的玩家将不执行此 RPC。

Others | 其他 发送 RPC 所有其他客户端。这个客户端不执行这个 RPC。后面加入的玩家将不执行此 RPC。

MasterClient |主客户端 只发送 RPC 到主客户端。注意：主客户端可能在执行 RPC 之前断开连接，这可能导致 RPCs 丢失。

AllBuffered | 所有缓存 .发送 RPC 到所有其他的客户端，并立即在这个客户端上执行。新玩家在加入时得到这个 RPC，因为它被缓存了（直到这个客户端离开）。

OthersBuffered | 其他缓存 发送 RPC 给所有其他客户端。这个客户端不执行 RPC。新玩家在加入时得到这个 RPC，因为它被缓存了（直到这个客户端离开）。

AllViaServer | 所有通过服务器的 发送 RPC 到通过服务器的所有客户端（包括这个客户端），客户端像任何其他客户端一样在从服务器收到信息后执行 RPC。好处：服务器发送 RPC 顺序是所有的客户端都一样。

AllBufferedViaServer | 所有通过服务器的缓存 发送 RPC 到所有(包括这个客户端)通

过服务器的客户端并为后面加入的玩家缓存，像其他任何客户端一样这个客户端在接收到来自服务器的 RPC 时执行。

好处: The server’ s order of sending the RPCs is the same on all clients.服务器对所有客户端发送 RPCs 的顺序都是一样的。

6.1.3 Function Documentation | 函数文档

6.1.3.1 函数 void IPunObservable.OnPhotonSerializeView (PhotonStream stream, PhotonMessageInfo info)

该函数被 PUN 每秒调用几次，这样你的脚本可以为 PhotonView 读写异步数据。

这个方法会在那些被指派作为 PhotonView 的观察组件的脚本里被调用。

PhotonNetwork.sendRateOnSerialize 影响这个方法被调用的频率。

PhotonNetwork.sendRate 影响这个客户端发包的频率。

实现这一方法，你可以自定义哪些数据是需要 PhotonView 定期同步的。您的代码定义了哪些是正在被发送的（内容）和接收数据的客户端如何使用数据。

不像其他的回调函数，OnPhotonSerializeView 只有在它被分配到一个 PhotonView 作为 PhotonView.observed 脚本时才会被调用。

要使用此方法，PhotonStream 是必不可少的。它将会在控制一个 PhotonView (PhotonStream.isWriting == true)的客户端上的“写入模式”里和在只需接收控制客户端发送数据的远程客户端的“阅读模式”里。

如果您跳过将任何值写入数据流中，PUN 则将跳过更新。仔细使用，这可以节省带宽和消息（每个房间每秒都有一个限制）。

注意当发送者没有发送任何更新时，OnPhotonSerializeView 不会在远程客户端上被调用。这不能作为“每秒 x 次 Update()”使用。

在 PhotonAnimatorView, PhotonTransformView, PhotonRigidbody2DView,和 PhotonRigidbodyView 类里被实现。

6.2 Optional Gui Elements | 可选 GUI 元素

PUN 里有用的 GUI 元素。

Classes | 类

- class PhotonLagSimulationGui

这个 MonoBehaviour 是一个针对 Photon 客户端的网络模拟功能的基本 GUI。它可以修改滞后（固定延迟），抖动（随机滞后）和数据包丢失。

- class PhotonStatsGui

展示连接到 Photon 服务器的网络流量和健康统计的基本 GUI，通过 shift+tab 快捷键开关。

6.2.1 Detailed Description | 详细描述

PUN 有用的 GUI 元素。

第 7 章 命名空间文档

7.1 Package ExitGames | ExitGames 包

Namespaces | 命名空间

- package Client | 客户端包

7.2 Package ExitGames.Client | 客户端包

Namespaces | 命名空间

- package GUI | GUI 包

7.3 Package ExitGames.Client.GUI | 客户端 UI 包

Classes | 小工具类

- class GizmoTypeDrawer

Enumerations | 枚举

- 枚举小工具类型 enum GizmoType { GizmoType.WireSphere, GizmoType.Sphere, GizmoType.WireCube, GizmoType.Cube }

7.3.1 Enumeration Type Documentation | 枚举类型文档

7.3.1.1 客户端 UI 小工具类型 enum ExitGames.Client.GUI.GizmoType

Enumerator | 枚举

WireSphere | 球形线框

Sphere | 球

WireCube | 立方体线框

Cube | 立方体

7.4 Package Photon | Photon 包

Classes | 行为类

- class MonoBehaviour

这个类添加 photonView 属性 ,同时在你的游戏仍在使用 networkView 时记录一个警告。

- class PunBehaviour

这个类提供了一个.photonView 以及 PUN 可以调用所有回调/事件。重写你想使用的事件/方法。

Typedefs | 类型定义

- using Hashtable = ExitGames.Client.Photon.Hashtable

7.4.1 Typedef Documentation | 类型定义文档

7.4.1.1 using Photon.Hashtable = typedef ExitGames.Client.Photon.Hashtable

7.5 Package UnityEngine | Unity 引擎包

Namespaces | 命名空间

- 场景管理包 package SceneManagement

7.6 Package UnityEngine.SceneManagement

Classes | 场景管理类

- class SceneManager

为老版本 Unity 最小化实现的 SceneManager，直到 v5.2 版本。

第 8 章 类文档

8.1 ActorProperties Class Reference | 属性类参考

常量类。这些（字节）值为一个演员/玩家定义了“众所周知的”属性。PUN 在内部使用这些常量。

Public Attributes | 公共属性

- 常量 const byte PlayerName = 255

(255) 一名玩家/演员的名字。

- 常量 const byte IsInactive = 254

(254) 告诉你玩家目前是否在这个游戏（获得事件现场）。

- 常量 const byte UserId = 253

(253) 玩家的 UserId。当用 RoomOptions.PublishUserId = true 创建房间时发送。

8.1.1 Detailed Description | 详细描述

常量类。这些（字节）值为一个演员/玩家定义了“众所周知的”属性。PUN 在内部使用这些常量。

"自定义属性"必须使用一个字符串类型来作为键。它们可以被开发者自由指派。

8.1.2 Member Data Documentation | 成员数据文档

8.1.2.1 常量 const byte ActorProperties.IsInactive = 254

(254) 告诉你玩家目前是否在这个游戏（获得事件现场）。

为异步游戏而设定的一个服务器的设定值，玩家可以离开游戏并稍后返回的地方。

8.1.2.2 常量 const byte ActorProperties.PlayerName = 255

(255) 一名玩家/演员的名字。

8.1.2.3 常量 const byte ActorProperties.UserId = 253

(253) 玩家的 UserId。当用 RoomOptions.PublishUserId = true 创建房间时发送。

8.2 AuthenticationValues Class Reference | 类参考

用于 Photon 中的用户认证的容器。在你连接别的所有客户端被处理前设置 AuthValues。

Public Member Functions | 公共成员函数

- AuthenticationValues ()

创建没有任何信息的空的 auth 值。

- AuthenticationValues (string userId)

创建关于用户的最小信息。认证与否取决于设定的 AuthType。

- virtual void SetAuthPostData (string stringData)

设置数据通过 POST 被传递给认证服务。

- virtual void SetAuthPostData (byte[ ] byteData)

设置数据通过 POST 被传递给认证服务。

- virtual void AddAuthParameter (string key, string value)

添加一个键值对来获取用于自定义验证的参数。

- override string ToString ()

Properties | 属性

- CustomAuthenticationType AuthType [get, set]

应该被使用的自定义身份验证提供者的类型。目前只有“自定义”或“无”（把这个关闭）。

- 字符串 string AuthGetParameters [get, set]

这个字符串必须包含所使用的验证服务期望的任何（HTTP GET）参数。默认情况下，用户名和验证令牌。

- 对象 object AuthPostData [get, set]

通过 POST 传递到认证服务的数据。默认：空（不发送）。无论是字符串类型或字节数组 byte[]（详见设置）。

- 字符串 string Token [get, set]

经过初步验证，Photon 服务器为这个客户端/用户提供了一个身份令牌，随后被用作（缓存）验证。

- 字符串 string UserId [get, set]

每个用户的 UserId 应该是一个唯一标识。这是为了寻找朋友，等等。

8.2.1 Detailed Description | 详细描述

用于 Photon 中的用户认证的容器。在你连接别的所有客户端被处理前设置 AuthValues。

在 Photon 服务器上，用户身份验证是可选的，但在许多情况下是有用的。如果你想用 FindFriends 方法寻找好友，每个用户的唯一 ID 很实用。

用户验证基本上有三种选择：完全没有用户认证，客户端设置一些 UserId，或者也可以使用一些网页服务对用户进行身份验证（并在服务器端设置 UserId）。

自定义身份验证可以让您通过某种登录或令牌来验证最终用户的身份。它会在授权登录或断开客户端之前把用于验证用户的这些值发送到 Photon 服务器。

当你连接时 AuthValues 在 OpAuthenticate 里被发送，所以 AuthValues 必须在你连接前被设置。如果你不设置任何 AuthValues，PUN 将创建 AuthValues 并设置里面的 playerName 作为 userId。如果在 AuthValues 被发送到服务器时 AuthValues.userId 是 null 或空，那么 Photon 服务器会分配一个 userId！

Photon 云的仪表板将让您启用此功能，并为它设置重要的服务器值。

https://www.photonengine.com/dashboard

8.2.2 Constructor & Destructor Documentation | 构造和析构函数文档

8.2.2.1 构造函数 AuthenticationValues.AuthenticationValues ( )

没有任何信息的情况下创建空的 auth 值。

8.2.2.2 构造函数 AuthenticationValues.AuthenticationValues (string userId)

创建关于用户的最小信息。认证与否取决于设定的 AuthType。

Parameters | 参数

userId	字符串类型，是一些在 Photon 服务器里设置的 UserId。

8.2.3 Member Function Documentation | 成员函数文档

8.2.3.1 虚函数 virtual void AuthenticationValues.AddAuthParameter (string key, string value) [virtual]

添加一个键值对来获取用于自定义验证的参数。

这个方法为你完成 URI 编码。

Parameters | 参数

key	参数是字符串类型，用于设置 Key 键值。
value	是有关自定义身份验证的一些值。

8.2.3.2 虚函数 virtual void AuthenticationValues.SetAuthPostData (string stringData) [virtual]

设置那些通过 POST 传递到授权服务的数据。

Parameters | 参数

stringData	参数传递的是将要使用在 POST 请求中的字符串数据。该参数如果为 null 或空字符串将设置 AuthPostData 为 null。

8.2.3.3 虚函数 virtual void AuthenticationValues.SetAuthPostData (byte[ ] byteData) [virtual]

设置那些通过 POST 传递到授权服务的数据。

Parameters | 参数

byteData	参数传递的是二进制令牌/授权数据。

8.2.3.4 override string AuthenticationValues.ToString ( )

8.2.4 Property Documentation | 属性文档

8.2.4.1 string AuthenticationValues.AuthGetParameters [get], [set]

这个字符串必须包含所使用的验证服务期望的任何（HTTP GET）参数。默认情况下，用户名和验证令牌。

标准的 HTTP GET 参数是被用在这里的，并被传递给服务器中定义的服务（Photon 云仪表板）。

8.2.4.2 object AuthenticationValues.AuthPostData [get], [set]

要传递到认证服务的数据通过 POST 被发送。默认：null（不发送）。无论是字符串或字节数组 byte[]（见设置）。

8.2.4.3 CustomAuthenticationType AuthenticationValues.AuthType [get], [set]

应该使用的自定义身份验证提供者的类型。目前只有“自定义”或“无”（把这个关闭）。

8.2.4.4 string AuthenticationValues.Token [get], [set]

经过初步验证，Photon 服务器为这个客户端/用户提供了一个身份令牌，随后被用作（缓存）验证。

8.2.4.5 string AuthenticationValues.UserId [get], [set]

The UserId should be a unique identifier per user. This is for finding friends, etc..See remarks of AuthValues for info about how this is set and used.每个用户的 UserId 应该是一个唯一标识。这是为了查找好友等功能。详情见 AuthValues 中关于 UserId 如何被设置和使用的备注。

8.3 EncryptionDataParameters Class Reference 参考

Public Attributes | 公共属性

- 常量 const byte Mode = 0 //加密模式的键值

- const byte Secret1 = 1 //第一密匙的键值
- const byte Secret2 = 2 //第二密匙的键值

8.3.1 Member Data Documentation | 成员数据文档

8.3.1.1 常量 const byte EncryptionDataParameters.Mode = 0 //加密模式键值

8.3.1.2 const byte EncryptionDataParameters.Secret1 = 1

8.3.1.3 const byte EncryptionDataParameters.Secret2 = 2

8.4 ErrorCode Class Reference | 错误代码类参考

ErrorCode 类定义了默认的与 Photon 客户端/服务器通信相关的编码。

Public Attributes | 公共属性

- const int Ok = 0

(0)总是代表"OK",任何其他错误或具体情况。

- const int OperationNotAllowedInCurrentState = -3

(-3) 表示操作还不能被执行（例如 OpJoin 在被认证之前不能被调用，RaiseEvent 不能在进入一个房间之前被调用）。

- const int InvalidOperationCode = -2

(-2) 您调用的操作在您连接到的服务器（应用程序）上没有实现。确保你运行合适的应用。

- const int InvalidOperation = -2

(-2) 您调用的操作无法在服务器上执行。

- const int InternalServerError = -1

(-1) 服务器上发生了一些错误。尝试该错误并联系 ExitGames 公司。

- const int InvalidAuthentication = 0x7FFF

(32767) 身份验证失败。可能原因：AppId 对于 Photon 服务器来说是未知（云服务）。

- const int GameIdAlreadyExists = 0x7FFF - 1

(32766) GameId（名字）已经在使用（不能创建另一个）。改名字。

- const int GameFull = 0x7FFF - 2

(32765) 游戏已满。这很少发生，一些玩家在你完成加入之前加入了房间。

- const int GameClosed = 0x7FFF - 3

(32764) 游戏是关闭的并且不能加入。加入另一个游戏。

- const int AlreadyMatched = 0x7FFF - 4

- const int ServerFull = 0x7FFF - 5

(32762) 目前不使用。

- const int UserBlocked = 0x7FFF - 6

(32761) 目前不使用。

- const int NoRandomMatchFound = 0x7FFF - 7

(32760) 随机配对只在一个现有的房间既不封闭也没有满的条件下成功。在几秒钟内重复或创建一个新的房间。

- const int GameDoesNotExist = 0x7FFF - 9

(32758) 如果房间（名称）不存在（不再），加入可能会失败。这可能会在当玩家离开而你正好加入时发生。

- const int MaxCcuReached = 0x7FFF - 10

(32757) 在 Photon 云服务器上授权失败，由于并发用户（CCU）的应用程序的订阅达到极限。

- const int InvalidRegion = 0x7FFF - 11

(32756) Photon 云的授权失败，因为应用程序的订阅不允许使用特定区域的服务器。

- const int CustomAuthenticationFailed = 0x7FFF - 12

(32755) 自定义用户身份验证失败，由于安装原因（见云仪表板）或提供的用户数据（如用户名或令牌）。检查错误消息的详细信息。

- const int AuthenticationTicketExpired = 0x7FF1

(32753) 身份验证票过期了。通常情况下，这是在幕后刷新。再次连接（和授权）。

- const int PluginReportedError = 0x7FFF - 15

(32752)一个服务器端的插件（或网页钩子）未能执行并报告了一个错误。检查调试消息 OperationResponse.DebugMessage。

- const int PluginMismatch = 0x7FFF - 16

(32751) CreateRoom/JoinRoom/Join 操作失败，如果预期的插件不符合已加载的。

- const int JoinFailedPeerAlreadyJoined = 32750

(32750)加入请求。指示当前的节点已被调用加入，并被加入到房间里。

- const int JoinFailedFoundInactiveJoiner = 32749

(32749) 加入请求。表明 InactiveActors 列表中已经包含所请求对应的 ActorNr 或 UserId 的演员。

- const int JoinFailedWithRejoinerNotFound = 32748

(32748)加入请求。表示演员名单(积极、消极)不包含所请求的 ActorNr 或 UserId 演员。

- const int JoinFailedFoundExcludedUserId = 32747

(32747) 加入请求。注：为未来使用-表示请求的 UserId 在 ExcludedList 列表中被发现。

- const int JoinFailedFoundActiveJoiner = 32746

(32746) 加入请求。表明 ActiveActors 列表中已经包含所请求对应的 ActorNr 或 UserId 的演员。

- const int HttpLimitReached = 32745

(32745)对于 SetProerties 和 Raisevent(如果旗帜 HttpForward 为 true)请求。表示达到了每分钟最大允许 HTTP 请求。

- const int ExternalHttpCallFailed = 32744

(32744)WebRpc 请求。指示调用外部服务失败。

- const int SlotError = 32742

(32742) 在与槽位预订的匹配过程中的服务器错误。例如，预留槽位不能超过最大玩家数量 MaxPlayers。

- const int InvalidEncryptionParameters = 32741

(32741) 如果由令牌提供的加密参数无效，服务器将反馈此错误。

8.4.1 Detailed Description | 详细描述

ErrorCode 类定义了默认的与 Photon 客户端/服务器通信相关的编码。

8.4.2 Member Data Documentation | 成员数据文档

8.4.2.1 常量 const int ErrorCode.AlreadyMatched = 0x7FFF - 4

8.4.2.2 const int ErrorCode.AuthenticationTicketExpired = 0x7FF1

(32753) 身份验证票过期了。通常情况下，这是在幕后刷新。再次连接（和授权）。

8.4.2.3 const int ErrorCode.CustomAuthenticationFailed = 0x7FFF - 12

8.4.2.4 const int ErrorCode.ExternalHttpCallFailed = 32744

8.4.2.5 const int ErrorCode.GameClosed = 0x7FFF - 3

8.4.2.6 const int ErrorCode.GameDoesNotExist = 0x7FFF - 9

8.4.2.7 const int ErrorCode.GameFull = 0x7FFF - 2

8.4.2.8 const int ErrorCode.GameIdAlreadyExists = 0x7FFF - 1

8.4.2.9 const int ErrorCode.HttpLimitReached = 32745

8.4.2.10 const int ErrorCode.InternalServerError = -1

8.4.2.11 const int ErrorCode.InvalidAuthentication = 0x7FFF

8.4.2.12 const int ErrorCode.InvalidEncryptionParameters = 32741

8.4.2.13 const int ErrorCode.InvalidOperation = -2

(-2) 您调用的操作无法在服务器上执行。

确保您连接到您期望的服务器。

此代码在几个情况下使用：操作的参数可能是不在范围内，完全丢失或冲突。你调用的操作没有在服务器上实现（应用）。服务器端插件会影响可用的操作。

8.4.2.14 const int ErrorCode.InvalidOperationCode = -2

8.4.2.15 const int ErrorCode.InvalidRegion = 0x7FFF - 11

(32756) Photon 云的授权失败，因为应用程序的订阅不允许使用特定区域的服务器。

Photon 云的一些订阅计划是区域绑定的。其他地区的服务器不能使用。

检查您的主服务器地址，并比较它与您的 Photon 云仪表板的信息。

https://cloud.photonengine.com/dashboard

OpAuthorize 是连接工作流的一部分，但只在 Photon 云服务器上，这个错误可能发生。带 CCU 限制执照的自托管 Photon 服务器完全不会让客户端连接。

8.4.2.16 const int ErrorCode.JoinFailedFoundActiveJoiner = 32746

8.4.2.17 const int ErrorCode.JoinFailedFoundExcludedUserId = 32747

8.4.2.18 const int ErrorCode.JoinFailedFoundInactiveJoiner = 32749

8.4.2.19 const int ErrorCode.JoinFailedPeerAlreadyJoined = 32750

8.4.2.20 const int ErrorCode.JoinFailedWithRejoinerNotFound = 32748

8.4.2.21 const int ErrorCode.MaxCcuReached = 0x7FFF - 10

(32757) 在 Photon 云服务器上授权失败，由于并发用户（CCU）的应用程序的订阅达到极限。

除非你有一个应对“CCU 爆满”的计划，客户端可能会在连接过程中在认证步骤失败。受影响的客户端无法调用操作。请注意，结束游戏并返回主服务器的玩家将断开和重新连接。这是一个临时措施。一旦 CCU 低于极限，玩家将能够再次连接到游戏。

8.4.2.22 const int ErrorCode.NoRandomMatchFound = 0x7FFF - 7

8.4.2.23 const int ErrorCode.Ok = 0

8.4.2.24 const int ErrorCode.OperationNotAllowedInCurrentState = -3

(-3) 表示操作还不能被执行（例如 OpJoin 在被认证之前不能被调用，RaiseEvent 不能在

进入一个房间之前被调用）。

在调用云服务器上的任何操作之前，自动化客户端工作流必须完成其授权。

在 PUN 里，等待直到状态是: JoinedLobby (AutoJoinLobby = true)或 ConnectedToMaster (AutoJoinLobby = false)

8.4.2.25 const int ErrorCode.PluginMismatch = 0x7FFF - 16

8.4.2.26 const int ErrorCode.PluginReportedError = 0x7FFF - 15

8.4.2.27 const int ErrorCode.ServerFull = 0x7FFF - 5

8.4.2.28 const int ErrorCode.SlotError = 32742

8.5 EventCode Class Reference | 事件代码类参考

常量类。这些值是由 Photon 的负载均衡定义的事件。PUN 在内部使用这些常量。

Public Attributes | 公共属性

- const byte GameList = 230

(230) 初始化 RoomInfos 列表（在主服务器上的游戏大厅里)

- const byte GameListUpdate = 229

(229) RoomInfos 的更新被合并到“初始化”列表里了(在主服务器上的游戏大厅里)

- const byte QueueState = 228

(228) 目前不使用。服务器满的情况下的排队状态。

- const byte Match = 227

(227) 目前不使用。匹配事件。

- const byte AppStats = 226

(226) 此应用程序的统计事件（玩家，房间等）

- const byte LobbyStats = 224

(224)这一事件提供了一个带玩家和游戏总数的大厅列表。

- const byte AzureNodeInfo = 210

(210) 在托管 Azure 的情况下内部使用。

- const byte Join = (byte)255

(255) 加入事件 :有人加入游戏。新 actorNumber 和演员的属性都提供了(如果在 OpJoin 里设置了）。

- const byte Leave = (byte)254

(254) 离开事件：离开游戏的玩家可以通过 actorNumber 来区分。

- const byte PropertiesChanged = (byte)253

(253) 当你在广播操作“开启”的情况下调用 OpSetProperties，这个事件就会被触发。它包含正在被设置的属性。

- const byte SetProperties = (byte)253

(253) 当你在广播操作“开启”的情况下调用 OpSetProperties，这个事件就会被触发。它包含正在被设置的属性。

- const byte ErrorInfo = 251

(252) 当玩家意外的离开游戏并且房间里有一个 playerTtl > 0，此事件就会被触发来让大家知道超时了。

- const byte CacheSliceChanged = 250

(250) 当事件缓存片被更改时由 Photon 发送。通过 OpRaiseEvent 完成。

8.5.1 Detailed Description | 详细描述

常量类。这些值是由 Photon 的负载均衡定义的事件。PUN 在内部使用这些常量。

这些常量从 255 开始，然后往下走。你自己的游戏内置事件可以从 0 开始。

8.5.2 Member Data Documentation | 成员数据文档

8.5.2.1 const byte EventCode.AppStats = 226

8.5.2.2 const byte EventCode.AzureNodeInfo = 210

8.5.2.3 const byte EventCode.CacheSliceChanged = 250

8.5.2.4 const byte EventCode.ErrorInfo = 251

8.5.2.5 const byte EventCode.GameList = 230

8.5.2.6 const byte EventCode.GameListUpdate = 229

8.5.2.7 const byte EventCode.Join = (byte)255

8.5.2.8 const byte EventCode.Leave = (byte)254

8.5.2.9 const byte EventCode.LobbyStats = 224

8.5.2.10 const byte EventCode.Match = 227

8.5.2.11 const byte EventCode.PropertiesChanged = (byte)253

8.5.2.12 const byte EventCode.QueueState = 228

8.5.2.13 const byte EventCode.SetProperties = (byte)253

8.6 Extensions Class Reference | 扩张类参考

这种静态的类定义了对几个现有类有用的一些扩展方法（如 Vector3，float 及其他）。

Static Public Member Functions | 静态公共成员函数

- static ParameterInfo[ ] GetCachedParemeters (this MethodInfo mo)
- static PhotonView[ ] GetPhotonViewsInChildren (this UnityEngine.GameObject go)
- static PhotonView GetPhotonView (this UnityEngine.GameObject go)
- static bool AlmostEquals (this Vector3 target, Vector3 second, float sqrMagnitudePrecision)

//与目标的平方进行对比 - second 转化成给定的浮点值

- static bool AlmostEquals (this Vector2 target, Vector2 second, float sqrMagnitudePrecision)

//与目标的平方进行对比 - second 转化成给定的浮点值

- static bool AlmostEquals (this Quaternion target, Quaternion second, float maxAngle)

//对比目标与 second 转化成给定的浮点值之间的角度，详见源码

- static bool AlmostEquals (this float target, float second, float floatDiff)

//比较两个浮点数，如果它们直接的差值小于 floatDiff 则返回 true

- static void Merge (this IDictionary target, IDictionary addHash)

//合并所有 addHash 的键到 target 目标，添加新的键并更新 target 里已存在的键值。

- static void MergeStringKeys (this IDictionary target, IDictionary addHash)

//合并键类型字符串到 target 哈希表

- static string ToStringFull (this IDictionary origin)

//调试 IDictionary 内容的帮助方法，包括类型信息。使用这个并非是永久的。

- static string ToStringFull (this object[ ] data)

//调试对象数组 object[]内容的帮助方法。非永久使用。

- static Hashtable StripToStringKeys (this IDictionary original)

//这个方法复制所有字符串类型的原始键到一个新的哈希表里面。

- static void StripKeysWithNullValues (this IDictionary original)

//该方法移除所有含空引用的键值对。Photon 属性通过设置它们的值为 null 来被移除。改变原始传入的 IDictionary！

- static bool Contains (this int[ ] target, int nr)

//检查一个特定的整形值是否在一个整形数组中。

Static Public Attributes | 静态公共字段

- static Dictionary< MethodInfo,ParameterInfo[ ]> parametersOfMethods = new Dictionary<MethodInfo,ParameterInfo[ ]>()

8.6.1 Detailed Description | 详细描述

这种静态为几个现有的类定义了一些有用的扩展方法（如 Vector3，float 及其他）。

8.6.2 Member Function Documentation | 成员函数文档

8.6.2.1 static bool Extensions.AlmostEquals (this Vector3 target, Vector3 second, float sqrMagnitudePrecision)[static]

//对比传入的 target 和 second 的平方值大小

8.6.2.2 static bool Extensions.AlmostEquals (this Vector2 target, Vector2 second, float sqrMagnitudePrecision)[static]

8.6.2.3 static bool Extensions.AlmostEquals (this Quaternion target, Quaternion second, float maxAngle) [static]

8.6.2.4 static bool Extensions.AlmostEquals (this float target, float second, float floatDiff) [static]

8.6.2.5 static bool Extensions.Contains (this int[ ] target, int nr) [static]

Parameters | 参数

target	参数传入要检查的整型数组。
nr	参数传入要检查的整型。

Returns | 返回值

如果整型数组中包含该整型则返回 true，否则返回 false。

8.6.2.6 static ParameterInfo [ ] Extensions.GetCachedParemeters (this MethodInfo mo) [static]

8.6.2.7 static PhotonView Extensions.GetPhotonView (this UnityEngine.GameObject go) [static]

8.6.2.8 static PhotonView [ ] Extensions.GetPhotonViewsInChildren (this UnityEngine.GameObject go) [static]

8.6.2.9 static void Extensions.Merge (this IDictionary target, IDictionary

addHash) [static]

合并所有 addHash 的键到 target 目标，添加新的键并更新 target 里已存在的键值。

Parameters | 参数

target	参数传入需要更新的 IDictionary。
addHash	参数包含即将合并到 target 数据的 IDictionary。

8.6.2.10 static void Extensions.MergeStringKeys (this IDictionary target, IDictionary addHash) [static]

合并键类型字符串到 target 哈希表

不会从 target 中移除键(这样非字符串的键可以在 target 里保留 ,如果该键之前就存在)。

8.6.2.11 static void Extensions.StripKeysWithNullValues (this IDictionary original) [static]

该方法移除所有含空引用的键值对。Photon 属性通过设置它们的值为 null 来被移除。改变原始传入的 IDictionary !

Parameters | 参数

original	参数传入需要去除字符串类型键的原始 IDictonary。

8.6.2.12 static Hashtable Extensions.StripToStringKeys (this IDictionary original) [static]

这个方法复制所有字符串类型的原始键到一个新的哈希表里面。

不递归（！）在哈希表中可能是根哈希值。这不修改原始值。

Parameters | 参数

original	参数传入需要去除字符串类型键的原始 IDictonary。

Returns | 返回值

返回一个新的只包含原有字符串类型的键的哈希表。

8.6.2.13 static string Extensions.ToStringFull (this IDictionary origin) [static]

8.6.2.14 static string Extensions.ToStringFull (this object[ ] data) [static]

8.6.3 Member Data Documentation

8.6.3.1 Dictionary<MethodInfo, ParameterInfo[ ]> Extensions.parametersOfMethods = new Dictionary<MethodInfo,ParameterInfo[ ]>() [static]

8.7 FriendInfo Class Reference | 好友信息类参考

用于存储关于一个朋友的在线状态以及在哪个房间里的信息。

Public Member Functions | 公共成员函数

- override string ToString ()

Properties | 属性

- string Name [get, set]
- bool IsOnline [get, set]
- string Room [get, set]
- bool IsInRoom [get]

8.8 GameObjectExtensions Class Reference

小数量的扩展方法，使 PUN 更容易跨 Unity 版本工作。

Static Public Member Functions | 静态公共成员函数

- static bool GetActive (this GameObject target)

8.9 GamePropertyKey Class Reference | 类参考

常量类。这些（字节）的值用于“常用的”房间/游戏的 Photon 负载均衡。

Pun 在内部使用这些常量

Public Attributes | 公共字段

- const byte MaxPlayers = 255

(255) 适合该房间的最大玩家数量。0 表示“无限制”。

- const byte IsVisible = 254

(254) 使该房间列入或者排除在主服务器上的游戏大厅里。

- const byte IsOpen = 253

(253) 允许更多的玩家加入房间（或不允许）。

- const byte PlayerCount = 252

(252) 当前在房间里的玩家总数。仅在主服务器上的游戏大厅里使用。

- const byte Removed = 251

(251) 如果房间从列表里被移除则为 true（用于更新主服务器上游戏大厅里的房间列表）。

- const byte PropsListedInLobby = 250

(250) 传递到在游戏大厅里 RoomInfo 列表的一系列房间属性。被用于 CreateRoom,该方法定义了每个房间里的这个列表。

- const byte CleanupCacheOnLeave = 249

(249) 等效于加入房间操作的参数 CleanupCacheOnLeave。

- const byte MasterClientId = (byte)248

(248) 由服务器同步的 MasterClientId 的编码,。当作为操作参数发送时是(byte)203。作为房间属性则是(byte)248。

- const byte ExpectedUsers = (byte)247

(247) 房间 ExpectedUsers 所期望用户的编码。匹配机制会为这些 userIDs 的玩家保留一

个槽位。

8.9.1 Detailed Description | 详细描述

与上文重复，这里省略。

8.9.2 Member Data Documentation | 成员数据文档

8.9.2.1 const byte GamePropertyKey.CleanupCacheOnLeave = 249

8.9.2.2 const byte GamePropertyKey.ExpectedUsers = (byte)247

8.9.2.3 const byte GamePropertyKey.IsOpen = 253

8.9.2.4 const byte GamePropertyKey.IsVisible = 254

8.9.2.5 const byte GamePropertyKey.MasterClientId = (byte)248

8.9.2.6 const byte GamePropertyKey.MaxPlayers = 255

8.9.2.7 const byte GamePropertyKey.PlayerCount = 252

8.9.2.8 const byte GamePropertyKey.PropsListedInLobby = 250

8.9.2.9 const byte GamePropertyKey.Removed = 251

8.10 ExitGames.Client.GUI.GizmoTypeDrawer

Static Public Member Functions | 静态公共成员函数

- static void Draw (Vector3 center, GizmoType type, Color color, float size)

8.10.1 Member Function Documentation | 成员函数文档

8.10.1.1 static void ExitGames.Client.GUI.GizmoTypeDrawer.Draw (Vector3 center, GizmoType type, Color color, float size)[static]

8.11 HelpURL Class Reference | 帮助超链接类参考

即将上线的 Unity5.1 版本 HelpURL 的空实现。该类只是为了字段兼容。

继承字段。

Public Member Functions | 公共成员函数

- HelpURL (string url)

8.11.1 Detailed Description | 详细描述

详见：

http://feedback.unity3d.com/suggestions/override-component-documentation-slash-help-link

8.11.2 Constructor & Destructor Documentation | 构造与析构文档

8.11.2.1 HelpURL.HelpURL (string url)

8.12 IPunCallbacks Interface Reference | 接口参考

这个接口是被用作 PUN 所有回调方法的定义，除了 OnPhotonSerializeView。最好单独实现它们。

由 Photon.PunBehaviour 继承。

Public Member Functions | 公共成员函数

- void OnConnectedToPhoton ()
- void OnLeftRoom ()
- void OnMasterClientSwitched (PhotonPlayer newMasterClient)
- void OnPhotonCreateRoomFailed (object[ ] codeAndMsg)
- void OnPhotonJoinRoomFailed (object[ ] codeAndMsg)

- void OnCreatedRoom ()
- void OnJoinedLobby ()
- void OnLeftLobby ()
- void OnFailedToConnectToPhoton (DisconnectCause cause)
- void OnConnectionFail (DisconnectCause cause)
- void OnDisconnectedFromPhoton ()
- void OnPhotonInstantiate (PhotonMessageInfo info)
- void OnReceivedRoomListUpdate ()
- void OnJoinedRoom ()
- void OnPhotonPlayerConnected (PhotonPlayer newPlayer)
- void OnPhotonPlayerDisconnected (PhotonPlayer otherPlayer)
- void OnPhotonRandomJoinFailed (object[ ] codeAndMsg)
- void OnConnectedToMaster ()
- void OnPhotonMaxCccuReached ()
- void OnPhotonCustomRoomPropertiesChanged (Hashtable propertiesThatChanged)
- void OnPhotonPlayerPropertiesChanged (object[ ] playerAndUpdatedProps)
- void OnUpdatedFriendList ()
- void OnCustomAuthenticationFailed (string debugMessage)
- void OnCustomAuthenticationResponse (Dictionary< string, object > data)

- void OnWebRpcResponse (OperationResponse response)
- void OnOwnershipRequest (object[ ] viewAndPlayer)
- void OnLobbyStatisticsUpdate ()

8.12.1 Detailed Description | 详细描述

这个接口是被用作 PUN 所有回调方法的定义，除了 OnPhotonSerializeView。最好单独实现它们。

这个接口还有待完善，但对于一个游戏的实际实现已经足够了。你可以无需实现 IPunCallbacks 的情况下在任何 MonoMehaviour 中单独实现每一个方法。

PUN 通过名称来调用所有的回调函数。不要使用全名实现的回调。例如：IPunCallbacks.OnConnectedToPhoton 不会被 Unity 的 SendMessage()方法调用。

PUN 将会调用那些在任何脚本上实现了回调函数的方法，类似于 Unity 的事件和回调。触发调用的情况在每一个方法中都有描述。

OnPhotonSerializeView 不像其他回调函数一样被调用！它的使用频率比较高并且实现了：IPunObservable。

8.12.2 Member Function Documentation | 成员函数文档

8.12.2.1 void IPunCallbacks.OnConnectedToMaster ( )

8.12.2.2 void IPunCallbacks.OnConnectedToPhoton ( )

8.12.2.3 void IPunCallbacks.OnConnectionFail (DisconnectCause cause)

8.12.2.4 void IPunCallbacks.OnCreatedRoom ( )

8.12.2.5 void IPunCallbacks.OnCustomAuthenticationFailed (string

debugMessage)

8.12.2.6 void IPunCallbacks.OnCustomAuthenticationResponse (Dictionary< string, object > data)

用例: void OnCustomAuthenticationResponse(Dictionary<string, object> data) { ... }

8.12.2.7 void IPunCallbacks.OnDisconnectedFromPhoton ( )

8.12.2.8 void IPunCallbacks.OnFailedToConnectToPhoton (DisconnectCause cause)

8.12.2.9 void IPunCallbacks.OnJoinedLobby ( )

8.12.2.10 void IPunCallbacks.OnJoinedRoom ( )

8.12.2.11 void IPunCallbacks.OnLeftLobby ( )

8.12.2.12 void IPunCallbacks.OnLeftRoom ( )

8.12.2.13 void IPunCallbacks.OnLobbyStatisticsUpdate ( )

8.12.2.14 void IPunCallbacks.OnMasterClientSwitched (PhotonPlayer newMasterClient)

8.12.2.15 void IPunCallbacks.OnOwnershipRequest (object[ ] viewAndPlayer)

8.12.2.16 void IPunCallbacks.OnPhotonCreateRoomFailed (object[ ] codeAndMsg)

8.12.2.17 void IPunCallbacks.OnPhotonCustomRoomPropertiesChanged (Hashtable propertiesThatChanged)

8.12.2.18 void IPunCallbacks.OnPhotonInstantiate (PhotonMessageInfo info)

8.12.2.19 void IPunCallbacks.OnPhotonJoinRoomFailed (object[ ] codeAndMsg)

8.12.2.20 void IPunCallbacks.OnPhotonMaxCccuReached ( )

8.12.2.21 void IPunCallbacks.OnPhotonPlayerConnected (PhotonPlayer newPlayer)

8.12.2.22 void IPunCallbacks.OnPhotonPlayerDisconnected (PhotonPlayer otherPlayer)

8.12.2.23 void IPunCallbacks.OnPhotonPlayerPropertiesChanged (object[ ] playerAndUpdatedProps)

8.12.2.24 void IPunCallbacks.OnPhotonRandomJoinFailed (object[ ] codeAndMsg)

8.12.2.25 void IPunCallbacks.OnReceivedRoomListUpdate ( )

8.12.2.26 void IPunCallbacks.OnUpdatedFriendList ( )

8.12.2.27 void IPunCallbacks.OnWebRpcResponse (OperationResponse response)

8.13 IPunObservable Interface Reference | 接口参考

该接口定义了 OnPhotonSerializeView 方法来使正确地实现可观察脚本更容易。

由 PhotonAnimatorView, PhotonRigidbody2DView, PhotonRigidbodyView 和 PhotonTransformView 继承。

Public Member Functions | 公共成员函数

- void OnPhotonSerializeView (PhotonStream stream, PhotonMessageInfo info)

8.14 IPunPrefabPool Interface Reference | 接口参考

定义了所有对象池必须要实现的方法，这样 PUN 才能使用它。

Public Member Functions | 公共成员函数

- GameObject Instantiate (string prefabId, Vector3 position, Quaternion rotation)

该方法在 PUN 想要创建一个新的完整预设的实例时被调用。必须返回有效的带 PhotonView 的 GameObject 游戏对象。

- void Destroy (GameObject gameObject)

在 PUN 想要摧毁实例时被调用。

8.14.1 Detailed Description | 详细描述

定义了所有对象池必须要实现的方法，这样 PUN 才能使用它。

要使用一个对象池来实例化，你可以设置 PhotonNetwork.ObjectPool。所有的对象都可以使用，只要 ObjectPool 不是 null 即可。当 PUN 调用实例化方法时对象池必须要返回一个有效的非 null 游戏对象。同样，position 位置和 rotation 旋转必须要被应用。

请注意，放入对象池的游戏对象不会得到常用的 Awake 和 Start 方法调用。OnEnable 将会被（你的对象池）调用，但是当调用该方法时网络值还不会被更新。OnEnable 将会为 PhotonView 使用过期的值(isMine，等)。你也许不得不适应脚本。

PUN 将会调用 OnPhotonInstantiate (详见 IPunCallbacks)。这应该被用来设置那些与网络值/所有权相关的重用对象。

8.14.2 Member Function Documentation | 成员函数文档

8.14.2.1 void IPunPrefabPool.Destroy (GameObject gameObject)

该方法在 PUN 想要摧毁预设实例时被调用。

对象池需要知道通过 Destroy()返回游戏对象类型。它可以是一个标签、名称或类似的东西。

Parameters | 参数

gameObject	参数传入要摧毁的实例。

8.14.2.2 GameObject IPunPrefabPool.Instantiate (string prefabId, Vector3 position, Quaternion rotation)

该方法在 PUN 想要创建一个新的完整预设的实例时被调用。必须返回有效的带 PhotonView 的 GameObject 游戏对象。

Parameters | 参数

prefabId	参数传入预设的 id。
position	实例化出来的游戏对象的位置。
rotation	实例化出来的游戏对象所应用的旋转。

Returns | 返回值

返回新的实例化对象，如果该预设没有被找到则返回 null。

8.15 Photon.MonoBehaviour Class Reference

该类添加合适的 photonView，当你的游戏仍然使用 networkView 时会记录一个警告。

继承 MonoBehaviour。

由 Photon.PunBehaviour 和 PhotonView 继承。

Properties | 属性

- PhotonView photonView [get]

一个在这个游戏对象上的 PhotonView 的缓存引用。

- new PhotonView networkView [get]

该属性在这里只是为了当开发者在使用过期值时通知开发者。

8.15.1 Detailed Description | 详细描述

8.15.2 Property Documentation | 属性文档

8.15.2.1 new PhotonView Photon.MonoBehaviour.networkView [get]

该属性在这里只是为了当开发者在使用过期值时通知开发者。

如果 Unity 5.x 版本记录了一个编译警告"Use the new keyword if hiding was intended"或"The new keyword is not required"，你可能遇到了 Editor 编辑器问题。试着用一个 if 定义条件来修改 networkView:

#if UNITY_EDITOR new #endif public PhotonView networkView

8.15.2.2 PhotonView Photon.MonoBehaviour.photonView [get]

在这个游戏对象上的一个缓存引用到的一个 PhotonView 。

如果你打算在脚本里面用到一个 PhotonView，直接写 this.photonView 无疑是最容易的。

如果你打算移除游戏对象上的 PhotonView 组件，但是保留这个 Photon.MonoBehaviour，那么你可以避免使用这个引用或者修改这个代码来使用 PhotonView.Get(obj)替代。

8.16 OperationCode Class Reference | 类参考

常量类。包含操作编码。PUN 在内部使用这些常量。

Public Attributes | 公共字段

- const byte ExchangeKeysForEncryption = 250
- const byte Join = 255

(255) OpJoin 加入房间操作编码,用来加入一个房间。

- const byte AuthenticateOnce = 231

(231) 认证该节点并连接到一个虚拟应用。

- const byte Authenticate = 230

(230)认证该节点并连接到一个虚拟应用。

- const byte JoinLobby = 229

(229) 加入游戏大厅（在主服务器上）。

- const byte LeaveLobby = 228

(228)离开游戏大厅（在主服务器上）。

- const byte CreateGame = 227

(227) 创建一个游戏（如果名称已存在就会创建失败）。

- const byte JoinGame = 226

(226) 加入游戏 (通过名称)。

- const byte JoinRandomGame = 225

(225) 加入随机游戏（在主服务器上）。

- const byte Leave = (byte)254

(254) OpLeave 离开操作编码,用来离开一个房间。

- const byte RaiseEvent = (byte)253

(253) 发起事件（在一个房间里，为其他的演员/玩家）

- const byte SetProperties = (byte)252

(252) 设置（房间或玩家的）属性

- const byte GetProperties = (byte)251

(251) 获取属性。

- const byte ChangeGroups = (byte)248

(248) 在房间里改变利益分组的操作编码（支持轻应用和扩展）。

- const byte FindFriends = 222

(222) 为好友列表请求获取房间和在线状态（按名称，名称应该是唯一的）。

- const byte GetLobbyStats = 221

(221) 请求获取关于一个具体列表的游戏大厅的统计（大厅的用户和游戏计数）。

- const byte GetRegions = 220

(220) 从域名服务器获取区域服务器列表。

- const byte WebRpc = 219

(219) WebRpc 操作。

- const byte ServerSettings = 218

(218) 设置一些服务器设置的操作。在不同的服务器上使用不同的参数。

8.16.1 Detailed Description | 详细描述

常量类。包含操作编码。PUN 在内部使用这些常量。

8.16.2 Member Data Documentation | 成员数据文档

8.16.2.1 const byte OperationCode.Authenticate = 230

(230) 认证该节点并连接到一个虚拟应用。

8.16.2.2 const byte OperationCode.AuthenticateOnce = 231

(231) 认证该节点并连接到一个虚拟应用。

8.16.2.3 const byte OperationCode.ChangeGroups = (byte)248

(248) 在房间里改变利益分组的操作编码（支持轻应用和扩展）。

8.16.2.4 const byte OperationCode.CreateGame = 227

(227) 创建一个游戏（如果名称已存在就会创建失败）。

8.16.2.5 const byte OperationCode.ExchangeKeysForEncryption = 250

8.16.2.6 const byte OperationCode.FindFriends = 222

(222) 为好友列表请求获取房间和在线状态（按名称，名称应该是唯一的）。

8.16.2.7 const byte OperationCode.GetLobbyStats = 221

(221) 请求获取关于一个具体列表的游戏大厅的统计（大厅的用户和游戏计数）。

8.16.2.8 const byte OperationCode.GetProperties = (byte)251

(251)获取属性。

8.16.2.9 const byte OperationCode.GetRegions = 220

(220) 从域名服务器获取区域服务器列表。

8.16.2.10 const byte OperationCode.Join = 255

(255) OpJoin 加入房间操作编码,用来加入一个房间。

8.16.2.11 const byte OperationCode.JoinGame = 226

(226) 加入游戏 (通过名称)。

8.16.2.12 const byte OperationCode.JoinLobby = 229

(229) 加入游戏大厅（在主服务器上）。

8.16.2.13 const byte OperationCode.JoinRandomGame = 225

(225) 加入随机游戏（在主服务器上）。

8.16.2.14 const byte OperationCode.Leave = (byte)254

(254) OpLeave 离开操作编码,用来离开一个房间。

8.16.2.15 const byte OperationCode.LeaveLobby = 228

(228) 离开游戏大厅（在主服务器上）。

8.16.2.16 const byte OperationCode.RaiseEvent = (byte)253

(253) 发起事件（在一个房间里，为其他的演员/玩家）。

8.16.2.17 const byte OperationCode.ServerSettings = 218

(218) 设置一些服务器设置的操作。在不同的服务器上使用不同的参数。

8.16.2.18 const byte OperationCode.SetProperties = (byte)252

(252) 设置（房间或玩家的）属性

8.16.2.19 const byte OperationCode.WebRpc = 219

(219) WebRpc 操作。

8.17 ParameterCode Class Reference | 类参考

常量类。操作和事件的参数编码。PUN 在内部使用这些常量。

Public Attributes | 公共字段

- const byte SuppressRoomEvents = 237

(237) 创建游戏用的一个 bool 参数。如果设置为 true，在连接和离开时没有房间事件被发送到客户端。

默认: false (不发送)。

- const byte EmptyRoomTTL = 236

(236) 当最后一个玩家离开后一个房间的生存时间（TTL，Time To Live 的首字母缩写，也可以叫做生命周期）。保持房间在内存中以备一个玩家很快加入。以毫秒为单位。

- const byte PlayerTTL = 235

(235) 如果客户端断开连接，这个演员第一时间变成无效的并在超过这个事件后被移除。以毫秒为单位。

- const byte EventForward = 234

(234) OpRaiseEvent 和 OpSetCustomProperties 的可选参数 ,用于向网页服务推送事件/操作。

- const byte IsComingBack = (byte)233

(233) OpLeave 在异步游戏中的可选参数。如果是 false,玩家抛弃了游戏（永远）。默认玩家会变成未激活并且可以重新加入。

- const byte IsInactive = (byte)233

(233) 被用在 EvLeave 离开事件里描述用户是否有效（以及可能会回来）或者不再回来。在有 PlayerTTL 玩家生存时间的房间里，默认的情况是离开的玩家变成无效。

- const byte CheckUserOnJoin = (byte)232

(232)被用于创建房间使定义是否任何用户 userid 只可以加入房间一次。

- const byte ExpectedValues = (byte)231

(231) 当改变属性时，"检查和交换" (CAS，Check And Swap 首字母缩写)的编码。

- const byte Address = 230

(230) 一个（游戏）服务器所使用的地址。

- const byte PeerCount = 229

(229)在这个应用程序中的一个房间（用于统计事件）的玩家数量

- const byte GameCount = 228

(228) 此应用程序中的游戏计数（用于统计事件）

- const byte MasterPeerCount = 227

(227)在主服务器上的玩家数（在这个应用程序里寻找房间的）

- const byte UserId = 225

(225) User’ s ID

- const byte ApplicationId = 224

(224) 你的应用程序的身份 ID：在你自己的 Photon 一个名称或在 Photon 云上的一个 GUID。

- const byte Position = 223

(223)目前未使用（作为"Position"位置）。如果你在连接之前排队，这就是你的位置。

- const byte MatchMakingType = 223

(223) 修改用于 OpJoinRandom 匹配算法。允许的参数值在枚举 MatchmakingMode 里被定义。

- const byte GameList = 222

(222) 关于开放/列出的房间的 RoomInfos 房间信息列表。

- const byte Secret = 221

(221) 内部用于建立加密。

- const byte AppVersion = 220

(220) 您的应用程序的版本。

- const byte AzureNodeInfo = 210

(210) 服务器托管在 Azure 的情况下内部使用。

- const byte AzureLocalNodeId = 209

(209) 服务器托管在 Azure 的情况下内部使用。

- const byte AzureMasterNodeId = 208

(208) 服务器托管在 Azure 的情况下内部使用。

- const byte RoomName = (byte)255

(255) gameId/roomName (每个房间都是唯一的)的编码。被用于 OpJoin 和类似的操作。

- const byte Broadcast = (byte)250

(250) OpSetProperties 方法的广播参数编码。

- const byte ActorList = (byte)252

(252) 一个房间里玩家列表的代码。目前不使用。

- const byte ActorNr = (byte)254

(254) 一个操作中的演员的编码。用于属性的获取和设置。

- const byte PlayerProperties = (byte)249

(249) 玩家属性集的编码（哈希表）。

- const byte CustomEventContent = (byte)245

(245) 事件的数据/自定义内容的编码。在 OpRaiseEvent 使用。

- const byte Data = (byte)245

(245) 事件的数据代码。在 OpRaiseEvent 使用。

- const byte Code = (byte)244

(244) 当发送一些代码相关的参数时使用，如 OpRaiseEvent 的事件代码。

- const byte GameProperties = (byte)248

(248) 游戏属性集的代码（哈希表）。

- const byte Properties = (byte)251

(251) 属性集的代码（哈希表）。此键在只发送一组属性时使用。如果无论是 PlayerProperties 或 GameProperties（或两者都）被使用，检查那些键。

- const byte TargetActorNr = (byte)253

(253)一个操作的目标演员的编码。用于属性集。0 是游戏。

- const byte ReceiverGroup = (byte)246

(246) 选择事件（用于 Lite、RaiseEvent 操作）接收器的编码。

- const byte Cache = (byte)247

(247) 在发起事件的同时缓存之的代码。

- const byte CleanupCacheOnLeave = (byte)241

(241) 创建房间操作的布尔参数。如果为 true，服务器清理离开玩家（他们的缓存事件被删除）的房间缓存。

- const byte Group = 240

(240) "组"操作参数的编码(如用于 OpRaiseEvent).

- const byte Remove = 239

(239) “删除”操作参数可以用于从列表中删除一些东西。例如从玩家的兴趣组中删除组。

- const byte PublishUserId = 239

(239) 用 OpJoin 中定义玩家的 UserIds 是否在房间里广播。用于 FindFriends 寻找好友，以及为期望的用户预留蹲位。

- const byte Add = 238

(238) “添加”操作参数可以用于向某些列表或集合中添加一些东西。例如，添加团体到玩家的利益组。

- const byte Info = 218

(218) EventCode.ErrorInfo 和内部调试操作的内容。

- const byte ClientAuthenticationType = 217

(217) 此键的（字节）值定义了目标自定义身份验证类型/服务。在 OpAuthenticate 里使用。

- const byte ClientAuthenticationParams = 216

(216) 此键的（字符串）值提供发送到客户端连接的自定义身份验证类型/服务的参数。在 OpAuthenticate 里使用。

- const byte JoinMode = 215

(215) 使服务器创建一个房间，如果它不存在。OpJoin 使用这个来确保总是可以进入一个房间，除非它已存在并且已满/被关闭。

- const byte ClientAuthenticationData = 214

(214) 此键的（字符串或 byte[]）值提供发送到在 Photon 仪表盘中的自定义验证设置所需的参数。在 OpAuthenticate 里使用。

- const byte MasterClientId = (byte)203

(203) MasterClientId 主客户端身份的编码，由服务器同步。当作为操作参数发送的时候这个代码是 203.

- const byte FindFriendsRequestList = (byte)1

(1) 用于 FindFriends 寻找好友操作请求。值必须是要寻找的好友的字符串数组 string[]。

- const byte FindFriendsResponseOnlineList = (byte)1

(1) 用于响应 FindFriends 操作。包含在线状态的布尔数组 bool[]列表（如不在线则为 false）。

- const byte FindFriendsResponseRoomIdList = (byte)2

(2) 用于响应 FindFriends 操作。包含房间名称（""空字符串表示不知道或没有房间加入的地方）的字符串数组 string[]。

- const byte LobbyName = (byte)213

(213) 用于匹配相关的方法和在创建一个房间时命名一个游戏大厅（来加入或连接一个房间）。

- const byte LobbyType = (byte)212

(212) 用于匹配相关的方法和在创建一个房间时定义一个大厅的类型。与大厅名称相结合，这样就可以识别游戏大厅。

- const byte LobbyStats = (byte)211

(211) 这个（可选）参数可以在 Authenticate 操作中被发送，来打开大堂统计（关于大厅的名称、用户和游戏计数）。详见:PhotonNetwork.Lobbies。

- const byte Region = (byte)210

(210) 用于在 OpAuth 和 OpGetRegions 里的区域价值。

- const byte UriPath = 209

(209) 被调用的 WebRPC 路径。也被称为"WebRpc Name"。类型: string。

- const byte WebRpcParameters = 208

(208) 一个 WebRPC 参数作为： Dictionary<string, object>。这将被序列化成 JSON。

- const byte WebRpcReturnCode = 207

(207) WebRPC 网页远程过程调用的 ReturnCode 返回码,被网页服务发送（并非由 Photon 发送，Photon 使用的是 ErrorCode）. 类型: byte。

- const byte WebRpcReturnMessage = 206

(206) 由 WebRPC 服务器返回的消息。类似于 Photon 的调试信息。类型: string。

- const byte CacheSliceIndex = 205

(205) 用于定义缓存事件的“片”。 片可以很容易地从缓存中删除。类型: int。

- const byte Plugins = 204

(204) 通知服务器的预期插件设置。

- const byte NickName = 202

(202) 当发送客户端的昵称（用户的昵称）时，被用于服务器的操作回应中。

- const byte PluginName = 201

(201) 通知用户关于加载到游戏中插件的名称。

- const byte PluginVersion = 200

(200) 通知用户关于加载到游戏中的插件版本。

- const byte ExpectedProtocol = 195

(195) 将被客户端使用来连接主/游戏服务器的协议。用于域名服务器。

- const byte CustomInitData = 194

(194) 自定义参数集，在认证请求里被发送。

- const byte EncryptionMode = 193

(193) 我们如何去加密数据。

- const byte EncryptionData = 192

(192) 参数的认证，其中包含加密密钥（取决于 AuthMode 和 EncryptionMode）。

8.17.1 Detailed Description | 详细描述

常量类。操作和事件的参数编码。PUN 在内部使用这些常量。

8.17.2 Member Data Documentation | 成员数据文档

8.17.2.1 const byte ParameterCode.ActorList = (byte)252

(252) 一个房间里玩家列表的代码。目前不使用。

8.17.2.2 const byte ParameterCode.ActorNr = (byte)254

8.17.2.3 const byte ParameterCode.Add = 238

8.17.2.4 const byte ParameterCode.Address = 230

8.17.2.5 const byte ParameterCode.ApplicationId = 224

8.17.2.6 const byte ParameterCode.AppVersion = 220

8.17.2.7 const byte ParameterCode.AzureLocalNodeId = 209

8.17.2.8 const byte ParameterCode.AzureMasterNodeId = 208

8.17.2.9 const byte ParameterCode.AzureNodeInfo = 210

8.17.2.10 const byte ParameterCode.Broadcast = (byte)250

8.17.2.11 const byte ParameterCode.Cache = (byte)247

8.17.2.12 const byte ParameterCode.CacheSliceIndex = 205

8.17.2.13 const byte ParameterCode.CheckUserOnJoin = (byte)232

8.17.2.14 const byte ParameterCode.CleanupCacheOnLeave = (byte)241

8.17.2.15 const byte ParameterCode.ClientAuthenticationData = 214

8.17.2.16 const byte ParameterCode.ClientAuthenticationParams = 216

8.17.2.17 const byte ParameterCode.ClientAuthenticationType = 217

8.17.2.18 const byte ParameterCode.Code = (byte)244

8.17.2.19 const byte ParameterCode.CustomEventContent = (byte)245

8.17.2.20 const byte ParameterCode.CustomInitData = 194

8.17.2.21 const byte ParameterCode.Data = (byte)245

8.17.2.22 const byte ParameterCode.EmptyRoomTTL = 236

8.17.2.23 const byte ParameterCode.EncryptionData = 192

8.17.2.24 const byte ParameterCode.EncryptionMode = 193

8.17.2.25 const byte ParameterCode.EventForward = 234

8.17.2.26 const byte ParameterCode.ExpectedProtocol = 195

8.17.2.27 const byte ParameterCode.ExpectedValues = (byte)231

8.17.2.28 const byte ParameterCode.FindFriendsRequestList = (byte)1

8.17.2.29 const byte ParameterCode.FindFriendsResponseOnlineList = (byte)1

8.17.2.30 const byte ParameterCode.FindFriendsResponseRoomIdList = (byte)2

8.17.2.31 const byte ParameterCode.GameCount = 228

8.17.2.32 const byte ParameterCode.GameList = 222

8.17.2.33 const byte ParameterCode.GameProperties = (byte)248

8.17.2.34 const byte ParameterCode.Group = 240

8.17.2.35 const byte ParameterCode.Info = 218

8.17.2.36 const byte ParameterCode.IsComingBack = (byte)233

8.17.2.37 const byte ParameterCode.IsInactive = (byte)233

8.17.2.38 const byte ParameterCode.JoinMode = 215

8.17.2.39 const byte ParameterCode.LobbyName = (byte)213

8.17.2.40 const byte ParameterCode.LobbyStats = (byte)211

8.17.2.41 const byte ParameterCode.LobbyType = (byte)212

8.17.2.42 const byte ParameterCode.MasterClientId = (byte)203

8.17.2.43 const byte ParameterCode.MasterPeerCount = 227

8.17.2.44 const byte ParameterCode.MatchMakingType = 223

8.17.2.45 const byte ParameterCode.NickName = 202

8.17.2.46 const byte ParameterCode.PeerCount = 229

8.17.2.47 const byte ParameterCode.PlayerProperties = (byte)249

8.17.2.48 const byte ParameterCode.PlayerTTL = 235

8.17.2.49 const byte ParameterCode.PluginName = 201

8.17.2.50 const byte ParameterCode.Plugins = 204

8.17.2.51 const byte ParameterCode.PluginVersion = 200

8.17.2.52 const byte ParameterCode.Position = 223

8.17.2.53 const byte ParameterCode.Properties = (byte)251

8.17.2.54 const byte ParameterCode.PublishUserId = 239

8.17.2.55 const byte ParameterCode.ReceiverGroup = (byte)246

8.17.2.56 const byte ParameterCode.Region = (byte)210

8.17.2.57 const byte ParameterCode.Remove = 239

8.17.2.58 const byte ParameterCode.RoomName = (byte)255

8.17.2.59 const byte ParameterCode.Secret = 221

8.17.2.60 const byte ParameterCode.SuppressRoomEvents = 237

8.17.2.61 const byte ParameterCode.TargetActorNr = (byte)253

8.17.2.62 const byte ParameterCode.UriPath = 209

8.17.2.63 const byte ParameterCode.UserId = 225

8.17.2.64 const byte ParameterCode.WebRpcParameters = 208

8.17.2.65 const byte ParameterCode.WebRpcReturnCode = 207

8.17.2.66 const byte ParameterCode.WebRpcReturnMessage = 206

8.18 PhotonAnimatorView Class Reference | 类参考

这个类帮助你同步 Mecanim 动画，只需添加该组件到你的游戏对象上，并确保 PhotonAnimatorView 被添加到观察组件的列表里。

继承 MonoBehaviour 和 IPunObservable 接口。

Classes | 类

- class SynchronizedLayer
- class SynchronizedParameter

Public Types | 公共类型

- enum ParameterType { ParameterType.Float = 1, ParameterType.Int = 3,

ParameterType.Bool = 4,ParameterType.Trigger = 9 }

- enum SynchronizeType { SynchronizeType.Disabled = 0,

SynchronizeType.Discrete = 1, SynchronizeType.Continuous = 2 }

Public Member Functions | 公共成员函数

- void CacheDiscreteTriggers ()

缓存离散触发器的值来跟踪被触动的触发器，并将在同步进程完成后复位。

- bool DoesLayerSynchronizeTypeExist (int layerIndex)

检查一个特定的层是否被配置为同步。

- bool DoesParameterSynchronizeTypeExist (string name)

检查指定的参数是否被配置为同步。

- List< SynchronizedLayer > GetSynchronizedLayers ()

获取所有同步层的列表。

- List< SynchronizedParameter > GetSynchronizedParameters ()

获取所有同步参数的列表。

- SynchronizeType GetLayerSynchronizeType (int layerIndex)

获取该图层如何同步的类型。

- SynchronizeType GetParameterSynchronizeType (string name)

获取参数如何同步的类型。

- void SetLayerSynchronized (int layerIndex, SynchronizeType synchronizeType)

设置一个层该如何同步。

- void SetParameterSynchronized (string name, ParameterType type, SynchronizeType synchronizeType)

设置一个参数应该如何被同步。

- void OnPhotonSerializeView (PhotonStream stream, PhotonMessageInfo info)

Called by PUN several times per second, so that your script can write and read synchronization data for the PhotonView.

8.18.1 Detailed Description | 详细描述

这个类帮助你同步 Mecanim 动画，只需添加该组件到你的游戏对象上，并确保 PhotonAnimatorView 被添加到观察组件的列表里。

当使用触发参数，确保组件设置游戏对象上的触发器在组件堆栈中的比' PhotonAnimatorView' 更高。触发器只在一帧中被触发。

8.18.2 Member Enumeration Documentation | 成员枚举文档

8.18.2.1 enum PhotonAnimatorView.ParameterType

Enumerator | 枚举

Float | 浮点数

Int | 整型

Bool | 布尔值

Trigger | 触发器

8.18.2.2 enum PhotonAnimatorView.SynchronizeType

Enumerator | 枚举

Disabled | 禁用同步

Discrete | 离散同步

Continuous | 连续同步

8.18.3 Member Function Documentation | 成员函数文档

8.18.3.1 void PhotonAnimatorView.CacheDiscreteTriggers ( )

缓存离散触发器的值来跟踪被触动的触发器，并将在同步进程完成后复位。

8.18.3.2 bool PhotonAnimatorView.DoesLayerSynchronizeTypeExist (int layerIndex)

检查一个特定的层是否被配置为同步。

Parameters | 参数

layerIndex	图层的索引。

Returns | 返回值

如果该图层被同步返回 true。

8.18.3.3 bool PhotonAnimatorView.DoesParameterSynchronizeTypeExist (string name)

检查指定的参数是否被配置为同步。

Parameters | 参数

name	传入需要检查参数的名称。

Returns | 返回值

如果该参数被同步则返回 true。

8.18.3.4 SynchronizeType PhotonAnimatorView.GetLayerSynchronizeType (int layerIndex)

获取该图层如何同步的类型。

Parameters | 参数

layerIndex	图层的索引。

Returns | 返回值

Disabled/Discrete/Continuous | 禁用同步/离散同步/持续同步。

8.18.3.5 SynchronizeType PhotonAnimatorView.GetParameterSynchronizeType (string name)

获取参数如何同步的类型。

Parameters | 参数

name	要获取同步类型的参数名。

Returns | 返回值

Disabled/Discrete/Continuous | 禁用同步/离散同步/持续同步。

8.18.3.6 List<SynchronizedLayer> PhotonAnimatorView.GetSynchronizedLayers ( )

获取所有同步层的列表。

Returns | 返回值

SynchronizedLayer 对象的列表。

8.18.3.7 List<SynchronizedParameter> PhotonAnimatorView.GetSynchronizedParameters ( )

获取所有同步参数的列表。

Returns | 返回值

返回 SynchronizedLayer 对象的列表。

8.18.3.8 void PhotonAnimatorView.OnPhotonSerializeView (PhotonStream stream, PhotonMessageInfo info)

被 PUN 每秒调用几次， 这样你的脚本才可以为 PhotonView 读写同步数据。

该方法会在那些被指派作为一个 PhotonView 的观察组件的脚本里被调用。

PhotonNetwork.sendRateOnSerialize 会影响该方法被调用的频率。

PhotonNetwork.sendRate 会影响该客户端发包的频率。

实现这个方法，你就可以自定义哪些数据是 PhotonView 定期同步的。你的代码定义什么正在发送（内容）和数据是如何被接收到的客户端使用的。

不像其他回调函数，OnPhotonSerializeView 只在其被作为一个 PhotonView.observed 脚本指派到一个 PhotonView 时被调用。

要使用该方法，PhotonStream 是必须的。它会在控制一个 PhotonView (PhotonStream.isWriting == true)的客户端上 "写入模式"里和在收到控制客户端发送信息的远程客户端上的“读取模式”里。

如果你跳过写入任何值到二进制流里，PUN 会跳过更新。谨慎使用，这样会节省宽带和消息（每个房间每秒只能发送有限的数量）。

需要注意的是当发送者没有发送任何更新时 OnPhotonSerializeView 不会被远程客户端调用。这不能被用于作为“x 次每秒的 Update()”。实现了 IPunObservable 接口。

8.18.3.9 void PhotonAnimatorView.SetLayerSynchronized (int layerIndex, SynchronizeType synchronizeType)

设置一个层应该如何被同步。

Parameters | 参数

layerIndex	层索引。
synchronizeType	同步类型 Disabled/Discrete/Continuous。

8.18.3.10 void PhotonAnimatorView.SetParameterSynchronized (string name, ParameterType type, SynchronizeType synchronizeType)

设置一个参数应该如何被同步。

Parameters | 参数

name	需要被同步的参数的名称。
type	需要同步的参数的类型。
synchronizeType	同步类型 Disabled/Discrete/Continuous。

8.19 PhotonLagSimulationGui Class Reference

这个 MonoBehaviour 是一个 Photon 客户端的网络模拟功能的基本 GUI。它可以修改滞后（固定延迟），抖动（随机滞后）和数据包丢失。

继承了 MonoBehaviour。

Public Member Functions | 公共成员函数

- void Start ()
- void OnGUI ()

Public Attributes | 公共字段

- Rect WindowRect = new Rect(0, 100, 120, 100)

为窗口占据位置。

- int WindowId = 101

Unity 的 GUI 窗口 ID (必须是唯一的，否则会造成问题)。

- bool Visible = true

展示或隐藏 GUI (不会影响设置).

Properties | 属性

- PhotonPeer Peer [get, set]

当前使用的节点 (用来设置网络模拟).

8.19.1 Detailed Description | 详细描述

这个 MonoBehaviour 是一个 Photon 客户端的网络模拟功能的基本 GUI。它可以修改滞后（固定延迟），抖动（随机滞后）和数据包丢失。

8.19.2 Member Function Documentation | 成员函数文档

8.19.2.1 void PhotonLagSimulationGui.OnGUI ( )

8.19.2.2 void PhotonLagSimulationGui.Start ( )

8.19.3 Member Data Documentation | 成员数据文档

8.19.3.1 bool PhotonLagSimulationGui.Visible = true

展示或隐藏 GUI (不会影响设置).

8.19.3.2 int PhotonLagSimulationGui.WindowId = 101

Unity 的 GUI 窗口 ID (必须是唯一的，否则会造成问题)。

8.19.3.3 Rect PhotonLagSimulationGui.WindowRect = new Rect(0, 100, 120, 100) //为窗口占位。

8.19.4 Property Documentation | 属性文档

8.19.4.1 PhotonPeer PhotonLagSimulationGui.Peer [get], [set]

当前使用的节点 (用来设置网络模拟).

8.20 PhotonMessageInfo Struct Reference | 结构

关于一个特定消息、RPC 或更新的信息容器类。

Public Member Functions | 公共成员函数

- PhotonMessageInfo (PhotonPlayer player, int timestamp, PhotonView view)
- override string ToString ()

Public Attributes | 公共字段

- readonly PhotonPlayer sender
- readonly PhotonView photonView

Properties | 属性

- double timestamp [get]

8.20.1 Detailed Description | 详细描述

关于一个特定消息、RPC 或更新的信息容器类。

8.20.2 Constructor & Destructor Documentation | 构造&析构

8.20.2.1 PhotonMessageInfo.PhotonMessageInfo (PhotonPlayer player, int timestamp, PhotonView view)

8.20.3 Member Function Documentation | 成员函数文档

8.20.3.1 override string PhotonMessageInfo.ToString ( )

8.20.4 Member Data Documentation | 成员数据文档

8.20.4.1 readonly PhotonView PhotonMessageInfo.photonView

8.20.4.2 readonly PhotonPlayer PhotonMessageInfo.sender

8.20.5 Property Documentation | 属性文档

8.20.5.1 double PhotonMessageInfo.timestamp [get]

8.21 PhotonNetwork Class Reference | 类参考

使用 PhotonNetwork 插件的主类. 这是一个静态类。

Public Member Functions | 公共成员函数

- delegate void EventCallback (byte eventCode, object content, int senderId)

定义了 OnEventCall 里可用的代理。

Static Public Member Functions | 静态公共成员函数

- static void SwitchToProtocol (ConnectionProtocol cp)

离线模式时，网络协议可以被切换(这会影响你可以使用来连接的端口)。

- static bool ConnectUsingSettings (string gameVersion)

如在编辑器里配置的那样连接到 Photon (保存在 PhotonServerSettings 文件中)。

- static bool ConnectToMaster (string masterServerAddress, int port, string appID, string gameVersion)

通过地址、端口、appID(应用 ID 用于验证身份)和游戏(客户端)版本连接到一个 Photon 主服务器。

- static bool Reconnect ()

可以被用来在一个断连之后重新连接到主服务器。

- static bool ReconnectAndRejoin ()

当客户端在游戏时掉线，这个方法企图重连并重新加入房间。

- static bool ConnectToBestCloudServer (string gameVersion)

在最低 ping 的条件下连接到 Photon 云域 (在支持 Unity 的 Ping 平台上)。

- static bool ConnectToRegion (CloudRegionCode region, string gameVersion)

连接到选择的 Photon 云域。

- static void OverrideBestCloudServer (CloudRegionCode region)

重写用于 ConnectToBestCloudServer(string gameVersion)的域。

- static void RefreshCloudServerRating ()

再次 Pings 所有云服务器来找出最佳 ping 服务器（当前）。

- static void NetworkStatisticsReset ()

重置流量状态并重新启用它们。

- static string NetworkStatisticsToString ()

只有在 NetworkStatisticsEnabled 被用来收集数据时可用。

- static void InitializeSecurity ()

仅用于兼容 Unity 的网络。加密是在连接时自动初始化的。

- static void Disconnect ()

使这个客户端从 photon 服务器断连，离开任何房间的一个过程并在完成时调用 OnDisconnectedFromPhoton。

- static bool FindFriends (string[ ] friendsToFind)

为好友列表请求房间和在线数据并将结果保存在 PhotonNetwork.Friends 里面。

- static bool CreateRoom (string roomName)

用给出的名字创建一个房间 ,但是如果对应名称的房间已存在则会创建失败。会为传入 null 房间名的 roomName 参数创建随机房间名。

- static bool CreateRoom (string roomName, RoomOptions roomOptions, TypedLobby typedLobby)

创建一个房间，但是如果该房间已存在则会创建失败。只能在主服务器上被调用。

- static bool CreateRoom (string roomName, RoomOptions roomOptions, TypedLobby typedLobby, string[ ] expectedUsers)

创建一个房间，但是如果该房间已存在则会创建失败。只能在主服务器上被调用。

- static bool JoinRoom (string roomName)

通过房间名加入房间并在成功时调用 OnJoinedRoom()。该方法不会被游戏大厅影响。

- static bool JoinRoom (string roomName, string[ ] expectedUsers)

通过房间名加入房间并在成功时调用 OnJoinedRoom()。该方法不会被游戏大厅影响。

- static bool JoinOrCreateRoom (string roomName, RoomOptions roomOptions, TypedLobby typedLobby)

让你要么加入一个命名的房间或临时创建一个-你没必要知道某人是否已经创建了该房间。

- static bool JoinOrCreateRoom (string roomName, RoomOptions roomOptions, TypedLobby typedLobby,string[ ] expectedUsers)

让你要么加入一个命名的房间或临时创建一个-你没必要知道某人是否已经创建了该房间。

- static bool JoinRandomRoom ()

在当前使用的游戏大厅里加入任何可用的房间，并且如果没有可用的房间则会失败。

- static bool JoinRandomRoom (Hashtable expectedCustomRoomProperties, byte expectedMaxPlayers)

试图加入一个适合的、自定义属性的开放房间，但是如果当前没有满足条件的房间就会加入失败。

- static bool JoinRandomRoom (Hashtable expectedCustomRoomProperties, byte expectedMaxPlayers,MatchmakingMode matchingType, TypedLobby typedLobby, string sqlLobbyFilter, string[ ] expectedUsers=null)

试图加入一个适合的、自定义属性的开放房间，但是如果当前没有满足条件的房间就会加入失败。

- static bool ReJoinRoom (string roomName)

可以被用来在断连和重连之后返回到一个房间。

- static bool JoinLobby ()

在主服务器上这个会加入默认游戏大厅，该大厅会列出当前在使用的房间。

- static bool JoinLobby (TypedLobby typedLobby)

在一个主服务器上你可以加入一个游戏大厅来获取可用房间的列表。

- static bool LeaveLobby ()

离开一个游戏大厅从而停止可用房间的更新。

- static bool LeaveRoom ()

离开当前房间并返回到主服务器，在主服务器上你可以加入或创建房间。

- static RoomInfo[ ] GetRoomList ()

以 RoomInfo 数组的形式获取当前已知的房间。这个列表在游戏大厅中可用并实时更新。

- static void SetPlayerCustomProperties (Hashtable customProperties)

在本地设置玩家的属性，并与其他的玩家同步（不要直接修改这些属性）。

- static void RemovePlayerCustomProperties (string[ ] customPropertiesToDelete)

本地移除该玩家的自定义属性。重要：这不会同步其改变！当切换房间时有用。

- static bool RaiseEvent (byte eventCode, object eventContent, bool sendReliable, RaiseEventOptions options)

在房间里发送全部自定义事件。事件中包含至少一个事件代码 EventCode (0..199)，并且可以有内容。

- static int AllocateViewID ()

为当前/本地玩家分配一个有效的 viewID。

- static int AllocateSceneViewID ()

启用主客户端来为场景对象分配有效的 viewID。

- static void UnAllocateViewID (int viewID)

注销一个 viewID (手动实例化并摧毁的网络对象的视图身份)。

- static GameObject Instantiate (string prefabName, Vector3 position, Quaternion rotation, int group)

在网络上实例化一个预设。该预设需要在“Resources”文件夹的根目录下。

- static GameObject Instantiate (string prefabName, Vector3 position, Quaternion rotation, int group, object[ ] data)

在网络上实例化一个预设。该预设需要在“Resources”文件夹的根目录下。

- static GameObject InstantiateSceneObject (string prefabName, Vector3 position, Quaternion rotation, int group, object[ ] data)

在网络上实例化一个场景所有的预设。PhotonViews 将会被主客户端控制。该预设需要在“Resources”文件夹的根目录下。

- static int GetPing ()

目前往返 photon 服务器的用时。

- static void FetchServerTimestamp ()

刷新服务器时间戳（异步操作，花费一个来回的时间）。

- static void SendOutgoingCommands ()

可以被用来立即发送远程过程调用 RPCs 并实例化刚才调用的，这样它们就会在去往其他

玩家的路上。

- static bool CloseConnection (PhotonPlayer kickPlayer)

请求一个客户端断连（踢人）。只有主客户端才可以这么做。

- static bool SetMasterClient (PhotonPlayer masterClientPlayer)

要求服务器指派另一个玩家来作为当前房间的主客户端。

- static void Destroy (PhotonView targetView)

网络摧毁与 PhotonView 相关的游戏对象，除非这个 PhotonView 是静态的，或者不在该客户端控制下。

- static void Destroy (GameObject targetGo)

网络摧毁游戏对象，除非它是静态的，或者不在该客户端的控制下。

- static void DestroyPlayerObjects (PhotonPlayer targetPlayer)

网络摧毁目标玩家的所有的游戏对象、PhotonViews 和它们的 RPCs。只能被本地玩家或主客户端调用。

- static void DestroyPlayerObjects (int targetPlayerId)

网络摧毁对应 ID 玩家的所有的游戏对象、PhotonViews 和它们的 RPCs。只能被本地玩家或主客户端调用。

- static void DestroyAll ()

网络摧毁房间内所有的游戏对象、PhotonViews 和它们的 RPCs。移除任何来自服务器的缓存。只能被主客户端调用。

- static void RemoveRPCs (PhotonPlayer targetPlayer)

移除所有被 targetPlayer 目标玩家发送且来自服务器缓存的 RPCs。只能被本地玩家或主客户端调用。

- static void RemoveRPCs (PhotonView targetPhotonView)

移除所有通过 targetPhotonView 被发送且来自服务器缓存的 RPCs。主客户端和 targetPhotonView 的拥有者才可以调用。

- static void RemoveRPCsInGroup (int targetGroup)

移除所有在 targetGroup 目标组中被发送且来自服务器缓存的 RPCs，如果这是一个主客户端或它控制独立的 PhotonView。

- static void CacheSendMonoMessageTargets (Type type)

用当前存在的拥有类型组件的游戏对象填充 SendMonoMessageTargets。

- static HashSet< GameObject > FindGameObjectsWithComponent (Type type)

找到有特定类型组件的游戏对象（使用 FindObjectsOfType 方法）。

- static void SetReceivingEnabled (int group, bool enabled)

在给定的组中启用/禁用接收（应用到 PhotonViews）。

- static void SetReceivingEnabled (int[ ] enableGroups, int[ ] disableGroups)

在给定的几个组中启用/禁用接收（应用到 PhotonViews）。

- static void SetSendingEnabled (int group, bool enabled)

在给定的组中启用/禁用发送（应用到 PhotonViews）。

- static void SetSendingEnabled (int[ ] enableGroups, int[ ] disableGroups)

在给定的几个组中启用/禁用发送（应用到 PhotonViews）。

- static void SetLevelPrefix (short prefix)

为接下来实例化的 PhotonViews 设置等级前缀。如果你只需要一个则不要设置！

- static void LoadLevel (int levelNumber)

包装加载一个关卡来暂停网络消息队列。可选地同步房间里已加载的关卡。

- static void LoadLevel (string levelName)

包装加载一个关卡来暂停网络消息队列。可选地同步房间里已加载的关卡。

- static bool WebRpc (string name, object parameters)

该操作使 Photon 通过名称（路径）以及给出的参数来调用你的自定义网页服务。

Public Attributes | 公共字段

- const string versionPUN = "1.78"

PUN 的版本号。也被用在游戏版本里来相互区分客户端版本。

Static Public Attributes | 静态公共字段

- static readonly int MAX_VIEW_IDS = 1000

被指派的每个玩家(或场景)的 PhotonViews 的最大数量。详见 General Documentation 综合文档中的"Limitations"话题关于如何提升限制。

- static ServerSettings PhotonServerSettings = (ServerSettings)Resources.Load(PhotonNetwork.serverSettingsAssetFile,typeof(ServerSettings))

序列化的服务器设置，安装向导写入以在 ConnectUsingSettings 使用。

- static bool InstantiateInRoomOnly = true

如果 true，实例化方法将检查你是否在一个房间里，如果你没有在房间里就会实例化失败。

- static PhotonLogLevel logLevel = PhotonLogLevel.ErrorsOnly

网络日志等级。控制 PUN 日志的内容。

- static float precisionForVectorSynchronization = 0.000099f

一个二维向量或三维向量（例如一个 transform 的旋转 Rotation）在我们通过一个 PhotonView 的 OnSerialize/ObservingComponent 组件发送之前需要改变的最小精度。

- static float precisionForQuaternionSynchronization = 1.0f

一个旋转 Rotation 在我们通过一个 PhotonView 的 OnSerialize/ObservingComponent 组件发送之前需要改变的最小角度。

- static float precisionForFloatSynchronization = 0.01f

浮点数之间在我们通过一个 PhotonView 的 OnSerialize/ObservingComponent 组件发送之前的最小差异。

- static bool UseRpcMonoBehaviourCache

当启用时，我们调用 RPC 的 MonoBehaviours 被缓存，避免开销大的 GetComponents<MonoBehaviour>()调用。

- static bool UsePrefabCache = true

当启用（true）时，实例化使用 PhotonNetwork.PrefabCache 来保持游戏对象在内存中（提高相同的预制实例化）。

- static Dictionary< string,GameObject > PrefabCache = new Dictionary<string,GameObject>()

保持游戏对象的引用来频繁的实例化（内存不足而不是加载资源）。

- static HashSet< GameObject > SendMonoMessageTargets

如果不为 null，这是（独家）被 PUN 的 SendMonoMessage()调用的游戏对象列表。

- static Type SendMonoMessageTargetType = typeof(MonoBehaviour)

Defines which classes can contain PUN Callback implementations.定义哪些类可以包含 PUN 回调实现。

- static bool StartRpcsAsCoroutine = true

可以被用来跳过作为协程启动 RPC，它可以是一个性能问题。

- static int maxConnections

仅在 UNet 中使用。在 PUN 里，在 PhotonNetwork.CreateRoom 里设置玩家数量。

- static float BackgroundTimeout = 60.0f

定义在 Unity 的方法 OnApplicationPause(true)调用之后 PUN 保持连接多少秒。默认：60 秒。

- static EventCallback OnEventCall

通过在这里使用"+="来注册你的 RaiseEvent 触发事件处理方法。

Properties | 属性

- static string gameVersion [get, set]

你这个项目构建的版本字符串。可以被用来分离不兼容的客户端。连接时发送。

- static string ServerAddress [get]

目前使用的服务器地址（无论是主机或游戏服务器）。

- static bool connected [get]

false 直到你最初连接到 Photon。在脱机模式下为 true，当连接到任何服务器时为 true，甚至在切换服务器时也为 true。

- static bool connecting [get]

当你调用 ConnectUsingSettings（或类似的函数）时为 true，直到低级的 Photon 连接建立。

- static bool connectedAndReady [get]

只有当到服务器的连接已准备好接受连接、离开等操作时，才是正确的连接版本。

- static ConnectionState connectionState [get]

简化的连接状态。

- static ClientState connectionStateDetailed [get]

详细的连接状态（忽略 PUN，因此在切换服务器时可以“断开连接”）。

- static ServerConnection Server [get]

当前连接或连接到此客户端的服务器（类型）。

- static AuthenticationValues AuthValues [get, set]

连接期间使用的用户身份验证值。

- static Room room [get]

获取我们目前在的房间。如果我们不在任何房间里则为 null。

- static PhotonPlayer player [get]

本地 PhotonPlayer。总是可用并代表该玩家。CustomProperties 自定义属性可以在进入

房间之前被设置，也将被同步。

- static PhotonPlayer masterClient [get]

当前房间或空（房间外）的主客户端。

- static string playerName [get, set]

设置与你进入房间里的所有人同步玩家的昵称。这会设置 PhotonPlayer.name。

- static PhotonPlayer[ ] playerList [get]

在当前房间的玩家名单，包括本地玩家。

- static PhotonPlayer[ ] otherPlayers [get]

在当前房间的玩家名单，不包括本地玩家。

- static List< FriendInfo > Friends [get, set]

好友的只读列表，他们的在线状态和其所在的房间。Null 直到由 FindFriends 调用初始化。

- static int FriendsListAge [get]

朋友列表信息的年龄（以毫秒为单位）。它是 0，直到一个朋友列表被获取。

- static IPunPrefabPool PrefabPool [get, set]

对象池可以用来保存和重用实例化的对象实例。它取代了 Unity 的默认初始化和销毁方法。

- static bool offlineMode [get, set]

离线模式可以被设置来在单人游戏模式中重用你的多人游戏的代码。当该模式开启时 PhotonNetwork 不会创造任何连接，并且几乎没有开销。

重用 RPC 和 PhotonNetwork.Instantiate 时最有用。

- static bool automaticallySyncScene [get, set]

定义是否房间里所有客户端应该加载和主客户端一样的关卡（如果使用了 PhotonNetwork.LoadLevel）。

- static bool autoCleanUpPlayerObjects [get, set]

此设置定义了每个房间，当玩家离开时网络实例化的游戏对象（与 PhotonView）是否得到清理。

- static bool autoJoinLobby [get, set]

在 PhotonServerSettings 资源里设置。定义了当连接到主服务器时，PhotonNetwork 是否应该加入“游戏大厅”。

- static bool EnableLobbyStatistics [get, set]

在 PhotonServerSettings 资源里设置。启用从主服务器获取活跃游戏大厅列表。

- static List< TypedLobbyInfo > LobbyStatistics [get, set]

如果打开，主服务器将为该应用提供活跃游戏大厅的信息。

- static bool insideLobby [get]

当该客户端在一个游戏大厅里时为 true。

- static TypedLobby lobby [get, set]

当 PUN 加入一个游戏大厅或创建一个游戏时将使用的游戏大厅。

- static int sendRate [get, set]

定义 PhotonNetwork 发包的频率。如果你改变该字段，别忘了也要改变 sendRateOnSerialize。

- static int sendRateOnSerialize [get, set]

定义在 PhotonViews 上调用 OnPhotonSerialize 的频率。（x 次每秒）

- static bool isMessageQueueRunning [get, set]

可以用来暂停传入事件调度（RPCs 远程过程调用，实例化和其他传入的事件）。

- static int unreliableCommandsLimit [get, set]

每次调度时使用一次以限制每个通道的不可靠命令（所以暂停之后，许多通道仍然会导致很多不可靠的命令）。

- static double time [get]

Photon 网络时间，和服务器同步。

- static int ServerTimestamp [get]

当前服务器的毫秒时间戳。

- static bool isMasterClient [get]

是否是主客户端？

- static bool inRoom [get]

是否在一个房间内(connectionStateDetailed == ClientState.Joined)。

- static bool isNonMasterClientInRoom [get]

如果客户端在一个房间里且不是该房间的主客户端则为 true。

- static int countOfPlayersOnMaster [get]

当前寻找房间的玩家数量（在主服务器上每隔 5 秒刷新）。

- static int countOfPlayersInRooms [get]

目前在一些房间里使用你的应用的用户数量（由主服务器每 5 秒发送一次）。使用

playerList.Count 来获取你所在房间里的玩家数！

- static int countOfPlayers [get]

正在使用该应用的玩家计数（在主服务器上每隔 5 秒更新）。

- static int countOfRooms [get]

目前使用的房间数量（在主服务器上每隔 5 秒更新）。

- static bool NetworkStatisticsEnabled [get, set]

启用或禁用有关此客户端通信的统计信息的集合。

- static int ResentReliableCommands [get]

有重复的命令计数（由于本地重复时间是在收到 ACK 之前）。

- static bool CrcCheckEnabled [get, set]

CRC 检查用来发现和避免破包问题时很有用。可以在没有连接的时候启用。

- static int PacketLossByCrcCheck [get]

如果 CrcCheckEnabled，该字段会计算传入的没有一个有效的 CRC 校验并被拒绝的包。

- static int MaxResendsBeforeDisconnect [get, set]

定义一个可靠的消息可以被重新发送的次数，在没有得到 ACK 之前会触发一个断连。

默认: 5。

- static int QuickResends [get, set]

在网络丢失的情况下，可靠的消息可以迅速重复发送多达 3 次。

8.21.1 Detailed Description | 详细描述

使用 PhotonNetwork 插件的主类. 这是一个静态类。

8.21.2 Member Function Documentation | 成员函数文档

8.21.2.1 static int PhotonNetwork.AllocateSceneViewID ( ) [static]

启用主客户端来分配一个对场景对象有效的 viewID。

Returns | 返回值

返回一个可以用于一个新的 PhotonView 的 viewID 或-1（在发生错误的情况下）。

8.21.2.2 static int PhotonNetwork.AllocateViewID ( ) [static]

分配一个对当前/本地玩家有效的 viewID。

Returns | 返回值

返回一个可以被用于一个新的 PhotonView 的 viewID。

8.21.2.3 static void PhotonNetwork.CacheSendMonoMessageTargets (Type type) [static]

用当前存在的拥有类型组件的游戏对象填充 SendMonoMessageTargets。

Parameters | 参数

type	参数如果为 null，这将使用 SendMonoMessageTargets 作为组件类型（默认是 MonoBehaviour）。

8.21.2.4 static bool PhotonNetwork.CloseConnection (PhotonPlayer kickPlayer) [static]

请求一个客户端断连（俗称踢人）。只有主客户端可以踢人。

只有目标玩家能收到该事件。收到该事件的玩家将自动断连，其他玩家也会注意到其断连。

Parameters | 参数

kickPlayer	参数传入被踢/断连的 PhotonPlayer。

8.21.2.5 static bool PhotonNetwork.ConnectToBestCloudServer (string gameVersion) [static]

在最低 ping 的条件下连接到 Photon 云域（在支持 Unity 的 Ping 平台上)。

将把所有的云服务器 ping 的结果保存在 PlayerPrefs 里。调用该函数第一次可以采取+ - 2 秒。ping 的结果可以通过 PhotonNetwork.OverrideBestCloudServer(..)重写。如果你是第一次使用该调用需要高达 2 秒，所有的云服务器将被 ping 检查来寻找最佳区域。

PUN 设置向导在一个设置文件中存储你的 appID 并应用一个服务器地址/端口。要连接 Photon 云，设置文件中必须要有一个有效的 AppId(你可以在 Photon 云的仪表盘找到该 ID). https://www.photonengine.com/dashboard

连接到 Photon 云可能会失败的原因：

- 无效的 AppId (调用: OnFailedToConnectToPhoton()来检查准确的 AppId 值)
- 网络问题（调用: OnFailedToConnectToPhoton())
- 无效区域（调用: OnConnectionFail()和 DisconnectCause.InvalidRegion)
- 到达订阅 CCU 限制（调用: OnConnectionFail() 和 DisconnectCause.MaxCcuReached。也会调用: OnPhotonMaxCccuReached())

更多关于连接的限制: http://doc.exitgames.com/en/pun

Parameters | 参数

gameVersion	参数传入客户端的版本号。用户通过版本号来相互区分(这让你可以做出突破性的改变)。

Returns | 返回值

如果此客户端将基于 ping 连接到云服务器。即使是 true，这并不保证连接，但正在尝试中。

8.21.2.6 static bool PhotonNetwork.ConnectToMaster (string masterServerAddress, int port, string appID, string gameVersion) [static]

通过地址、端口、appID 和游戏（客户端）版本来连接 Photon 主服务器。

Parameters | 参数

masterServerAddress	参数传入服务器的地址(无论是你自己的或 Photon 云地址)
port	参数传入服务器连接到的端口。
appID	参数传入你的应用程序 ID (Photon 云为你的游戏提供给你一个 GUID ）。
gameVersion	参数传入客户端的版本号。用户通过版本号来相互区分(这让你可以做出突破性的改变)。

8.21.2.7 static bool PhotonNetwork.ConnectToRegion (CloudRegionCode region, string gameVersion) [static]

连接到选择的 Photon 云区域。

8.21.2.8 static bool PhotonNetwork.ConnectUsingSettings (string gameVersion) [static]

如在编辑器里配置的那样连接到 Photon(配置保存在 PhotonServerSettings 文件里)。

该方法将禁用离线模式（这不会摧毁任何实例化对象），将设置 isMessageQueueRunning 为 true。

您的服务器配置是由 PUN 设置向导创建的，并包含 Photon 云游戏所需域名的 AppID，以及服务器地址（如果你使用自己的主机托管 Photon）。这些设置通常不会经常改变。

要忽略配置文件并连接任何地方只需调用: PhotonNetwork.ConnectToMaster。

Parameters | 参数

gameVersion	参数传入客户端的版本号。用户通过版本号来相互区分(这让你可以做出突破性的改变)。

8.21.2.9 static bool PhotonNetwork.CreateRoom (string roomName) [static]

创建一个给定名称的房间，但如果这个房间（名称）已经存在。如果 roomName 参数为空则创建随机名称的房间。

如果你不想创建一个唯一的房间名，传入 null 或“”作为名称，那么服务器将会指派一个 roomName（一个作为字符串的 GUID）。

所创建的房间将自动放置在当前使用的游戏大厅（如果有的话）或默认大厅（如果你没有明确加入一个）。

仅在主服务器上调用这个。在内部，主服务器会回应一个服务器地址（和 roomName，如果需要的话）。都是在内部使用来切换到指定的游戏服务器和 roomName。

PhotonNetwork.autoCleanUpPlayerObjects 将变成该房间的 AutoCleanUp 属性，并且被所有加入该房间的客户端使用。

Parameters | 参数

roomName	参数传入将要创建的房间的唯一名称。传入 null 或“”空字符串来使服务器生成一个名称。

Returns | 返回值

如果操作得到排队，将被发送。

8.21.2.10 static bool PhotonNetwork.CreateRoom (string roomName, RoomOptions roomOptions, TypedLobby typedLobby) [static]

创建一个房间，但是如果这个房间已经存在就会创建失败。只能在主服务器上调用。

房间创建成功时，就会调用回调函数 OnCreatedRoom 和 OnJoinedRoom（后者，因为你在创建房间的同时作为第一个玩家加入该房间）。如果房间不能被创建（因为它已经存在了），则是 OnPhotonCreateRoomFailed 被调用。

如果你不想创建一个独特的房间名称，传入努力了或“”空字符串作为名称，服务器就会分配一个 roomName（即一个 GUID 字符串）。

房间可以在任何数量的大厅内创建。这些大厅不必在你创建一个房间之前存在（游戏大厅会根据需求自动创建）。大厅在服务器上分割房间列表时很有用。这可以帮助保持房间列表短和易于管理。如果你设定了一个 typedLobby 参数，房间则将在指定的大厅被创建（不管你是否在任何大厅内被激活）。如果你不设定 typedLobby，房间自动放置在当前激活的大厅（如果有的话）或默认的大厅。

仅在主服务器上调用这个。在内部，主服务器会回应一个服务器地址（和 roomName，如果需要的话）。都是在内部使用来切换到指定的游戏服务器和 roomName。

PhotonNetwork.autoCleanUpPlayerObjects 将变成该房间的 AutoCleanUp 属性，并且被所有加入该房间的客户端使用。

Parameters | 参数

roomName	参数传入将要创建的房间的唯一名称。传入 null 或 “” 空字符串来使服务器生成一个名称。
roomOptions	参数是一个像 MaxPlayers 一样的常用选项，初始化自定义房间属性。
typedLobby	参数如果是 null，该房间会自动创建在当前使用的游戏大厅（当你没有加入一个明确大厅时该大厅会是默认大厅）。

Returns | 返回值

如果操作得到排队，将被发送。

8.21.2.11 static bool PhotonNetwork.CreateRoom (string roomName,

RoomOptions roomOptions, TypedLobby typedLobby, string[ ] expectedUsers)

[static]

创建一个房间，但是如果这个房间已经存在就会创建失败。只能在主服务器上调用。

房间创建成功时，就会调用回调函数 OnCreatedRoom 和 OnJoinedRoom（后者，因为你在创建房间的同时作为第一个玩家加入该房间）。如果房间不能被创建（因为它已经存在了），则是 OnPhotonCreateRoomFailed 被调用。

如果你不想创建一个独特的房间名称，传入努力了或“”空字符串作为名称，服务器就会分配一个 roomName（即一个 GUID 字符串）。

房间可以在任何数量的大厅内创建。这些大厅不必在你创建一个房间之前存在（游戏大厅会根据需求自动创建）。大厅在服务器上分割房间列表时很有用。这可以帮助保持房间列表短和易于管理。如果你设定了一个 typedLobby 参数，房间则将在指定的大厅被创建（不管你是否在任何大厅内被激活）。如果你不设定 typedLobby，房间自动放置在当前激活的大厅（如果有的话）或默认的大厅。

仅在主服务器上调用这个。在内部，主服务器会回应一个服务器地址（和 roomName，如果需要的话）。都是在内部使用来切换到指定的游戏服务器和 roomName。

PhotonNetwork.autoCleanUpPlayerObjects 将变成该房间的 AutoCleanUp 属性，并且被所有加入该房间的客户端使用。

你可以定义一个 expectedUsers 的数组，来在房间里为这些玩家预留位置。Photon 中相应的功能称为“Slot Reservation”，并且可以在文档页中找到该功能。

Parameters | 参数

roomName	参数传入将要创建的房间的唯一名称。传入 null 或 “” 空字符串来使服务器生成一个名称。
roomOptions	参数是一个像 MaxPlayers 一样的常用选项，初始化自定义房间属性。
typedLobby	参数如果是 null，该房间会自动创建在当前使用的游戏大厅（当你没有加入一个明确大厅时该大厅会是默认大厅）。
expectedUsers	参数传入可选的玩家列表（由 UserId 组成），这个列表里的玩家都是你所期望能加入游戏并为其预留位置的玩家。

Returns | 返回值

如果操作得到排队，将被发送。

8.21.2.12 static void PhotonNetwork.Destroy (PhotonView targetView)

[static]

网络摧毁与 PhotonView 相关的游戏对象 ,除非 PhotonView 是一个静态的或不在该客户端的控制之下。

摧毁一个在房间里的网络游戏对象包括:

- 从服务器的房间缓存中移除实例化调用。
- 移除为 PhotonViews 缓存的 RPCs。
- 发送消息给其他客户端删除游戏对象（也会受网络延迟影响）。

通常，当你离开房间时，GOs 会自动销毁。如果你必须摧毁一个不在房间里的 GO，销毁

只在本地完成。

摧毁网络对象只有对那些由 PhotonNetwork.Instantiate()方法实例化的对象有效。和场景一起被加载的对象会被忽略，无论它们是否有 PhotonView 组件。

摧毁的网络对象必须在客户端的掌控之下:

- 由客户端实例化并拥有。
- 由离开该房间的玩家所实例化的对象是被主客户端所控制的。
- 场景游戏对象由主客户端控制。
- 当客户端不在房间时也可以摧毁游戏对象。

Returns | 返回值

没有返回值。如果有问题请检测错误调试日志。

8.21.2.13 static void PhotonNetwork.Destroy (GameObject targetGo) [static]

网络摧毁该游戏对象，除非它是静态的或不在该客户端的控制之下。

摧毁一个在房间里的网络游戏对象包括:

- 从服务器的房间缓存中移除实例化调用。
- 移除为 PhotonViews 缓存的 RPCs。
- 发送消息给其他客户端删除游戏对象（也会受网络延迟影响）。

通常，当你离开房间时，GOs 会自动销毁。如果你必须摧毁一个不在房间里的 GO，销毁只在本地完成。

摧毁网络对象只有对那些由 PhotonNetwork.Instantiate()方法实例化的对象有效。和场

景一起被加载的对象会被忽略，无论它们是否有 PhotonView 组件。

摧毁的网络对象必须在客户端的掌控之下:

- 由客户端实例化并拥有。
- 由离开该房间的玩家所实例化的对象是被主客户端所控制的。
- 场景游戏对象由主客户端控制。
- 当客户端不在房间时也可以摧毁游戏对象。

Returns | 返回值

没有返回值。如果有问题请检测错误调试日志。

8.21.2.14 static void PhotonNetwork.DestroyAll ( ) [static]

网络摧毁该房间里的所有游戏对象、PhotonViews 和它们的 RPCs。移除服务器上所有的缓存。只能被主客户端调用。

不像之前的摧毁方法，这个函数会移除服务器上该房间的所有缓存。如果你的游戏缓存了实例化对象和 RPC 调用以外的东西，那么这些缓存的东西一样会从服务器清除。

摧毁一个在房间里的网络游戏对象包括:

- 从服务器的房间缓存中移除实例化调用。
- 移除为 PhotonViews 缓存的 RPCs。
- 发送消息给其他客户端删除游戏对象（也会受网络延迟影响）。

摧毁网络对象只有对那些由 PhotonNetwork.Instantiate()方法实例化的对象有效。和场景一起被加载的对象会被忽略，无论它们是否有 PhotonView 组件。

Returns | 返回值

没有返回值。如果有问题请检测错误调试日志。

8.21.2.15 static void PhotonNetwork.DestroyPlayerObjects (PhotonPlayer targetPlayer) [static]

网络摧毁 targetPlayer 参数传入的目标玩家的所有游戏对象、PhotonViews 和它们的 RPCs。只能被本地玩家或主客户端调用。

摧毁一个在房间里的网络游戏对象包括:

- 从服务器的房间缓存中移除实例化调用。
- 移除为 PhotonViews 缓存的 RPCs。
- 发送消息给其他客户端删除游戏对象（也会受网络延迟影响）。

摧毁网络对象只有对那些由 PhotonNetwork.Instantiate()方法实例化的对象有效。和场景一起被加载的对象会被忽略，无论它们是否有 PhotonView 组件。

Returns | 返回值

没有返回值。如果有问题请检测错误调试日志。

8.21.2.16 static void PhotonNetwork.DestroyPlayerObjects (int targetPlayerId) [static]

网络摧毁 targetPlayerId 参数传入的对应 Id 的目标玩家的所有游戏对象、PhotonViews 和它们的 RPCs。只能被本地玩家或主客户端调用。

摧毁一个在房间里的网络游戏对象包括:

- 从服务器的房间缓存中移除实例化调用。
- 移除为 PhotonViews 缓存的 RPCs。
- 发送消息给其他客户端删除游戏对象（也会受网络延迟影响）。

摧毁网络对象只有对那些由 PhotonNetwork.Instantiate()方法实例化的对象有效。和场景一起被加载的对象会被忽略，无论它们是否有 PhotonView 组件。

Returns | 返回值

没有返回值。如果有问题请检测错误调试日志。

8.21.2.17 static void PhotonNetwork.Disconnect ( ) [static]

使该客户端从 photon 服务器断连，离开任何房间并在完成时调用 OnDisconnectedFromPhoton 的一个进程。

当你断连时，客户端将会发送一个“disconnecting”消息到服务器。这加快了发送给和你一样在同一个房间的玩家的离开/断开消息（否则该客户端连接服务器将超时）。在 offlineMode 里使用时，状态变化和事件调用 OnDisconnectedFromPhoton 会立即执行。离线模式也会被设置为 false。一旦断开，客户端可以再次连接。使用 ConnectUsingSettings。

8.21.2.18 delegate void PhotonNetwork.EventCallback (byte eventCode, object content, int senderId)

定义了在 OnEventCall 里使用的代理。

任何 eventCode < 200 将会被转发到你的代理。

Parameters | 参数

eventCode	参数传入被指派到收入事件的代码。
content	参数传入发送者要放进事件里的内容。
senderId	参数传入发送事件的玩家的 ID。如果是“房间”发送了该事件，它可能是 0。

8.21.2.19 static void PhotonNetwork.FetchServerTimestamp ( ) [static]

刷新服务器的时间戳（异步操作，需要往返）。

如果有一个坏连接使时间戳不可用或不精确的时候很有用。

8.21.2.20 static bool PhotonNetwork.FindFriends (string[ ] friendsToFind) [static]

为好友列表请求房间和在线状态，并在 PhotonNetwork.Friends 里保存结果。

仅在主服务器上有效，用于查找由一个选定用户列表所游戏的房间。

当可用时结果会被保存在 PhotonNetwork.Friends 里。该列表在第一次使用 OpFindFriends 时被初始化（在这之前是 null）。要刷新该列表，再一次调用 FindFriends 即可（在 5 秒、10 秒或 20 秒内）。

用户通过在 PhotonNetwork.AuthValues 里设置一个唯一的 userId 来识别自己的。查看 AuthenticationValues 的描述中关于如何设置和使用 PhotonNetwork.AuthValues 的信息。

好友列表必须从其他来源获取（不是由 Photon 提供）。

内部: 服务器回应包括两个信息数组（每一个索引匹配来自请求的一个好友）：ParameterCode.FindFriendsResponseOnlineList = bool[] 在线状态数组，

ParameterCode.FindFriendsResponseRoomIdList = string[] 房间名称数组（没有在房间里则为空字符串）。

Parameters | 参数

friendsToFind	参数传入好友名称数组 Array of friend (make sure to use unique playerName or AuthValues).（确保使用唯一的 playerName 或 AuthValues）。

Returns | 返回值

如果该操作可以被发送（必须连接，在任何时候都只允许一个请求）。在离线模式下总是返回 false。

8.21.2.21 static HashSet<GameObject> PhotonNetwork.FindGameObjectsWithComponent (Type type) [static]

找到有特定类型的组件的游戏对象（使用 FindObjectsOfType）。

Parameters | 参数

type	参数传入的类型必须是一个组件。

Returns | 返回值

返回有一个相应类型组件的游戏对象 HashSet。

8.21.2.22 static int PhotonNetwork.GetPing ( ) [static]

当前到 photon 服务器的往返时间。

Returns | 返回值

返回往返服务器的时间。

8.21.2.23 static RoomInfo [ ] PhotonNetwork.GetRoomList ( ) [static]

获取当前已知的房间作为 RoomInfo 数组。当在游戏大厅里时该数组是可用的，并会实时更新（详见 insideLobby）。

此列表是一个内部房间列表的缓存副本，这样它就可以在需要的时每一帧都能被访问。在你允许玩家加入房间之前，在每一个 RoomInfo 中你都可以通过比较 playerCount 和 MaxPlayers 来检查该房间是否已满。

房间的名称必须被用来加入它（通过 JoinRoom）。

封闭的房间也会在游戏大厅被列出，但它们不能被加入。而在一个房间里，任何玩家都可以设置 Room.visible 和 Room.open 来从匹配中隐藏房间和关闭房间。

Returns | 返回值

返回当前在游戏大厅里的房间的 RoomInfo[]数组。

8.21.2.24 static void PhotonNetwork.InitializeSecurity ( ) [static]

仅用于与 Unity 网络兼容。加密是在连接时自动初始化。

8.21.2.25 static GameObject PhotonNetwork.Instantiate (string prefabName, Vector3 position, Quaternion rotation, int group) [static]

在网络上实例化一个预设。这种预制需要位于“Resources”文件夹的根部。

如果不使用 Resources 文件夹下的预制体，您可以手动实例化 Instantiate 并分配 PhotonViews。

Parameters | 参数

参数名	参数定义
prefabName	要实例化的预设名称。
position	应用到实例上的 Vector3 位置。
rotation	应用到实例上的旋转 Quaternion。
group	应用于该实例的 PhotonView 分组。

Returns | 返回值

返回一个初始化了 PhotonView 的游戏对象的新实例。

8.21.2.26 static GameObject PhotonNetwork.Instantiate (string prefabName, Vector3 position, Quaternion rotation, int group,object[ ] data) [static]

在网络上实例化一个预设。这种预制需要位于“Resources”文件夹的根部。

如果不使用 Resources 文件夹下的预制体，您可以手动实例化 Instantiate 并分配 PhotonViews。

Parameters | 参数

参数名	参数定义
prefabName	要实例化的预设名称。
position	应用到实例上的 Vector3 位置。

rotation	应用到实例上的旋转 Quaternion。
group	应用于该实例的 PhotonView 分组。
data	可选的实例化数据。该数据将会被保存在它的 PhotonView.instantiationData 里面。

Returns | 返回值

返回一个初始化了 PhotonView 的游戏对象的新实例。

8.21.2.27 static GameObject PhotonNetwork.InstantiateSceneObject (string prefabName, Vector3 position, Quaternion rotation, int group, object[ ] data)

[static]

在网络上实例化一个场景拥有的预设。对应的 PhotonViews 将会被主客户端掌控。该预制件需要位于“Resources”文件夹的根部。

只有主客户端可以实例化场景对象。如果不使用 Resources 文件夹下的预制体，您可以手动实例化 Instantiate 并分配 PhotonViews。

Parameters | 参数

参数名	参数定义
prefabName	要实例化的预设名称。
position	应用到实例上的 Vector3 位置。
rotation	应用到实例上的旋转 Quaternion。
group	应用于该实例的 PhotonView 分组。

data	可选的实例化数据。该数据将会被保存在它的 PhotonView.instantiationData 里面。

Returns | 返回值

返回一个初始化了 PhotonView 的游戏对象的新实例。

8.21.2.28 static bool PhotonNetwork.JoinLobby ( ) [static]

在主服务器上该方法会加入默认的游戏大厅，默认大厅会列出当前在使用的房间列表。该房间列表通过服务器发送和刷新。你可以通过 PhotonNetwork.GetRoomList()函数获取房间列表缓存。

每个房间你都应该在加入之前检查该房间是否已满。Photon 也会列出已满的房间，除非你关闭并隐藏它们(room.open = false and room.visible = false)。

在最好的情况下，您使您的客户端加入随机游戏，如这里所述:

http://doc.exitgames.com/en/realtime/current/reference/matchmaking-and-lobby

您可以在不加入游戏大厅的情况下显示您当前的玩家和房间计数（但您必须在主服务器上）。

使用: countOfPlayers, countOfPlayersOnMaster, countOfPlayersInRooms 和 countOfRooms.

你可以使用不止一个游戏大厅来使房间列表更短。详见 JoinLobby(TypedLobby lobby) 。当创建新的房间时，这些房间会依附在当前使用的大厅或默认大厅。

无需在大厅中你就可以使用 JoinRandomRoom！在你连接前设置 autoJoinLobby =

false，就可以不加入游戏大厅。在这种情况下，连接的工作流将会在完成后调用 OnConnectedToMaster（如果你实现了它）。

8.21.2.29 static bool PhotonNetwork.JoinLobby (TypedLobby typedLobby)

[static]

在主服务器上你可以加入游戏大厅来获取可用房间的列表。

该房间列表通过服务器发送和刷新。你可以通过 PhotonNetwork.GetRoomList()函数获取房间列表缓存。

任何客户端都可以在运行时“组成”任何大厅。将所有房间都拆分成多个大厅将使每一个列表都更短。

然而，列表太多会使玩家匹配的体验很差。

在最好的情况下，你创建一个有限数量的游戏大厅。例如，每个游戏模式创建一个大厅：“koth”代表山丘之王和“ffa”代表全部免费，等等。

目前来没有大厅列表。

Sql 类型的游戏大厅为随机匹配提供一个不同的过滤模型。这也许更适合基于技术的游戏。

然而，你也将需要按照惯例在 sql 大厅中命名刷选的属性。两者都都在下面链接的匹配文档里面解释了。

在最好的情况下，您使您的客户加入随机游戏，如这里所述:

http://confluence.exitgames.com/display/PTN/Op+JoinRandomGame

每个房间你都应该在加入之前检查该房间是否已满。Photon 也会列出已满的房间，除非

你关闭并隐藏它们(room.open = false and room.visible = false)。

您可以在不加入游戏大厅的情况下显示您当前的玩家和房间计数（但您必须在主服务器上）。

使用: countOfPlayers, countOfPlayersOnMaster, countOfPlayersInRooms 和 countOfRooms.

当创建新的房间时，这些房间会依附在当前使用的大厅或默认大厅。

无需在大厅中你就可以使用 JoinRandomRoom！在你连接前设置 autoJoinLobby = false，就可以不加入游戏大厅。在这种情况下，连接的工作流将会在完成后调用 OnConnectedToMaster（如果你实现了它）。

Parameters | 参数

参数名	参数定义
typedLobby	将要加入大厅的类型（必须要有名称和类型）。

8.21.2.30 static bool PhotonNetwork.JoinOrCreateRoom (string roomName, RoomOptions roomOptions, TypedLobby typedLobby) [static]

让你要么加入一个命名的房间，要么在运行时创建该房间。你不需要知道是否有人已经创建了该房间。

这使得分组的玩家进入同一个房间更简单。一旦该组交换了 roomName，任何玩家可以调用 JoinOrCreateRoom，并且谁实际上加入或创建了该房间已经不重要。

参数 roomOptions 和 typedLobby 只在当房间实际上被该客户端创建时才被使用。

如果你得到一个回调函数 OnCreatedRoom（在之前 OnJoinedRoom 也被调用了），则房间是该客户端创建的。

Parameters | 参数

参数名	参数定义
roomName	要加入的房间名称。必须是非 null。
roomOptions	房间选项，以防它还没有存在。如果已经存在了，这些值会被忽略。
typedLobby	你想要新创建的房间被列入的大厅。如果该房间已经存在并已加入，则忽略。

Returns | 返回值

如果操作加入了队列则将被发送。

8.21.2.31 static bool PhotonNetwork.JoinOrCreateRoom (string roomName, RoomOptions roomOptions, TypedLobby typedLobby, string[ ] expectedUsers)

[static]

让你要么加入一个命名的房间，要么在运行时创建该房间。你不需要知道是否有人已经创建了该房间。

这使得分组的玩家进入同一个房间更简单。一旦该组交换了 roomName，任何玩家可以调用 JoinOrCreateRoom，并且谁实际上加入或创建了该房间已经不重要。

参数 roomOptions 和 typedLobby 只在当房间实际上被该客户端创建时才被使用。

如果你得到一个回调函数 OnCreatedRoom（在之前 OnJoinedRoom 也被调用了），则

房间是该客户端创建的。

你可以定义一个 expectedUsers 的数组，来在房间里为这些玩家预留位置。Photon 中相应的功能称为“Slot Reservation”，并且可以在文档页中找到该功能。

Parameters | 参数

参数名	参数定义
roomName	要加入的房间名称。必须是非 null。
roomOptions	房间选项，以防它还没有存在。如果已经存在了，这些值会被忽略。
typedLobby	你想要新创建的房间被列入的大厅。如果该房间已经存在并已加入，则忽略。
expectedUsers	参数传入可选的玩家列表（由 UserId 组成），这个列表里的玩家都是你所期望能加入游戏并为其预留位置的玩家。

Returns | 返回值

如果操作加入了队列则将被发送。

8.21.2.32 static bool PhotonNetwork.JoinRandomRoom ( ) [static]

在当前使用的游戏大厅里加入任何可用的房间，并且如果没有可用的房间则会失败。房间可以在任意大厅里根据需求创建。你可以从任何大厅加入房间，无需实际地加入该大厅。使用 JoinRandomRoom 重载 TypedLobby 参数。

该方法将只匹配依附在同一个大厅的房间！如果你使用许多大厅，你可能不得不重复 JoinRandomRoom，来寻找一些合适的房间。该方法查找当前激活大厅或默认大厅（如果没

有大厅加入）里的房间。

如果失败，你仍然可以创建一个房间（并为接下来 JoinRandomRoom 的使用者提供房间）。或者，稍后再试一次。

8.21.2.33 static bool PhotonNetwork.JoinRandomRoom (Hashtable expectedCustomRoomProperties, byte expectedMaxPlayers) [static]

试图加入一个匹配自定义属性的开放房间，但是如果当前没有可用的房间就会加入失败。

房间可以在任意大厅里根据需求创建。你可以从任何大厅加入房间，无需实际地加入该大厅。使用 JoinRandomRoom 重载 TypedLobby 参数。

该方法将只匹配依附在同一个大厅的房间！如果你使用许多大厅，你可能不得不重复 JoinRandomRoom，来寻找一些合适的房间。该方法查找当前激活大厅或默认大厅（如果没有大厅加入）里的房间.

如果失败，你仍然可以创建一个房间（并为接下来 JoinRandomRoom 的使用者提供房间）。或者，稍后再试一次。

Parameters | 参数

expectedCustomRoomProperties	符合这些自定义属性（字符串键和值）的房间过滤器。如果要忽略则传递 null。
expectedMaxPlayers	一个特定的 maxplayer 设置的过滤器。使用 0 来接受任何 maxPlayer 值。

Returns | 返回值

如果该操作在队列中则将被发送。

8.21.2.34 static bool PhotonNetwork.JoinRandomRoom (Hashtable expectedCustomRoomProperties, byte expectedMaxPlayers, MatchmakingMode matchingType, TypedLobby typedLobby, string sqlLobbyFilter,string[ ] expectedUsers = null) [static]

试图加入一个匹配自定义属性的开放房间，但是如果当前没有可用的房间就会加入失败。

房间可以在任意大厅里根据需求创建。你可以从任何大厅加入房间，无需实际地加入该大厅。使用 JoinRandomRoom 重载 TypedLobby 参数。

该方法将只匹配依附在同一个大厅的房间！如果你使用许多大厅，你可能不得不重复 JoinRandomRoom，来寻找一些合适的房间。该方法查找当前激活大厅或默认大厅（如果没有大厅加入）里的房间.

如果失败 ,你仍然可以创建一个房间(并为接下来 JoinRandomRoom 的使用者提供房间)。或者，稍后再试一次。

在离线模式里，房间将被创建，但是没有属性将被设置，且 JoinRandomRoom 调用的所有参数将被忽略。事件/回调函数 OnJoinedRoom 被调用（查看枚举 enum PhotonNetworkingMessage）

你可以定义一个 expectedUsers 的数组，来在房间里为这些玩家预留位置。Photon 中相应的功能称为“Slot Reservation”，并且可以在文档页中找到该功能。

Parameters | 参数

expectedCustomRoomProperties	符合这些自定义属性（字符串键和值）的房间过滤器。如果要忽略则传递 null。
expectedMaxPlayers	一个特定的 maxplayer 设置的过滤器。使用 0 来接受任何 maxPlayer 值。
matchingType	选择一个可用的匹配算法。详见 MatchmakingMode 枚举选项。
typedLobby	参数传入你想要寻找房间的游戏大厅。传入 null 来使用默认大厅。这样不会加入该大厅，也不会设置大厅属性。
sqlLobbyFilter	针对 SQL 类型的大厅的字符串过滤器。
expectedUsers	参数传入可选的玩家列表（由 UserId 组成），这个列表里的玩家都是你所期望能加入游戏并为其预留位置的玩家。

Returns | 返回值

如果该操作在队列中则将被发送。

8.21.2.35 static bool PhotonNetwork.JoinRoom (string roomName) [static]

该方法通过房间名称来加入房间，并在执行成功的时候调用 OnJoinedRoom()。该函数不受游戏大厅的影响。

在函数执行成功的时候，OnJoinedRoom()方法可以在任意脚本中被调用。你可以实现该

方法来回应加入房间操作。

如果房间要么满了、要么不再可用（房间可能在你试图加入的时候变成空房间），则 JoinRoom 失败。

实现 OnPhotonJoinRoomFailed()来在发生错误时获取一个回调函数。

要从游戏大厅的列表中加入一个房间，在这里使用 RoomInfo.name 作为 roomName。尽管使用多大厅，一个 roomName 对于你的应用来说总是"global" ，这样你就不需要指定该房间在哪个游戏大厅了。主服务器将会找到指定的房间。在 Photon 云端中，一个应用通过 AppId、游戏版本和 PUN 版本被定义。

PhotonNetworkingMessage.OnPhotonJoinRoomFailed

PhotonNetworkingMessage.OnJoinedRoom

Parameters | 参数

roomName	参数要加入房间的唯一名称。

Returns | 返回值

如果操作在队列中则将被发送。

8.21.2.36 static bool PhotonNetwork.JoinRoom (string roomName, string[ ] expectedUsers) [static]

该方法通过房间名称来加入房间，并在执行成功的时候调用 OnJoinedRoom()。该函数不受游戏大厅的影响。

在函数执行成功的时候，OnJoinedRoom()方法可以在任意脚本中被调用。你可以实现该方法来回应加入房间操作。

如果房间要么满了、要么不再可用（房间可能在你试图加入的时候变成空房间），则 JoinRoom 失败。

实现 OnPhotonJoinRoomFailed()来在发生错误时获取一个回调函数。

要从游戏大厅的列表中加入一个房间，在这里使用 RoomInfo.name 作为 roomName。尽管使用多大厅，一个 roomName 对于你的应用来说总是"global" ，这样你就不需要指定该房间在哪个游戏大厅了。主服务器将会找到指定的房间。在 Photon 云端中，一个应用通过 AppId、游戏版本和 PUN 版本被定义。

PhotonNetworkingMessage.OnPhotonJoinRoomFailed

PhotonNetworkingMessage.OnJoinedRoom

Parameters | 参数

roomName	参数要加入房间的唯一名称。
expectedUsers	参数传入可选的玩家列表（由 UserId 组成），这个列表里的玩家都是你所期望能加入游戏并为其预留位置的玩家。

Returns | 返回值

如果操作在队列中则将被发送。

8.21.2.37 static bool PhotonNetwork.LeaveLobby ( ) [static]

该方法离开游戏大厅来停止获取关于可用房间的更新。

这不会重置 PhotonNetwork.lobby！而会允许你稍后容易地加入该特定的游戏大厅。

即使不在游戏大厅中，也可以收到 countOfPlayers、 countOfPlayersOnMaster、 countOfPlayersInRooms 和 countOfRooms 的值。

不在游戏大厅里你也可以使用 JoinRandomRoom。当你连接时使用 autoJoinLobby 来不加入大厅。

8.21.2.38 static bool PhotonNetwork.LeaveRoom ( ) [static]

离开当前的房间并返回主服务器，在主服务器你可以加入或创建房间。

该方法会清理所有(网络)带 PhotonView 的游戏对象 ,除非你改变 autoCleanUp 成 false。返回到主服务器。

在离线模式里，本地的"fake"房间会被清理并立即调用 OnLeftRoom。

8.21.2.39 static void PhotonNetwork.LoadLevel (int levelNumber) [static]

包装加载关卡来暂停网络消息队列。可选地在房间里同步已加载的关卡。

要在房间里同步已加载的关卡 ,把 PhotonNetwork.automaticallySyncScene 设置成 true 即可。然后房间的主客户端将同步已加载的关卡到房间里的每一个玩家。

在加载关卡的时候，不分发其他玩家收到的信息是有道理的。该方法通过设置 PhotonNetwork.isMessageQueueRunning = false 来处理消息，并在关卡被加载后启用消息队列。

你应该确保在你加载另一个场景（该场景中不包含与上一个场景相同的游戏对象和 PhotonViews）前不发射 RPCs。你可以在 OnJoinedRoom 里面调用该函数。

该函数使用了 Application.LoadLevel。

Parameters | 参数

levelNumber	要加载的关卡数。当使用关卡数时 ,确保它们在所有客户端上都一样。

8.21.2.40 static void PhotonNetwork.LoadLevel (string levelName) [static]

包装加载关卡来暂停网络消息队列。可选地在房间里同步已加载的关卡。

在加载关卡的时候，不分发其他玩家收到的信息是有道理的。该方法通过设置 PhotonNetwork.isMessageQueueRunning = false 来处理消息，并在关卡被加载后启用消息队列。

要在房间里同步已加载的关卡 ,把 PhotonNetwork.automaticallySyncScene 设置成 true 即可。然后房间的主客户端将同步已加载的关卡到房间里的每一个玩家。

你应该确保在你加载另一个场景（该场景中不包含与上一个场景相同的游戏对象和 PhotonViews）前不发射 RPCs。你可以在 OnJoinedRoom 里面调用该函数。

该函数使用了 Application.LoadLevel。

Parameters | 参数

levelName	要加载的关卡的名称。确保它在同一个房间里的所有客户端上都可用。

8.21.2.41 static void PhotonNetwork.NetworkStatisticsReset ( ) [static]

重置流量统计并重启它们。

8.21.2.42 static string PhotonNetwork.NetworkStatisticsToString ( ) [static]

只有当 NetworkStatisticsEnabled 被用来收集一些数据时可用。

Returns | 返回值

一个具有重要网络统计的字符串。

8.21.2.43 static void PhotonNetwork.OverrideBestCloudServer

(CloudRegionCode region) [static]

重写用于 ConnectToBestCloudServer(string gameVersion)的域。

该方法将重写 ping 所有云服务器的结果。

使用该方法来允许你的用户在玩家偏好中保存一个手动的选择区域。

注意：你也可以使用 PhotonNetwork.ConnectToRegion 来（暂时地）连接到一个特定的区域。

8.21.2.44 static bool PhotonNetwork.RaiseEvent (byte eventCode, object eventContent, bool sendReliable,RaiseEventOptions options) [static]

在房间内发送完全可定制的事件。事件至少包含一个 EventCode（0..199）并可以有内容。

要收到某人发送的事件，在 PhotonNetwork.OnEventCall 里注册你的处理方法。

用例:

```
private void OnEventHandler(byte eventCode, object content, int senderId)
{
    Debug.Log("OnEventHandler");
}
PhotonNetwork.OnEventCall += this.OnEventHandler;
```

有了 senderId，你就可以寻找发送该事件的 PhotonPlayer 了。它是为内容和动作的各种不同类型分配一个 eventCode 的最佳实践。你必须映射该内容。

eventContent 是可选参数。要使其可以发送一些对象，则必须是一个"serializable type"，该类型基本上说是可以被客户端转化成 byte[]字节数组的对象。大多数基本类型和数组都被支持，包括 Unity 的 Vector2,Vector3, Quaternion。某些项目定义的 Transforms 或类不被支持！你可以使你自己的类按照下面这个案例 CustomTypes.cs 来转化成"serializable type"。

RaiseEventOptions 有一些（不太直观的）组合规则：如果你设置了 targetActors（PhotonPlayer.ID 值的数组），则 receivers 参数被忽略。当使用事件缓存时，targetActors、receivers 和 interestGroup 不能被使用。缓存的事件会发到所有终端。当使用 cachingOption removeFromRoomCache 时，eventCode 和 content 参数实际上没有发送，但是被用来作为过滤器。

Parameters | 参数

eventCode	一个确定事件类型的字节。您可能想每个动作都使用一个代码或发出预期内容的信号。允许：0 .. 199。
eventContent	一些序列化的对象，如字符串、字节、整型、浮点数（等）和这些对象类型的数组。字节键的哈希表能很好的发送变量内容。
sendReliable	确保这一事件达到所有玩家。事件得到确认，这需要带宽，并且它不能被跳过（在丢失的情况下可能会增加延迟）。
options	允许更复杂的事件使用。如果为 null，RaiseEventOptions.Default 将被使用（这很好）。

Returns | 返回值

如果事件不能被发送则返回 false。

8.21.2.45 static bool PhotonNetwork.Reconnect ( ) [static]

断开后，可用于重新连接到主服务器.。

失去连接后，您可以使用此方法将客户端再次连接到区域主服务器.。缓存你所在房间的roomname，并使用 ReJoin(roomname)方法来重新回到游戏。常见用例：在 iOS 设备上按住 Lock 按钮会立即断开连接。

8.21.2.46 static bool PhotonNetwork.ReconnectAndRejoin ( ) [static]

当客户端在游戏过程中丢失连接，该方法尝试重新回到房间。

此方法直接重新连接到托管 PUN 之前所在房间的游戏服务器。如果房间在这个时候被关闭，PUN 会调用 OnPhotonJoinRoomFailed 并返回这个客户端到主服务器。

检查返回值，该客户端是否会尝试重新连接并重新加入（如果符合条件）。如果 ReconnectAndRejoin 返回 false，你仍然可以尝试 Reconnect 和 ReJoin。

和 PhotonNetwork.ReJoin 类似，该方法必须要每个玩家使用唯一的 IDs（即 UserID）。

Returns | 返回值

如果没有已知的房间或用来返回的游戏服务器，则是 false。然后，该客户端不再尝试 ReconnectAndRejoin。

8.21.2.47 static void PhotonNetwork.RefreshCloudServerRating ( ) [static]

再一次 ping 所有的云服务器来寻找最佳 ping 的云服务器（当前的）。

8.21.2.48 static bool PhotonNetwork.ReJoinRoom (string roomName) [static]

可用于在断开和重新连接后返回房间。

在失去连接后，如果客户端重新连接的速度足够快，你也许可以回到房间继续玩。使用前面讲的 Reconnect() 和这个方法。缓存你所在的房间名，并使用 ReJoin(roomname)方法来回到游戏。

注: 为了能够使用 ReJoin 加入任何房间，你需要使用 UserIDs! 你也需要设置 RoomOptions.PlayerTtl。

重要: Instantiate()和使用 RPCs 还没有被支持。对 PhotonViews 所有权规则防止无缝回到游戏。使用自定义属性和带事件缓存的 RaiseEvent 取而代之。

常见使用案例：在 iOS 设备上按下锁定按钮会立即断开。

8.21.2.49 static void PhotonNetwork.RemovePlayerCustomProperties (string[ ] customPropertiesToDelete) [static]

本地删除“此”玩家的自定义属性。重要：这不同步更改！当你切换房间的时候很有用。

谨慎使用此方法。它可以使玩家之间的状态不一致！这只是本地改变 player.customProperties。该方法用来清除您的游戏间的自定义属性很有用（假设这些自定义属性储存了你创造的回合/关卡，击杀数，等等）。

在一个房间里时 SetPlayerCustomProperties()同步并可以被用来把自定义属性值设置为 null。这可以被认为是“删除”，当在一个房间里时。

如果 customPropertiesToDelete 为 null 或有 0 个条目 那么所有自定义属性被删除了(取

而代之的是一个新的哈希表）。如果指定键来删除，那些指定的就会从 Hashtable 哈希表中删除，但其他键是不受影响的。

Parameters | 参数

customPropertiesToDelete	要移除的自定义属性键列表。见备注。

8.21.2.50 static void PhotonNetwork.RemoveRPCs (PhotonPlayer targetPlayer) [static]

删除所有由 targetPlayer 发送到服务器上缓冲的 RPCs。只能在本地玩家客户端上被调用（对“自己”）或主客户端（对任何人）。

此方法必须下列条件中的一个：

- 这就是 targetPlayer 的客户端。
- 该客户端就是主客户端（可以移除任何 PhotonPlayer 的 RPCs）。

如果 targetPlayer 调用 RPCs 的同时这个方法也被调用，网络延迟将决定那些是否得到缓冲或像其余的一样被清除了。

Parameters | 参数

targetPlayer	该目标玩家的缓存 RPCs 将从服务器缓冲中被移除。

8.21.2.51 static void PhotonNetwork.RemoveRPCs (PhotonView targetPhotonView) [static]

移除所有通过 targetPhotonView 被发送到服务器缓存的 RPCs。主客户端和 targetPhotonView 的拥有者可以调用这个方法。

此方法必须下列条件中的一个：

- targetPhotonView 由这个客户端所拥有（即由其实例化的）。
- 这个客户端就是主客户端（可以移除任何 PhotonView 的 RPCs）。

Parameters | 参数

targetPhotonView	为这个 PhotonView 所缓冲的 RPCs 会从服务器缓冲中移除。

8.21.2.52 static void PhotonNetwork.RemoveRPCsInGroup (int targetGroup)

[static]

如果这个客户端是主客户端，或者这个客户端控制单独的 PhotonView，则移除所有在 targetGroup 里被发送到服务器缓冲的 RPCs。

此方法必须下列条件中的一个：

- 这个客户端是主客户端（可以移除组里的任何 RPCs）。
- 任何其他客户端：每一个 PhotonView 都会被检查是否在这个客户端的控制之下。只有那些被控制的 RPCs 会被移除。

Parameters | 参数

targetGroup	需要删除所有 RPCs 的目标分组。

8.21.2.53 static void PhotonNetwork.SendOutgoingCommands ( ) [static]

可以被用来立即发送 RPCs 并实例化刚刚调用的，这样 RPCs 就会在到其他玩家路上。

这可能会很有用，如果你做一个 RPC 加载关卡，然后自行加载。加载的时候，没有 RPCs 被发送给其他客户端，那么这将推迟“加载”RPC。你可以用这个的方法发送 RPC 给“别人”，

禁用消息队列（通过设置 isMessageQueueRunning）然后加载。

8.21.2.54 static void PhotonNetwork.SetLevelPrefix (short prefix) [static]

为稍后实例化的 PhotonViews 设置关卡前缀。如果你只需要一个关卡则不要设置它！

重要：如果你不使用多关卡的前缀，简单地不设置此值即可。默认值被优化来节省流量。

这个方法不会影响存在的 PhotonViews（存在的 PhotonViews 还不能被改变）。

发送的不同关卡前缀的消息将会被接收，但不会被执行。这影响了 RPCs、实例化和同步。

要知道 PUN 永远不会重置该值，你将不得不自己动手。

Parameters | 参数

prefix	受 short 类型的限制，最大值=32767。

8.21.2.55 static bool PhotonNetwork.SetMasterClient (PhotonPlayer masterClientPlayer) [static]

请求服务器指派另一个玩家作为当前房间的主客户端。

RPC 和 RaiseEvent 可以选择发送消息只到房间的主客户端。SetMasterClient 影响哪一个客户端获取这些信息。

此方法调用服务器上的一个操作来设置一个新的主客户端，这需要一个来回。在成功的情况下，此客户端和其他的客户端从服务器获得新的主客户端。

SetMasterClient 告诉服务器哪个当前主客户端应被新的代替。如果任何切换主客户端更早片刻它会失败。对于该错误没有任何回调函数。所有客户端无论怎样都应该得到服务器分配

的新主客户端。

参见: PhotonNetwork.masterClient

在版本 3 的服务器上: ReceiverGroup.MasterClient（在 RPCs 里很有用）不会被这个方法影响（仍然会指向房间里最低的 player.ID）。避免使用此枚举值（而是发送到一个具体的玩家）。

如果当前主客户端离开，PUN 将通过“最低玩家 ID”检测到一个新的主客户端。在这种情况下，实现 OnMasterClientSwitched 来得到一个回调函数。PUN 选择的主客户端可能分配一个新的主客户端。

确保你不创建一个无限循环的主客户端分配！当选择一个自定义的主客户端，所有的客户端都应该指向同一个玩家，不管谁实际分配的这个玩家。

主客户端在本地立即切换，而远程客户端获得一个事件。这就意味着游戏暂时地没有主客户端，就像一个当前主客户端离开时一样。

当手动切换主客户端时，请记住该用户可能会离开并抛弃其职责，就像任何主客户端一样。

Parameters | 参数

masterClientPlayer	下一个成为主客户端的玩家。

Returns | 返回值

当这个操作不能完成时返回 false。必须在一个房间里（不在 offlineMode 里）。

8.21.2.56 static void PhotonNetwork.SetPlayerCustomProperties (Hashtable customProperties) [static]

要设置这个（本地）玩家的属性并将其同步到其他玩家（不要直接地修改他们）。

在一个房间里时，你的属性是与其他玩家同步的。CreateRoom、JoinRoom 和 JoinRandomRoom 都将会在当你进入房间时应用玩家的自定义属性。整个哈希表将被发送。通过设置只更新的键/值来最小化流量。

如果哈希表是 null，自定义属性将被清除。自定义属性永远不会自动清除，所以他们转移到下一个房间，如果你不改变它们。

不要通过修改 PhotonNetwork.player.customProperties 设置属性！

Parameters | 参数

customProperties	这个哈希表中只有字符串类型的键将被使用。如果是 null，自定义属性都被删除。

8.21.2.57 static void PhotonNetwork.SetReceivingEnabled (int group, bool enabled) [static]

启用/禁用在给出的组上接收（应用到 PhotonViews）。

Parameters | 参数

group	要影响的利益组。
enabled	设置是否从启用的组上接收（或不接收）。

8.21.2.58 static void PhotonNetwork.SetReceivingEnabled (int[ ] enableGroups, int[ ] disableGroups) [static]

启用/禁用在给出的组上接收（应用到 PhotonViews）。

Parameters | 参数

enableGroups	要启用的利益组（或 null）。
disableGroups	要禁用的利益组（或 null）。

8.21.2.59 static void PhotonNetwork.SetSendingEnabled (int group, bool enabled) [static]

启用/禁用在给出的组上发送（应用到 PhotonViews）。

Parameters | 参数

group	要影响的利益组。
enabled	设置是否发送到启用的组（或不）。

8.21.2.60 static void PhotonNetwork.SetSendingEnabled (int[ ] enableGroups, int[ ] disableGroups) [static]

启用/禁用在给出的组上发送（应用到 PhotonViews）。

Parameters | 参数

enableGroups	要启用发送的利益组（或 null）。
disableGroups	要禁用发送的利益组（或 null）。

8.21.2.61 static void PhotonNetwork.SwitchToProtocol (ConnectionProtocol

cp) [static]

离线时，可以切换网络协议（这个操作会影响你可以用来连接的端口）。

当您切换协议时，请确保为主服务器切换端口.。默认端口是 TCP: 4530　UDP: 5055

这个操作可能看起来像这样：

Connect(serverAddress, <udpportjtcpport>, appID, gameVersion)

或者当你使用 ConnectUsingSettings()，在设置里的端口可以像这样被切换：

PhotonNetwork.PhotonServerSettings.ServerPort = 4530;

当前协议可以这样读取：

PhotonNetwork.networkingPeer.UsedProtocol

这在 PUN 的原生 socket 插件和移动设备上不起作用！

Parameters | 参数

cp	用作低级别连接的网络协议。UDP 是默认的。TCP 在所有平台上不可用。

8.21.2.62 static void PhotonNetwork.UnAllocateViewID (int viewID) [static]

注销 viewID（手动实例化并摧毁网络对象）。

Parameters | 参数

viewID	由该玩家手动分配的。

8.21.2.63 static bool PhotonNetwork.WebRpc (string name, object parameters) [static]

这个操作使 Photon 通过名称（路径）和给出的参数调用你的自定义网络服务。

这是一个服务器端的功能，必须要在 Photon 云端仪表盘（优先使用）中被设置。

详见 Turnbased 功能预览中的一段简介。

http://doc.photonengine.com/en/turnbased/current/getting-started/feature-overview

参数将被转换成 JSON 格式，所以确保你的参数是兼容的。

详见 PhotonNetworkingMessage.OnWebRpcResponse 中怎样获得一个回应。

明白 OperationResponse 只告诉 WebRPC 是否可以被调用是很重要的。响应的内容包含 Web 服务发送的任何值和错误/成功代码.。万一 Web 服务失败，错误代码和调试消息通常是在 OperationResponse 里面。

WebRpcResponse 类是一个用来从 WebRPC 回应中提取最有价值的类容的辅助类。

回调函数的实现案例:

```
public void OnWebRpcResponse(OperationResponse response)
{
    WebRpcResponse webResponse = new WebRpcResponse(operationResponse);
    if (webResponse.ReturnCode != 0) { //判断返回值
    }
    switch (webResponse.Name) { //根据反应的名称来做不同的操作
    }
    // 等等
}
```

8.21.3 Member Data Documentation | 成员数据文档

8.21.3.1 float PhotonNetwork.BackgroundTimeout = 60.0f [static]

定义 PUN 在 Unity 调用 OnApplicationPause(true)之后保持连接多少秒。默认：60 秒。

最好的做法是一段时间后断开非活动的应用程序/连接，但也允许用户去调用，等。我们认为合理的背景超时为 60 秒。

处理超时，像往常一样实现 :OnDisconnectedFromPhoton()。当应用再次变得活跃时（运行 Update() 循环）你的应用将“通知”的背景断开连接。

如果您需要将这种情况其他案例区分开来，您需要跟踪应用程序是否在后台（没有特殊的 PUN 回调）。

低于 0.1 秒的值将禁用此超时（小心：可以保持无限期连接）。

信息：PUN 正在运行一个“回调函数线程”来发送 ACKs 到服务器，即使当 Unity 没有定

期调用 Update()。

这有助于在加载场景和资产或当应用程序是在后台运行时保持连接。

注意：一些平台（如 iOS）不允许当应用程序在后台时保持连接。在这种情况下，这个值不会改变任何东西，应用程序在后台运行时立即失去连接。

某些版本的 Unity 的 OnApplicationPause()回调在一些输出(Android)里被损坏。确保在你针对的平台上 OnApplicationPause()得到你期望的回调！检查 PhotonHandler.OnApplicationPause(bool pause)，来查看其实现。

8.21.3.2 bool PhotonNetwork.InstantiateInRoomOnly = true [static]

如果是 true，实例化方法将检查你是否在一个房间里，并且如果你不在房间里就会失败。在一个特定的房间外实例化任何东西都非常容易损坏。如果你知道该怎么做可以关闭这个。

8.21.3.3 PhotonLogLevel PhotonNetwork.logLevel = PhotonLogLevel.ErrorsOnly [static]

网络日志级别。控制 PUN 的冗长程度。

8.21.3.4 readonly int PhotonNetwork.MAX_VIEW_IDS = 1000 [static]

每个玩家（或场景）指派的 PhotonViews 的最大数量。详见 General Documentation 主题“Limitations”中如何提高这个限制。

8.21.3.5 int PhotonNetwork.maxConnections [static]

只在 Unet 中使用。在 PUN 里，在 PhotonNetwork.CreateRoom 里设置玩家的数量。

8.21.3.6 EventCallback PhotonNetwork.OnEventCall [static]

在这里使用"+="来注册你的 RaiseEvent 处理方法。

任何 eventCode < 200 的事件将被转发到你的代理。

8.21.3.7 ServerSettings PhotonNetwork.PhotonServerSettings = (ServerSettings)Resources.Load(PhotonNetwork.serverSettingsAssetFile, typeof(ServerSettings)) [static]

序列化的服务器设置，在安装向导中写入，在 ConnectUsingSettings 中使用。

8.21.3.8 float PhotonNetwork.precisionForFloatSynchronization = 0.01f [static]

浮点数之间在我们通过一个 PhotonView 的 OnSerialize/ObservingComponent 组件发送之前的最小差异。

8.21.3.9 float PhotonNetwork.precisionForQuaternionSynchronization = 1.0f [static]

一个旋转 Rotation 在我们通过一个 PhotonView 的 OnSerialize/ObservingComponent 组件发送之前需要改变的最小角度。

8.21.3.10 float PhotonNetwork.precisionForVectorSynchronization = 0.000099f [static]

一个二维向量或三维向量（例如一个 transform 的旋转 Rotation）在我们通过一个 PhotonView 的 OnSerialize/ObservingComponent 组件发送之前需要改变的最小精度。

注意这是一个 sqrMagnitude。例如只有在 Y 轴上 0.01 改变之后发送，我们使用 0.01f*0.01f=0.0001f。

作为一项补救浮点数不精确的措施我们使用 0.000099f 代替 0.0001f。

8.21.3.11 Dictionary<string, GameObject> PhotonNetwork.PrefabCache = new Dictionary<string, GameObject>()[static]

保持游戏对象的引用来频繁的实例化（内存不足取代加载资源）。

你应该可以在任何时间修改缓存，除了在实例化被使用的时候。最好只在主线程中这样做。

8.21.3.12 HashSet<GameObject> PhotonNetwork.SendMonoMessageTargets [static]

如果不为 null，这是（独家）被 PUN 的 SendMonoMessage()调用的游戏对象列表。

对于所有在 PhotonNetworkingMessage 里定义的回调函数，PUN 将使用 SendMonoMessage 方法，并调用 FindObjectsOfType()方法来寻找所有可能想要一个 PUN 回调的脚本和游戏对象。

PUN 回调是非常频繁的（在游戏里，属性更新是最频繁的），但是 FindObjectsOfType 是耗费时间的，在有大量游戏对象的情况下性能会受到影响。

可选，SendMonoMessageTargets 可以被用来提供一个目标对象列表。这样会跳过 FindObjectsOfType()，但任何需要回调的对象将必须添加自身到这个列表。

如果为 null，默认行为是每个游戏对象上用 MonoBehaviour 做一个 SendMessage。

8.21.3.13 Type PhotonNetwork.SendMonoMessageTargetType = typeof(MonoBehaviour) [static]

定义哪些类可以包含 PUN 回调实现。

这提供了优化运行速度的选项.。

此类型越具体，越少的类将被用反射来检查回调方法。

8.21.3.14 bool PhotonNetwork.StartRpcsAsCoroutine = true [static]

用于跳过启动 RPCs 作为协程，它可以是一个性能问题。

8.21.3.15 bool PhotonNetwork.UsePrefabCache = true [static]

当启用（true）时，实例化使用 PhotonNetwork.PrefabCache 来保持游戏对象在内存中（优化相同预制体的实例化）。

在运行时设置 UsePrefabCache 到 false 将不会清除 PrefabCache，但马上就忽略它。你可以自己清理和修改缓存。

8.21.3.16 bool PhotonNetwork.UseRpcMonoBehaviourCache [static]

启用时，调用 RPCs 的 MonoBehaviours 会被缓存，避免消耗大的 GetComponents<MonoBehaviour>()方法调用。

RPCs 在目标 PhotonView 的 MonoBehaviours 上被调用。那些 MonoBehaviour 必须通过 GetComponents 来找到。

当这个布尔值设置到 true 时，每一个 PhotonView 的 MonoBehaviours 列表得到缓存。你可以根据需求来使用 photonView.RefreshRpcMonoBehaviourCache()方法手动刷新 PhotonView 的 MonoBehaviours 列表（例如，当一个新的 MonoBehaviour 被添加一个网络游戏对象上时）。

8.21.3.17 const string PhotonNetwork.versionPUN = "1.78"

PUN 的版本号。也被用于游戏版本中来相互区分客户端版本。

8.21.4 Property Documentation | 属性文档

8.21.4.1 AuthenticationValues PhotonNetwork.AuthValues [static], [get], [set]

用于连接时的用户授权值。

如果你想要自定义身份验证，请在调用 Connect 方法前设置这些。这些值设置的 userId，userId 是否和如何得到验证（服务器端）等。

如果任何值认证失败，PUN 将调用你实现的 OnCustomAuthenticationFailed(string debugMsg)回调。详见：PhotonNetworkingMessage.OnCustomAuthenticationFailed

8.21.4.2 bool PhotonNetwork.autoCleanUpPlayerObjects [static], [get], [set]

此设置每个房间都要定义，当网络实例化的游戏对象（有 PhotonView）的创造者离开时这些对应的对象是否得到清理。

这个设置是每个房间都做的。它不能在房间里改变，并且它将重写个别客户端的设置。

如果房间里的 room.AutoCleanUp 被启用，PUN 客户端将在玩家离开时摧毁对应的游戏对象。这包括手动实例化的游戏对象（例如，通过 RPCs）。当启用时，服务器也将清除离开玩家的 RPCs、实例化的游戏对象和 PhotonViews。并且在有人离开后面加入的玩家不会收到离开玩家的事件。

在引擎盖下，此设置存储为自定义房间属性。默认启用。

8.21.4.3 bool PhotonNetwork.autoJoinLobby [static], [get], [set]

在 PhotonServerSetting 资产中设置。定义当客户端连接到主服务器时 PhotonNetwork 是否应该加入“lobby”。

如果这是 false，当到主服务器的连接可用时 OnConnectedToMaster()将被调用，而 OnJoinedLobby()将不会被调用。

默认启用。

房间列表将不可用。房间可以在没有加入游戏大厅的情况下被创建和加入（随机地）（并发送房间列表）。

8.21.4.4 bool PhotonNetwork.automaticallySyncScene [static], [get], [set]

Defines if all clients in a room should load the same level as the Master Client (if that used PhotonNetwork.LoadLevel).定义房间里的所有客户端是否应该加载和主客户端一样的关卡（如果主客户端使用了 PhotonNetwork.LoadLevel）。

要同步加载的关卡，主客户端应该使用 PhotonNetwork.LoadLevel。所有的客户端在得到更新或加入时加载该新场景。

在内部，自定义房间属性被设置来加载场景。当一个客户端读取时，又还不在同一个场景里，它将会立即暂停消息队列（PhotonNetwork.isMessageQueueRunning = false），并加载。当场景完成加载后，PUN 会自动重新启用消息队列。

8.21.4.5 bool PhotonNetwork.connected [static], [get]

直到你最初连接到 Photon 时为 false。在脱机模式下、连接到任何服务器时、甚至在切换服务器时为 true。

8.21.4.6 bool PhotonNetwork.connectedAndReady [static], [get]

只有当您到服务器的连接已准备好接受像加入、离开等一类操作时才是 true，这是一个精炼的 **connected** 版本。

8.21.4.7 bool PhotonNetwork.connecting [static], [get]

当你调用 ConnectUsingSettings（或类似的）方法，直到客户端到 Photon 服务器的低

等级连接被建立了时，才为 true。

8.21.4.8 ConnectionState PhotonNetwork.connectionState [static], [get]

简化的连接状态。

8.21.4.9 ClientState PhotonNetwork.connectionStateDetailed [static], [get]

详细的连接状态（忽略 PUN，因此在切换服务器时可以“断开连接”）。

在 OfflineMode 里，这是 ClientState。已加入（在创建/加入之后）或在其他的所有情况下是 ConnectedToMaster 。

8.21.4.10 int PhotonNetwork.countOfPlayers [static], [get]

正在使用这个应用的玩家数量（在主服务器上每 5 秒间隔可用）。

8.21.4.11 int PhotonNetwork.countOfPlayersInRooms [static], [get]

目前正在应用中的房间里玩游戏的用户数量（主服务器每 5 秒发送一次）。使用 playerList.Count 来获取你所在房间里的玩家数量！

8.21.4.12 int PhotonNetwork.countOfPlayersOnMaster [static], [get]

现在正在寻找房间的玩家数量（在主服务器上每 5 秒间隔可用）。

8.21.4.13 int PhotonNetwork.countOfRooms [static], [get]

目前正在使用的房间数量（在主服务器上每 5 秒间隔可用）

在游戏大厅里时也可以检查房间列表的数量，例如：PhotonNetwork.GetRoomList().Length。

由于 PUN 版本 v1.25 只是基于统计 Photon 发送的事件（计数所有房间）。

8.21.4.14 bool PhotonNetwork.CrcCheckEnabled [static], [get], [set]

CRC 检查用来发现和避免破包问题时很有用。可以在没有连接的时候启用。

8.21.4.15 bool PhotonNetwork.EnableLobbyStatistics [static], [get], [set]

在 PhotonServerSettings 资产里设置。启用来从主服务器获取一个活跃游戏大厅列表。

如果在游戏中使用多个游戏大厅，并且你想要显示每个玩家的活动，大厅的统计数据可以是有用的。

该值被保存在 PhotonServerSettings 里。

当你连接到主服务器时 PhotonNetwork.LobbyStatistics 被更新。同样有一个对应的回调 PunBehaviour。

8.21.4.16 List<FriendInfo> PhotonNetwork.Friends [static], [get], [set]

只读好友列表，他们的在线状态和所在的房间。直到由 FindFriends 调用初始化为 null。

不要修改这个列表！它是由 FindFriends 内部处理的，并只可读取值。FriendListAge 告诉你该数据已经存在多少毫秒了。

不要频繁获取这个列表（> 10 秒）。在最好的情况下，保持你获取的列表尽量简短。你可以（举个例子）只获取一次完整的列表，然后只请求在线好友的少量更新。过了一会儿（例如 1 分钟），你可以再次获取完整的列表（用来更新在线状态）。

8.21.4.17 int PhotonNetwork.FriendsListAge [static], [get]

好友列表信息的年龄（以毫秒计量）。直到获取好友列表之前都为 0.

8.21.4.18 string PhotonNetwork.gameVersion [static], [get], [set]

你构建的版本字符串。可以被用来区分不兼容的客户端。在连接的时候发送。

.这是只在连接时发送，也会放置在通常设置的地方（例如在 ConnectUsingSettings 里）。

8.21.4.19 bool PhotonNetwork.inRoom [static], [get]

在房间里时为 true（connectionStateDetailed == ClientState.Joined）。很多动作只能在房间里被执行，例如实例化或离开，等等。你也可以在离线模式里加入房间。

8.21.4.20 bool PhotonNetwork.insideLobby [static], [get]

当客户端在游戏大厅里时为 true。

当房间列表变成可用或被更新时实现 IPunCallbacks.OnReceivedRoomListUpdate()来通知。

当你加入一个房间时会自动离开所在的任何游戏大厅！大厅只存在在主服务器上（那里房间由游戏服务器处理）。

8.21.4.21 bool PhotonNetwork.isMasterClient [static], [get]

我们是主客户端吗？

8.21.4.22 bool PhotonNetwork.isMessageQueueRunning [static], [get], [set]

可以被用来暂停对收到的事件（RPCs、实例化和任何其他进来的事件）进行调度。

当 IsMessageQueueRunning == false 时，OnPhotonSerializeView 不会被完成，并且客户端不会发送任何消息。同样的，进来的消息将被放进排队，直到你重新激活消息队列。

如果你首先想要加载一个关卡，然后继续接收 PhotonViews 和 RPCs 数据，这个布尔值会很有用。客户端将继续为进来的数据包和 RPCs/Events 接收、发送确认。当暂停时间过长时会增加“延迟”并造成一些问题，因为所有进来的消息被排队。

8.21.4.23 bool PhotonNetwork.isNonMasterClientInRoom [static], [get]

如果我们在一个房间（客户端）里并且不是该房间的主客户端，则为 true。

8.21.4.24 TypedLobby PhotonNetwork.lobby [static], [get], [set]

当 PUN 加入游戏大厅或创建一个游戏时将要使用的大厅。

默认大厅使用一个空字符串作为名称。如果 autoJoinLobby 被设置为 true，PUN 将进入一个在主服务器上的大厅。所以当你连接或离开一个房间时 ,PUN 自动把你再次放进一个大厅。

通过检查 PhotonNetwork.insideLobby 来判断客户端是否在一个游戏大厅内。

8.21.4.25 List<TypedLobbyInfo> PhotonNetwork.LobbyStatistics [static], [get], [set]

如果启用该列表，主服务器将为这个应用程序提供关于活跃大厅的信息。

如果在游戏中使用多个游戏大厅，并且你想要显示每个玩家的活动，大厅的统计数据可以是有用的。每个大厅，你可以获取其：名称，类型，房间和玩家计数。

当你连接到主服务器时 PhotonNetwork.LobbyStatistics 被更新。同样有一个对应的回调函数 PunBehaviour.OnLobbyStatisticsUpdate，你应该实现该回调来更新你的 UI。

默认情况下不会打开大堂统计信息。在项目的 PhotonServerSettings 文件里启用。

8.21.4.26 PhotonPlayer PhotonNetwork.masterClient [static], [get]

当前房间的主客户端或 null（不在房间内）。

可以被用于“权威的”客户端/玩家做决定，运行 AI 或其他。

如果当前的主客户端离开房间（leave/disconnect），服务器将很快指派其他客户端。如果当前主客户端超时（关闭应用，丢失连接，等等），发送到这个客户端的消息对于其他客户端来说就有效地丢失了！一个超时可以花费 10 秒，在这个时间内没有活跃的主客户端。

实现 IPunCallbacks.OnMasterClientSwitched 方法以供在主客户端切换时调用。

使用 PhotonNetwork.SetMasterClient，来手动切换到一些别的玩家/客户端。

offlineMode == true 时，该属性总是返回 PhotonNetwork.player.

8.21.4.27 int PhotonNetwork.MaxResendsBeforeDisconnect [static], [get], [set]

定义一个可靠消息在没有得到 ACK 之前可以被重新发送的时间数，没有得到服务器确认会触发断开连接。

默认值：5。

更少重新发送数意味着更快断连，而更多则会导致大量的延迟。最小值：3.最大值：10.

8.21.4.28 bool PhotonNetwork.NetworkStatisticsEnabled [static], [get], [set]

启用或禁用关于客户端流量的数据收集。

如果你遇到客户端问题，流量统计是寻找解决办法的一个好的起点。只有启用统计，你才可以使用 GetVitalStats 方法。

8.21.4.29 bool PhotonNetwork.offlineMode [static], [get], [set]

离线模式可以被设置来在单人游戏模式中重用你的多人游戏代码。当这是在 PhotonNetwork 上的离线模式时，将不会创建任何连接，并且几乎没有开销。对于重用 RPC 和 PhotonNetwork.Instantiate 来说最有用。

8.21.4.30 PhotonPlayer [ ] PhotonNetwork.otherPlayers [static], [get]

在当前房间内的玩家列表，除了本地玩家。

这个列表只有在客户端在房间里时才有效。当某个玩家加入或离开时列表会自动更新。

这被用来列出房间里的所有其他玩家。每个玩家的 PhotonPlayer.customProperties 都是可以被访问的（通过 PhotonPlayer.SetCustomProperties 方法设置并同步）。

你可以使用一个 PhotonPlayer.TagObject 来储存一个任意对象用于引用。这个对象不会通过网络被同步。

8.21.4.31 int PhotonNetwork.PacketLossByCrcCheck [static], [get]

如果 CrcCheckEnabled 被启用，这个会计数那些没有有效 CRC 校验并被拒绝的数据包。

8.21.4.32 PhotonPlayer PhotonNetwork.player [static], [get]

代表本地的 PhotonPlayer。总是可用并代表这个玩家。CustomProperties 可以在进入房间之前被设置，并也将被同步。

8.21.4.33 PhotonPlayer [ ] PhotonNetwork.playerList [static], [get]

当前房间里的玩家列表，包括本地玩家。

这个列表只有在客户端在房间里时才有效。当某个玩家加入或离开时列表会自动更新。

这被用来列出房间里的所有玩家。每个玩家的 PhotonPlayer.customProperties 都是可以被访问的（通过 PhotonPlayer.SetCustomProperties 方法设置并同步）。

你可以使用一个 PhotonPlayer.TagObject 来储存一个任意对象用于引用。这个对象不会通过网络被同步。

8.21.4.34 string PhotonNetwork.playerName [static], [get], [set]

设置来和玩家所进入房间的所有人同步玩家的昵称。这会设置 PhotonPlayer.name 的值。

playerName 只是一个昵称，并且不需要是唯一的或用一些账户来备份。

可以在任何时候（例如在你连接之前）设置该值，并且它对于和你一起游戏的任何玩家都是可用的。

通过 PhotonPlayer.name 来访问玩家的名称。

8.21.4.35 IPunPrefabPool PhotonNetwork.PrefabPool [static], [get], [set]

一个对象池可以被用来保留和重用实例化的对象实例。它取代了 Unity 的默认实例化和摧毁方法。

要使用一个游戏对象池，实现 IPunPrefabPool 接口并在这里指派。预制体是通过名称来确定的。

8.21.4.36 int PhotonNetwork.QuickResends [static], [get], [set]

在网络丢失的情况下，可靠的消息可以迅速重复发送，多达 3 次。

当可靠的消息丢失不止一次时，后续的重复会延迟一点以允许网络恢复..

使用此选项，可以加快重复 2 和 3 次。这可以帮助避免超时，但也会增加缺口关闭的速度。

当你设置这个值，会增加 PhotonNetwork.MaxResendsBeforeDisconnect 到 6 或 7.

8.21.4.37 int PhotonNetwork.ResentReliableCommands [static], [get]

重复发送的命令数量（收到 ACK 之前本地的重复时间）。

如果该值增加很多，有很大的几率由于恶劣的条件，超时断开会发生。

8.21.4.38 Room PhotonNetwork.room [static], [get]

获取当前所在的房间。如果我们没有在任何房间里则为 null。

8.21.4.39 int PhotonNetwork.sendRate [static], [get], [set]

定义每秒 PhotonNetwork 应该发包多少次。如果你改变该值，不要忘了也要改变' sendRateOnSerialize' 。

越少数据包就越少开销，但是更多延迟。设置 sendRate 到 50 将增加到每秒 50 个数据包（这是很大的量！）。记住你的目标平台：手机网络更慢且更不可靠。

8.21.4.40 int PhotonNetwork.sendRateOnSerialize [static], [get], [set]

定义了 OnPhotonSerialize 每秒应该调用 PhotonViews 多少次。

改变该值会关联到 PhotonNetwork.sendRate。OnPhotonSerialize 将创建更新并发送消息。

较低的速率占用较少的性能，但会导致更多的滞后。

8.21.4.41 ServerConnection PhotonNetwork.Server [static], [get]

这个客户端当前连接到的或正在连接的服务器（类型）。

Photon 使用 3 种不同的服务器角色：Name Server、Master Server 和 Game Server。

8.21.4.42 string PhotonNetwork.ServerAddress [static], [get]

当前使用的服务器地址（无论是主服务器或游戏服务器）。

8.21.4.43 int PhotonNetwork.ServerTimestamp [static], [get]

当前服务器的毫秒级的时间戳。

这在同步同个房间内所有客户端的动作和事件时很有用。该时间戳是基于服务器的 Environment.TickCount。

它会经常溢出从一个正值到负值，所以要注意当事情发送的时候只使用时间的差异来检查时间变化。

这是 PhotonNetwork.time 的基础。

8.21.4.44 double PhotonNetwork.time [static], [get]

Photon 网络时间，和服务器同步。

v1.55 版本

这个时间值取决于服务器的 Environment.TickCount。每一个服务器都不同，但是在同一个房间里的所有客户端应该有相同的值（同一个房间只会在一个服务器上）。

这不是 DateTime!

谨慎使用此值:

它可以从任何正值开始。

它将“环绕”从 4294967.295 到 0！

8.21.4.45 int PhotonNetwork.unreliableCommandsLimit [static], [get], [set]

每次调度使用一次以限制每个通道的不可靠的命令（所以暂停之后，许多通道仍然会导致很多不可靠的命令）。

8.22 PhotonPingManager Class Reference | 参考

Public Member Functions | 公共成员函数

- IEnumerator PingSocket (Region region)

Static Public Member Functions | 静态公共成员函数

- static string ResolveHost (string hostName)

尝试解析主机名到 IP 字符串或在失败时返回空字符串。

Public Attributes | 公共属性

- bool UseNative

Static Public Attributes | 静态公共属性

- static int Attempts = 5

- static bool IgnoreInitialAttempt = true
- static int MaxMilliseconsPerPing = 800

Properties | 属性

- Region BestRegion [get]
- bool Done [get]

8.22.1 Member Function Documentation | 成员函数文档

8.22.1.1 IEnumerator PhotonPingManager.PingSocket (Region region)

会被应用的帧率影响，因为这个协程每帧检查一次 socket 结果。

8.22.1.2 static string PhotonPingManager.ResolveHost (string hostName)

[static]

尝试解析主机名到 IP 字符串或在失败时返回空字符串。

要与大多数平台兼容，地址族会这样检查：

```
if (ipAddress.AddressFamily.ToString().Contains("6")) // ipv6
```

Parameters | 参数

hostName	要解析的主机名。

Returns | 返回值

返回解析好的 IP 字符串或在解析失败时空字符串。

8.22.2 Member Data Documentation | 成员数据文档

8.22.2.1 int PhotonPingManager.Attempts = 5 [static]

8.22.2.2 bool PhotonPingManager.IgnoreInitialAttempt = true [static]

8.22.2.3 int PhotonPingManager.MaxMilliseconsPerPing = 800 [static]

8.22.2.4 bool PhotonPingManager.UseNative

8.22.3 Property Documentation | 属性文档

8.22.3.1 Region PhotonPingManager.BestRegion [get]

8.22.3.2 bool PhotonPingManager.Done [get]

8.23 PhotonPlayer Class Reference | 玩家类参考

总结了一个房间内的“玩家”，由 actorID 来确定身份（在房间里）。

继承 IComparable< PhotonPlayer >, IComparable< int >, IEquatable< PhotonPlayer >, 和 IEquatable< int>。

Public Member Functions | 公共成员函数

- PhotonPlayer (bool isLocal, int actorID, string name)

创建一个 PhotonPlayer 实例。

- override bool Equals (object p)

使 PhotonPlayer 具有可比性。

- override int GetHashCode ()
- void SetCustomProperties (Hashtable propertiesToSet, Hashtable expectedValues=null, bool webForward=false)

用新的/已更新的键-值对更新这个玩家的自定义属性。

- PhotonPlayer Get (int id)
- PhotonPlayer GetNext ()
- PhotonPlayer GetNextFor (PhotonPlayer currentPlayer)
- PhotonPlayer GetNextFor (int currentPlayerId)
- int CompareTo (PhotonPlayer other)
- int CompareTo (int other)
- bool Equals (PhotonPlayer other)
- bool Equals (int other)
- override string ToString ()

PhotonPlayer 的字符串小结。包括 name 或 player.ID，以及该玩家是否是主客户端。

- string ToStringFull ()

PhotonPlayer 的字符串总结：player.ID、name 和该用户的所有自定义的属性。

Static Public Member Functions | 静态公共成员函数

- static PhotonPlayer Find (int ID)

尝试通过 id 获取一个指定玩家。

Public Attributes | 公共属性

- readonly bool isLocal = false

每个客户端只掌控一个玩家。其他的玩家则并非本地的。

- object TagObject

可以用来存储一个引用。

Protected Member Functions | 受保护的成员函数

- PhotonPlayer (bool isLocal, int actorID, Hashtable properties)

内部用于创建从事件加入的玩家。

Properties | 属性

- int ID [get]

玩家的 actorID。

- string name [get, set]

玩家的昵称。

- string userId [get, set]

玩家的 UserId，当房间被创建时（RoomOptions.PublishUserId = true）可用。

- bool isMasterClient [get]

如果该玩家是当前房间的主客户端则为 true。

- bool isInactive [get, set]

当房间的 PlayerTTL>0 时，该房间里的玩家可能是未激活的。如果为 true，该玩家不会从这个房间获取事件（现在），但是可以稍后返回。

- Hashtable customProperties [get, set]

玩家的自定义属性的只读缓存。通过 PhotonPlayer.SetCustomProperties 设置。

- Hashtable allProperties [get]

用所有属性（包含自定义属性和“众所周知”的属性）来创建一个哈希表。

8.23.1 Detailed Description | 详细描述

总结了一个房间内的“玩家”，由 actorID 来确定身份（在房间里）。

每个玩家都有一个对房间有效的 actorId (或 ID)，直到被服务器指派前都是-1。每个客户端都可以用 SetCustomProperties 方法来设置其玩家的自定义属性，甚至在进入房间之前。自定义属性会在加入房间时被同步。

8.23.2 Constructor & Destructor Documentation | 构造和析构文档

8.23.2.1 PhotonPlayer.PhotonPlayer (bool isLocal, int actorID, string name)

创建一个 PhotonPlayer 实例。

Parameters | 参数

isLocal	该玩家是否是本地玩家（或一个远程玩家）。
actorID	当前房间里的玩家的 ID 或 ActorNumber(识别房间里的每一个玩家的快捷方式）。
name	玩家的名称（一个“众所周知的属性”）。

8.23.2.2 PhotonPlayer.PhotonPlayer (bool isLocal, int actorID, Hashtable properties) [protected]

内部用来从事件 Join 中创建玩家。

8.23.3 Member Function Documentation | 成员函数文档

8.23.3.1 int PhotonPlayer.CompareTo (PhotonPlayer other)

8.23.3.2 int PhotonPlayer.CompareTo (int other)

8.23.3.3 override bool PhotonPlayer.Equals (object p)

使 PhotonPlayer 具有可比性。

8.23.3.4 bool PhotonPlayer.Equals (PhotonPlayer other)

8.23.3.5 bool PhotonPlayer.Equals (int other)

8.23.3.6 static PhotonPlayer PhotonPlayer.Find (int ID) [static]

尝试通过 id 来获取指定的玩家。

Parameters | 参数

ID	ActorID

Returns | 返回值

返回匹配 actorID 的玩家，或在没有使用 actorID 的玩家时返回 null。

8.23.3.7 PhotonPlayer PhotonPlayer.Get (int id)

8.23.3.8 override int PhotonPlayer.GetHashCode ( )

8.23.3.9 PhotonPlayer PhotonPlayer.GetNext ( )

8.23.3.10 PhotonPlayer PhotonPlayer.GetNextFor (PhotonPlayer currentPlayer)

8.23.3.11 PhotonPlayer PhotonPlayer.GetNextFor (int currentPlayerId)

8.23.3.12 void PhotonPlayer.SetCustomProperties (Hashtable propertiesToSet, Hashtable expectedValues = null, bool webForward = false)

用新的/已更新的键-值对更新这个玩家的自定义属性。

自定义属性是一个对同房间内的所有玩家可用的键值对集合（哈希表）。它们可以关联到所在的房间或个别玩家，当只有某件事物的当前值是有用的时非常有用。例如：房间地图。所有的键必须是字符串。

Room 和 PhotonPlayer 类都有 SetCustomProperties 方法。同样的，两个类都通过 customProperties 提供访问当前键值对。

总是使用 SetCustomProperties 来改变值。为了减少网络流量，只设置实际改变了的值。

当新的属性被添加或存在的值被更新时，其他值将不会被改变，所以只需提供改变了的值或新增的。

要删除一个该房间里的已命名的（自定义）属性，使用 null 来作为值即可。

在本地，SetCustomProperties 将无延迟地更新其缓存。其他客户端通过 Photon（服务器）恰当的操作来被更新。

Check and Swap | 检查和交换

SetCustomProperties 可以选择做一个服务器端的 Check-And-Swap (CAS)：只有在期望的值是正确的时候才会被更新。expectedValues 可以和 propertiesToSet 不同的键/值。所以你可以检查一些键并设置另一个键的值（如果检查成功了）。

如果客户端已知的属性是错误的或过期的，则不能通过 CAS 设置值。CAS 被用来避免玩家同时设置值时很有用。举个例子：如果所有的玩家都尝试拾取一些卡片或物品，只有一个玩家可以获取它。有了 CAS，只有第一个 SetProperties 会在服务器端得到执行，其他的（同时被发送）会失败。

服务器将广播成功地改变了值，本地的 customProperties“缓存”只有在往返后得到更

新（如果有什么变化）。

你可以做一个"webForward"：Photon 将发送改变了的属性到为你的应用定义的 WebHook。

OfflineMode | 离线模式

当 PhotonNetwork.offlineMode 为 true 时，expectedValues 和 webForward 参数会被忽略。在离线模式里，本地的 customProperties 值被立即更新（无需往返服务器）。

Parameters | 参数

propertiesToSet	要设置的新属性。
expectedValues	至少一个属性键/值集，服务器端检查。键和值必须正确。离线模式里被忽略。
webForward	设置为 true，把设置属性推送到一个在仪表盘里为应用定义的 WebHook。离线模式里被忽略。

8.23.3.13 override string PhotonPlayer.ToString ( )

PhotonPlayer 的字符串小结。包括 name 或 player.ID，以及该玩家是否是主客户端。

8.23.3.14 string PhotonPlayer.ToStringFull ( )

PhotonPlayer 的字符串总结：player.ID、name 和该用户的所有自定义的属性。

谨慎使用且不能每一帧使用！在每一次调用时把 customProperties 转化成一个字符串。

8.23.4 Member Data Documentation | 成员数据文档

8.23.4.1 readonly bool PhotonPlayer.isLocal = false

每一个客户端只能控制一个玩家。其他的都不是本地的。

8.23.4.2 object PhotonPlayer.TagObject

可以被用来储存一个对玩家有用的引用。

例如：在实例化时通过指派该游戏对象来设置玩家的角色作为标签。

8.23.5 Property Documentation | 属性文档

8.23.5.1 Hashtable PhotonPlayer.allProperties [get]

用所有的属性（自定义和“众所周知的”属性）创建一个哈希表。

如果很常用，可以将其缓存。

8.23.5.2 Hashtable PhotonPlayer.customProperties [get], [set]

玩家的自定义属性的只读缓存。通过 PhotonPlayer.SetCustomProperties 设置。

不要修改这个哈希表的内容。使用 SetCustomProperties 和这个类的属性来修改值。当你使用它们的时候，客户端将与服务器同步值。

8.23.5.3 int PhotonPlayer.ID [get]

该玩家的 actorID。

8.23.5.4 bool PhotonPlayer.isInactive [get], [set]

当房间的 PlayerTTL>0 时，该房间里的玩家可能是未激活的。如果为 true，该玩家不会从这个房间获取事件（现在），但是可以稍后返回。

8.23.5.5 bool PhotonPlayer.isMasterClient [get]

如果该玩家是当前房间的主客户端则为 true。

详见：PhotonNetwork.masterClient.

8.23.5.6 string PhotonPlayer.name [get], [set]

这个玩家的昵称。

设置 PhotonNetwork.playerName 来使其在房间内同步。

8.23.5.7 string PhotonPlayer.userId [get], [set]

玩家的 UserId，当房间创建后（RoomOptions.PublishUserId = true）可用。

对于 PhotonNetwork.FindFriends 和为期待的玩家预留位置很有用（例如在 PhotonNetwork.CreateRoom 里面）。

8.24 PhotonRigidbody2DView Class Reference

这个类帮助你同步 2d 物理刚体组件的速度。注意只有速度被同步，因为 Unity 的物理引擎的不确定性（例如，其结果在所有的电脑上不尽相同）-对象的实际位置可能在同步之外。如果你想要让该对象的位置在所有客户端上都一样，你也应该添加一个 PhotonTransformView 来同步位置。简单地添加该组件到你的游戏对象上，并确保 PhotonRigidbody2DView 被添加到观察组件的列表。

继承 MonoBehaviour 和 IPunObservable 接口。

Public Member Functions | 公有成员函数

- void OnPhotonSerializeView (PhotonStream stream, PhotonMessageInfo info)

每秒被 PUN 调用几次，这样你的脚本可以为 PhotonView 读写同步数据。

8.24.1 Detailed Description | 详细描述

这个类帮助你同步 2d 物理刚体组件的速度。注意只有速度被同步，因为 Unity 的物理引

擎的不确定性（例如，其结果在所有的电脑上不尽相同）-对象的实际位置可能在同步之外。如果你想要让该对象的位置在所有客户端上都一样，你也应该添加一个 PhotonTransformView 来同步位置。简单地添加该组件到你的游戏对象上，并确保 PhotonRigidbody2DView 被添加到观察组件的列表。

8.24.2 Member Function Documentation | 成员函数文档

8.24.2.1 void PhotonRigidbody2DView.OnPhotonSerializeView (PhotonStream stream, PhotonMessageInfo info)

每秒被 PUN 调用几次，这样你的脚本可以为 PhotonView 读写同步数据。

该方法将被在那些被指派作为 PhotonView 的观察组件的脚本里调用。

PhotonNetwork.sendRateOnSerialize 会影响该方法被调用的频率。

PhotonNetwork.sendRate 影响这个客户端的发包频率。

实现该方法，你可以自定义哪些数据 PhotonView 定时同步。你的代码定义什么正在被发送（内容），以及你的数据被接收的客户端使用。

不像其他回调函数，OnPhotonSerializeView 只有在被指派到一个 PhotonView 做为 PhotonView.observed 脚本时被调用。

要利用这个方法，PhotonStream 是必须的。它将会在客户端上是“写入模式”，从而控制 PhotonView（PhotonStream.isWriting == true），而在远程客户端上是“读取模式”，只是接收控制客户端发送的数据。

如果你跳过写入任何值到二进制流中，PUN 将跳过更新。谨慎使用，这可以节省宽带和消息（每个房间/秒都会有一个限制）。

注意当发送者没有发送任何更新时，OnPhotonSerializeView 不会在远程客户端上被调用。这不能被用来作为"x-次每秒的 Update()"。

实现 IPunObservable 接口。

8.25 PhotonRigidbodyView Class Reference | 参考

这个类帮助你同步物理刚体组件的速度。注意只有速度被同步，因为 Unity 的物理引擎的不确定性（例如，其结果在所有的电脑上不尽相同）-对象的实际位置可能在同步之外。如果你想要让该对象的位置在所有客户端上都一样，你也应该添加一个 PhotonTransformView 来同步位置。简单地添加该组件到你的游戏对象上，并确保 PhotonRigidbodyView 被添加到观察组件的列表。

继承 MonoBehaviour 和 IPunObservable 接口。

Public Member Functions | 公共成员函数

- void OnPhotonSerializeView (PhotonStream stream, PhotonMessageInfo info)

每秒被 PUN 调用几次，这样你的脚本可以为 PhotonView 读写同步数据。

8.25.1 Detailed Description | 详细描述

这个类帮助你同步物理刚体组件的速度。注意只有速度被同步，因为 Unity 的物理引擎的不确定性（例如，其结果在所有的电脑上不尽相同）-对象的实际位置可能在同步之外。如果你想要让该对象的位置在所有客户端上都一样，你也应该添加一个 PhotonTransformView 来同步位置。简单地添加该组件到你的游戏对象上，并确保 PhotonRigidbodyView 被添加到观察组件的列表。

8.25.2 Member Function Documentation | 成员函数文档

8.25.2.1 void PhotonRigidbodyView.OnPhotonSerializeView (PhotonStream stream, PhotonMessageInfo info)

每秒被 PUN 调用几次，这样你的脚本可以为 PhotonView 读写同步数据。

该方法将被在那些被指派作为 PhotonView 的观察组件的脚本里调用。

PhotonNetwork.sendRateOnSerialize 会影响该方法被调用的频率。

PhotonNetwork.sendRate 影响这个客户端的发包频率。

实现该方法，你可以自定义哪些数据 PhotonView 定时同步。你的代码定义什么正在被发送（内容），以及你的数据被接收的客户端使用。

不像其他回调函数，OnPhotonSerializeView 只有在被指派到一个 PhotonView 做为 PhotonView.observed 脚本时被调用。

要利用这个方法，PhotonStream 是必须的。它将会在客户端上是“写入模式”，从而控制 PhotonView（PhotonStream.isWriting == true），而在远程客户端上是“读取模式”，只是接收控制客户端发送的数据。

如果你跳过写入任何值到二进制流中，PUN 将跳过更新。谨慎使用，这可以节省宽带和消息（每个房间/秒都会有一个限制）。

注意当发送者没有发送任何更新时，OnPhotonSerializeView 不会在远程客户端上被调用。这不能被用来作为"x-次每秒的 Update()"。

实现 IPunObservable 接口。

8.26 PhotonStatsGui Class Reference | 类参考

展示 Photon 连接的流量和健康统计的基础 GUI，通过快捷键 shift+tab 来开关。

继承 MonoBehaviour.

Public Member Functions | 公共成员函数

- void Start ()
- void Update ()

检查 shift+tab 输入组合键（用来开关 statsOn）。

- void OnGUI ()
- void TrafficStatsWindow (int windowID)

Public Attributes | 公共字段

- bool statsWindowOn = true

展示或隐藏 GUI（不会影响统计数据是否被收集）。

- bool statsOn = true

开关搜集数据的选项（在 Update()里使用）。

- bool healthStatsVisible

显示附加的连接“健康”值。

- bool trafficStatsOn

显示额外的“低级别”流量统计。

- bool buttonsOn

显示控制和重置统计的按钮。

- Rect statsRect = new Rect(0, 100, 200, 50)

矩形窗口的位置。

- int WindowId = 100

Unity 的 GUI 窗口 ID（必须是唯一的，否则会出问题）。

8.26.1 Detailed Description | 详细描述

展示 Photon 连接的流量和健康统计的基础 GUI，通过快捷键 shift+tab 来开关。

展示的健康值可以帮助你通过连接丢失或性能来识别问题。例如：如果两个连续 SendOutgoingCommands 调用之间的时间增量大于等于 1 秒，那么由此造成断连的几率就会增加（因为到服务器的确认需要在适当的时间被发送）。

8.26.2 Member Function Documentation | 成员函数文档

8.26.2.1 void PhotonStatsGui.OnGUI ( )

8.26.2.2 void PhotonStatsGui.Start ( )

8.26.2.3 void PhotonStatsGui.TrafficStatsWindow (int windowID)

8.26.2.4 void PhotonStatsGui.Update ( )

检查 shift+tab 输入组合键（用来开关 statsOn）。

8.26.3 Member Data Documentation | 成员数据文档

8.26.3.1 bool PhotonStatsGui.buttonsOn

显示控制和重置统计的按钮。

8.26.3.2 bool PhotonStatsGui.healthStatsVisible

显示附加的连接“健康”值。

8.26.3.3 bool PhotonStatsGui.statsOn = true

开关搜集数据的选项（在 Update()里使用）。

8.26.3.4 Rect PhotonStatsGui.statsRect = new Rect(0, 100, 200, 50)

矩形窗口的位置。

8.26.3.5 bool PhotonStatsGui.statsWindowOn = true

展示或隐藏 GUI（不会影响统计数据是否被收集）。

8.26.3.6 bool PhotonStatsGui.trafficStatsOn

显示额外的“低级别”流量统计。

8.26.3.7 int PhotonStatsGui.WindowId = 100

Unity 的 GUI 窗口 ID（必须是唯一的，否则会出问题）。

8.27 PhotonStream Class Reference | 类参考

该容器被用于 OnPhotonSerializeView()中，用来要么提供 PhotonView 的输入数据，要么由你为其提供数据。

Public Member Functions | 公共成员函数

- PhotonStream (bool write, object[ ] incomingData)

创建一个流并初始化它。在 PUN 内部使用。

- void SetReadStream (object[ ] incomingData, byte pos=0)
- object ReceiveNext ()

当 isReading 为 true 的时从数据流读取下一块。

- object PeekNext ()

从数据流中读取下一个数据，而无需推进“当前”项。

- void SendNext (object obj)

当 isWriting 为 true 时添加另一个数据来发送。

- object[ ] ToArray ()

将数据流转化成一个新的 object[]对象数组。

- void Serialize (ref bool myBool)

将读取或写入值，根据数据流的 isWriting 值而定。

- void Serialize (ref int myInt)

将读取或写入值，根据数据流的 isWriting 值而定。

- void Serialize (ref string value)

将读取或写入值，根据数据流的 isWriting 值而定。

- void Serialize (ref char value)

将读取或写入值，根据数据流的 isWriting 值而定。

- void Serialize (ref short value)

将读取或写入值，根据数据流的 isWriting 值而定。

- void Serialize (ref float obj)

将读取或写入值，根据数据流的 isWriting 值而定。

- void Serialize (ref PhotonPlayer obj)

将读取或写入值，根据数据流的 isWriting 值而定。

- void Serialize (ref Vector3 obj)

将读取或写入值，根据数据流的 isWriting 值而定。

- void Serialize (ref Vector2 obj)

将读取或写入值，根据数据流的 isWriting 值而定。

- void Serialize (ref Quaternion obj)

将读取或写入值，根据数据流的 isWriting 值而定。

Properties | 属性

- bool isWriting [get]

如果为 true，客户端应该添加数据到数据流来将其发送出去。

- bool isReading [get]

如果为 true，客户端应该读取其他客户端发送的数据。

- int Count [get]

数据流里面的项目计数。

8.27.1 Detailed Description | 详细描述

该容器被用于 OnPhotonSerializeView()中，用来要么提供 PhotonView 的输入数据，要么由你为其提供数据。

如果该客户端是 PhotonView 的“拥有者”（这样的游戏对象），那么 isWriting 属性将是 true。添加数据到信息流，并且这些数据通过服务器被发送到房间里的其他玩家。在接收端这边，isWriting 是 false，并且数据应该被读取。

尽可能少的发送数据来保持连接质量。一个空的 PhotonStream 将不被发送。

可以使用 Serialize() 、SendNext()、ReceiveNext()方法来读写。后两个是显式的读写方法，但做的是和 Serialize()同样的工作。这是你使用哪种方法的偏好问题.。

See also | 参见

PhotonNetworkingMessage

8.27.2 Constructor & Destructor Documentation | 构造与析构文档

8.27.2.1 PhotonStream.PhotonStream (bool write, object[ ] incomingData)

创建一个流并初始化它。在 PUN 内部使用。

8.27.3 Member Function Documentation | 成员函数文档

8.27.3.1 object PhotonStream.PeekNext ( )

从数据流中读取下一个数据，而无需推进“当前”项。

8.27.3.2 object PhotonStream.ReceiveNext ( )

当 isReading 为 true 的时从数据流读取下一块。

8.27.3.3 void PhotonStream.SendNext (object obj)

当 isWriting 为 true 时添加另一个数据来发送。

8.27.3.4 void PhotonStream.Serialize (ref bool myBool)

将读取或写入值，根据数据流的 isWriting 值而定。

8.27.3.5 void PhotonStream.Serialize (ref int myInt)

将读取或写入值，根据数据流的 isWriting 值而定。

8.27.3.6 void PhotonStream.Serialize (ref string value)

将读取或写入值，根据数据流的 isWriting 值而定。

8.27.3.7 void PhotonStream.Serialize (ref char value)

将读取或写入值，根据数据流的 isWriting 值而定。

8.27.3.8 void PhotonStream.Serialize (ref short value)

将读取或写入值，根据数据流的 isWriting 值而定。

8.27.3.9 void PhotonStream.Serialize (ref float obj)

将读取或写入值，根据数据流的 isWriting 值而定。

8.27.3.10 void PhotonStream.Serialize (ref PhotonPlayer obj)

将读取或写入值，根据数据流的 isWriting 值而定。

8.27.3.11 void PhotonStream.Serialize (ref Vector3 obj)

将读取或写入值，根据数据流的 isWriting 值而定。

8.27.3.12 void PhotonStream.Serialize (ref Vector2 obj)

将读取或写入值，根据数据流的 isWriting 值而定。

8.27.3.13 void PhotonStream.Serialize (ref Quaternion obj)

将读取或写入值，根据数据流的 isWriting 值而定。

8.27.3.14 void PhotonStream.SetReadStream (object[ ] incomingData, byte pos = 0)

8.27.3.15 object [ ] PhotonStream.ToArray ( )

将数据流转化成一个新的 object[]对象数组。

8.27.4 Property Documentation | 属性文档

8.27.4.1 int PhotonStream.Count [get]

数据流里面的项目计数。

8.27.4.2 bool PhotonStream.isReading [get]

如果为 true，客户端应该读取其他客户端发送的数据。

8.27.4.3 bool PhotonStream.isWriting [get]

如果为 true，客户端应该添加数据到数据流来将其发送出去。

8.28 PhotonStreamQueue Class Reference | 类参考

PhotonStreamQueue 帮助你在更高的频率获取对象状态，然后是由 PhotonNetwork.sendRate 决定的频率，再然后当 Serialize()被调用时马上发送所有获取到的状态。在接收端，你可以调用 Deserialize()，然后数据流将以记录的与序列化相同的顺序和 timeStep 推出接收的对象状态。

Public Member Functions | 公共成员函数

- PhotonStreamQueue (int sampleRate)

初始化一个新的 PhotonStreamQueue 类实例。

- void Reset ()

重置 PhotonStreamQueue。你需要在你观察的对象数量改变时做这个。

- void SendNext (object obj)

添加下一个对象到队列。这个方法的工作原理和 PhotonStream.SendNext 是一样的。

- bool HasQueuedObjects ()

确定队列是否存储了任何对象。

- object ReceiveNext ()

从队列接收下一个对象。这个方法的工作原理和 PhotonStream.ReceiveNext 是一样的。

- void Serialize (PhotonStream stream)

序列化指定的数据流。在 OnPhotonSerializeView 方法里调用这个方法来发送整个记录的数据流。

- void Deserialize (PhotonStream stream)

解序列化指定的数据流。在 OnPhotonSerializeView 方法里调用这个方法来接收整个记录的数据流。

8.28.1 Detailed Description | 详细描述

PhotonStreamQueue 帮助你在更高的频率获取对象状态，然后是由 PhotonNetwork.sendRate 决定的频率，再然后当 Serialize()被调用时马上发送所有获取到的状态。在接收端，你可以调用 Deserialize()，然后数据流将以记录的与序列化相同的顺序和 timeStep 推出接收的对象状态。

8.28.2 Constructor & Destructor Documentation | 构造与析构文档

8.28.2.1 PhotonStreamQueue.PhotonStreamQueue (int sampleRate)

初始化一个新的 PhotonStreamQueue 类实例。

Parameters | 参数

sampleRate	每秒要采样多少次对象状态。

8.28.3 Member Function Documentation | 成员函数文档

8.28.3.1 void PhotonStreamQueue.Deserialize (PhotonStream stream)

解序列化指定的数据流。在 OnPhotonSerializeView 方法里调用这个方法来接收整个记录的数据流。

Parameters | 参数

stream	你收到的在 OnPhotonSerializeView 里的 PhotonStream 作为参数。

8.28.3.2 bool PhotonStreamQueue.HasQueuedObjects ( )

确定队列是否存储了任何对象.

8.28.3.3 object PhotonStreamQueue.ReceiveNext ( )

从队列接收下一个对象。这个方法的工作原理和 PhotonStream.ReceiveNext 是一样的。

8.28.3.4 void PhotonStreamQueue.Reset ( )

重置 PhotonStreamQueue。你需要在你观察的对象数量改变时做这个。

8.28.3.5 void PhotonStreamQueue.SendNext (object obj)

添加下一个对象到队列。这个方法的工作原理和 PhotonStream.SendNext 是一样的。

Parameters | 参数

obj	您要添加到队列中的对象。

8.28.3.6 void PhotonStreamQueue.Serialize (PhotonStream stream)

序列化指定的数据流。在 OnPhotonSerializeView 方法里调用这个方法来发送整个记录的数据流。

Parameters | 参数

stream	你收到的在 OnPhotonSerializeView 里的 PhotonStream 作为参数。

8.29 PhotonTransformView Class Reference | 参考

这个类帮助你同步游戏对象的 position、rotation 和 scale。它还给你许多不同的选择来使同步值显得平滑，即使当数据只是每秒发送几次。只需要简单地把该组件添加到你的游戏对象上，并确保 PhotonTransformView 被添加到观察组件的列表。

继承 MonoBehaviour 和 IPunObservable 接口。

Public Member Functions | 公共成员函数

- void SetSynchronizedValues (Vector3 speed, float turnSpeed)

如果插补模式或外推模式 SynchronizeValues 被使用，这些值会被同步到远程对象上。您的移动脚本应该传递当前速度（单位/秒）和旋转速度（角度/秒），以便远程对象可以使用它们来预测对象的运动。

- void OnPhotonSerializeView (PhotonStream stream, PhotonMessageInfo info)

被 PUN 每秒调用几次，以致于你的脚本可以为 PhotonView 读写同步数据。

8.29.1 Detailed Description | 详细描述

这个类帮助你同步游戏对象的 position、rotation 和 scale。它还给你许多不同的选择来使同步值显得平滑，即使当数据只是每秒发送几次。只需要简单地把该组件添加到你的游戏对象上，并确保 PhotonTransformView 被添加到观察组件的列表。

8.29.2 Member Function Documentation | 成员函数文档

8.29.2.1 void PhotonTransformView.OnPhotonSerializeView (PhotonStream stream, PhotonMessageInfo info)

被 PUN 每秒调用几次，以致于你的脚本可以为 PhotonView 读写同步数据。

这个方法将在那些被指派作为 PhotonView 的观察组件的脚本里调用。

PhotonNetwork.sendRateOnSerialize 影响该方法被调用的频率。

PhotonNetwork.sendRate 影响该客户端发包的频率。

实现该方法，你可以自定义哪些数据是 PhotonView 定时同步的。你的代码定义什么正在被发送（内容）和你的数据怎样被接收的客户端使用。

不像其他回调函数，OnPhotonSerializeView 只有在被指派到一个 PhotonView 做为 PhotonView.observed 脚本时被调用。

要利用这个方法，PhotonStream 是必须的。它将会在客户端上是“写入模式”，从而控制 PhotonView（PhotonStream.isWriting == true），而在远程客户端上是“读取模式”，只是接收控制客户端发送的数据。

如果你跳过写入任何值到二进制流中，PUN 将跳过更新。谨慎使用，这可以节省宽带和消息（每个房间/秒都会有一个限制）。

注意当发送者没有发送任何更新时，OnPhotonSerializeView 不会在远程客户端上被调用。这不能被用来作为"x-次每秒的 Update()"。

实现 IPunObservable 接口。

8.29.2.2 void PhotonTransformView.SetSynchronizedValues (Vector3 speed, float turnSpeed)

如果插补模式或外推模式 SynchronizeValues 被使用，这些值会被同步到远程对象上。您的移动脚本应该传递当前速度（单位/秒）和旋转速度（角度/秒），以便远程对象可以使用它

们来预测对象的运动。

Parameters | 参数

speed	对象当前的移动向量，单位/秒。
turnSpeed	对象当前的旋转速度，角度/秒。

8.30 PhotonTransformViewPositionControl Class Reference | 类参考

Public Member Functions | 公共成员函数

- PhotonTransformViewPositionControl (PhotonTransformViewPositionModel model)
- void SetSynchronizedValues (Vector3 speed, float turnSpeed)

如果插补模式或外推模式 SynchronizeValues 被使用，这些值会被同步到远程对象上。您的移动脚本应该传递当前速度（单位/秒）和旋转速度（角度/秒），以便远程对象可以使用它们来预测对象的运动。

- Vector3 UpdatePosition (Vector3 currentPosition)

基于设置在观察窗口里的值计算新的位置。

- Vector3 GetNetworkPosition ()

获取通过网络接收的最后一个位置.

- Vector3 GetExtrapolatedPositionOffset ()

根据最后一个同步位置、该同步位置接收的时间和对象的移动速度计算一个估计位置.

- void OnPhotonSerializeView (Vector3 currentPosition, PhotonStream stream, PhotonMessageInfo info)

8.30.1 Constructor & Destructor Documentation | 构造与析构文档

8.30.1.1 PhotonTransformViewPositionControl.PhotonTransformViewPositionControl (PhotonTransformViewPositionModel model)

8.30.2 Member Function Documentation | 成员函数文档

8.30.2.1 Vector3 PhotonTransformViewPositionControl.GetExtrapolatedPositionOffset ( )

根据最后一个同步位置、该同步位置接收的时间和对象的移动速度计算一个估计位置.

Returns | 返回值

远程对象的估计位置。

8.30.2.2 Vector3 PhotonTransformViewPositionControl.GetNetworkPosition ( )

获取通过网络接收的最后一个位置.

8.30.2.3 void PhotonTransformViewPositionControl.OnPhotonSerializeView (Vector3 currentPosition, PhotonStream stream, PhotonMessageInfo info)

8.30.2.4 void PhotonTransformViewPositionControl.SetSynchronizedValues (Vector3 speed, float turnSpeed)

如果插补模式或外推模式 SynchronizeValues 被使用，这些值会被同步到远程对象上。您的移动脚本应该传递当前速度（单位/秒）和旋转速度（角度/秒），以便远程对象可以使用它们来预测对象的运动。

Parameters | 参数

speed	对象当前的移动向量，单位/秒。
turnSpeed	对象当前的旋转速度，角度/秒。

8.30.2.5 Vector3 PhotonTransformViewPositionControl.UpdatePosition (Vector3 currentPosition)

基于在观察窗口里设置的值计算新的位置。

Parameters | 参数

currentPosition	当前的位置。

Returns | 返回值

返回新的位置。

8.31 PhotonTransformViewPositionModel Class Reference | 类参考

Public Types | 公共类型

- enum InterpolateOptions {

InterpolateOptions.Disabled, InterpolateOptions.FixedSpeed,

InterpolateOptions.EstimatedSpeed, Interpolate -

Options.SynchronizeValues,

InterpolateOptions.Lerp }

- enum ExtrapolateOptions { ExtrapolateOptions.Disabled,

ExtrapolateOptions.SynchronizeValues,

ExtrapolateOptions.EstimateSpeedAndTurn,

ExtrapolateOptions.FixedSpeed }

Public Attributes | 公共字段

- bool SynchronizeEnabled
- bool TeleportEnabled = true
- float TeleportIfDistanceGreaterThan = 3f
- InterpolateOptions InterpolateOption = InterpolateOptions.EstimatedSpeed
- float InterpolateMoveTowardsSpeed = 1f
- float InterpolateLerpSpeed = 1f
- float InterpolateMoveTowardsAcceleration = 2
- float InterpolateMoveTowardsDeceleration = 2
- AnimationCurve InterpolateSpeedCurve
- ExtrapolateOptions ExtrapolateOption = ExtrapolateOptions.Disabled
- float ExtrapolateSpeed = 1f
- bool ExtrapolateIncludingRoundTripTime = true
- int ExtrapolateNumberOfStoredPositions = 1
- bool DrawErrorGizmo = true

8.31.1 Member Enumeration Documentation | 成员枚举文档

8.31.1.1 enum PhotonTransformViewPositionModel.ExtrapolateOptions

Enumerator | 枚举器

Disabled | 禁用

SynchronizeValues | 同步值

EstimateSpeedAndTurn | 估算速度和旋转度

FixedSpeed | 固定速度

8.31.1.2 enum PhotonTransformViewPositionModel.InterpolateOptions

Enumerator | 枚举器

Disabled | 禁用

FixedSpeed | 固定速度

EstimatedSpeed | 估算速度

SynchronizeValues | 同步值

Lerp | 插值

8.31.2 Member Data Documentation | 成员数据文档

8.31.2.1 bool PhotonTransformViewPositionModel.DrawErrorGizmo = true

8.31.2.2 bool PhotonTransformViewPositionModel.ExtrapolateIncludingRoundTripTime = true

8.31.2.3 int PhotonTransformViewPositionModel.ExtrapolateNumberOfStoredPositions = 1

8.31.2.4 ExtrapolateOptions

PhotonTransformViewPositionModel.ExtrapolateOption =

ExtrapolateOptions.Disabled

8.31.2.5 float PhotonTransformViewPositionModel.ExtrapolateSpeed = 1f

8.31.2.6 float PhotonTransformViewPositionModel.InterpolateLerpSpeed = 1f

8.31.2.7 float PhotonTransformViewPositionModel.InterpolateMoveTowardsAcceleration = 2

8.31.2.8 float PhotonTransformViewPositionModel.InterpolateMoveTowardsDeceleration = 2

8.31.2.9 float PhotonTransformViewPositionModel.InterpolateMoveTowardsSpeed = 1f

8.31.2.10 InterpolateOptions PhotonTransformViewPositionModel.InterpolateOption = InterpolateOptions.EstimatedSpeed

8.31.2.11 AnimationCurve PhotonTransformViewPositionModel.InterpolateSpeedCurve

初始值:

```
= new AnimationCurve( new Keyframe[] {new Keyframe( -1, 0, 0, Mathf.Infinity ),
new Keyframe( 0, 1, 0, 0 ),
new Keyframe( 1, 1, 0, 1 ),
new Keyframe( 4, 4, 1, 0 ) })
```

8.31.2.12 bool PhotonTransformViewPositionModel.SynchronizeEnabled

8.31.2.13 bool PhotonTransformViewPositionModel.TeleportEnabled = true

8.31.2.14 float PhotonTransformViewPositionModel.TeleportIfDistanceGreaterThan = 3f

8.32 PhotonTransformViewRotationControl Class Reference | 类参考

Public Member Functions | 公共成员函数

- PhotonTransformViewRotationControl (PhotonTransformViewRotationModel model)
- Quaternion GetNetworkRotation ()

  获取通过网络接收的最后一个旋转。
- Quaternion GetRotation (Quaternion currentRotation)
- void OnPhotonSerializeView (Quaternion currentRotation, PhotonStream stream, PhotonMessageInfo info)

8.32.1 Constructor & Destructor Documentation | 构造与析构文档

8.32.1.1 PhotonTransformViewRotationControl.PhotonTransformViewRotationControl (PhotonTransformViewRotationModel model)

8.32.2 Member Function Documentation | 成员函数文档

8.32.2.1 Quaternion PhotonTransformViewRotationControl.GetNetworkRotation ( )

获取通过网络接收的最后一个旋转。

8.32.2.2 Quaternion PhotonTransformViewRotationControl.GetRotation (Quaternion currentRotation)

8.32.2.3 void PhotonTransformViewRotationControl.OnPhotonSerializeView (Quaternion currentRotation, PhotonStream stream, PhotonMessageInfo info)

8.33 PhotonTransformViewRotationModel Class Reference | 类参考

Public Types | 公共类型

- enum InterpolateOptions { InterpolateOptions.Disabled, InterpolateOptions.RotateTowards, InterpolateOptions.Lerp }

Public Attributes | 公共字段

- bool SynchronizeEnabled
- InterpolateOptions InterpolateOption = InterpolateOptions.RotateTowards
- float InterpolateRotateTowardsSpeed = 180
- float InterpolateLerpSpeed = 5

8.33.1 Member Enumeration Documentation | 成员枚举文档

8.33.1.1 enum PhotonTransformViewRotationModel.InterpolateOptions

Enumerator | 枚举器

Disabled | 禁用

RotateTowards | 往一个方向旋转

Lerp | 插值

8.33.2 Member Data Documentation | 成员数据文档

8.33.2.1 float PhotonTransformViewRotationModel.InterpolateLerpSpeed = 5

8.33.2.2 InterpolateOptions PhotonTransformViewRotationModel.InterpolateOption = InterpolateOptions.RotateTowards

8.33.2.3 float PhotonTransformViewRotationModel.InterpolateRotateTowardsSpeed = 180

8.33.2.4 bool PhotonTransformViewRotationModel.SynchronizeEnabled

8.34 PhotonTransformViewScaleControl Class Reference | 类参考

Public Member Functions | 公共成员函数

- PhotonTransformViewScaleControl (PhotonTransformViewScaleModel model)
- Vector3 GetNetworkScale ()

获取通过网络接收的最后一个 Scale.

- Vector3 GetScale (Vector3 currentScale)
- void OnPhotonSerializeView (Vector3 currentScale, PhotonStream stream, PhotonMessageInfo info)

8.34.1 Constructor & Destructor Documentation | 构造与析构文档

8.34.1.1 PhotonTransformViewScaleControl.PhotonTransformViewScaleControl (PhotonTransformViewScaleModel model)

8.34.2 Member Function Documentation | 成员函数文档

8.34.2.1 Vector3 PhotonTransformViewScaleControl.GetNetworkScale ( )

获取通过网络接收的最后一个 Scale.

8.34.2.2 Vector3 PhotonTransformViewScaleControl.GetScale (Vector3 currentScale)

8.34.2.3 void PhotonTransformViewScaleControl.OnPhotonSerializeView (Vector3 currentScale, PhotonStream stream,PhotonMessageInfo info)

8.35 PhotonTransformViewScaleModel Class Reference | 类参考

Public Types | 公共类型

- enum InterpolateOptions { InterpolateOptions.Disabled, InterpolateOptions.MoveTowards, InterpolateOptions.Lerp }

Public Attributes | 公共字段

- bool SynchronizeEnabled
- InterpolateOptions InterpolateOption = InterpolateOptions.Disabled
- float InterpolateMoveTowardsSpeed = 1f
- float InterpolateLerpSpeed

8.35.1 Member Enumeration Documentation | 成员枚举文档

8.35.1.1 enum PhotonTransformViewScaleModel.InterpolateOptions

Enumerator | 枚举器

Disabled | 禁用

MoveTowards | 朝向移动

Lerp | 插值

8.35.2 Member Data Documentation | 成员数据文档

8.35.2.1 float PhotonTransformViewScaleModel.InterpolateLerpSpeed

8.35.2.2 float PhotonTransformViewScaleModel.InterpolateMoveTowardsSpeed = 1f

8.35.2.3 InterpolateOptions PhotonTransformViewScaleModel.InterpolateOption = InterpolateOptions.Disabled

8.35.2.4 bool PhotonTransformViewScaleModel.SynchronizeEnabled

8.36 PhotonView Class Reference | 类参考

PUN 的网络 NetworkView 替代类。和 NetworkView 一样的使用方法。

继承 Photon.MonoBehaviour.

Public Member Functions | 公共成员函数

- void RequestOwnership ()

根据 PhotonView 的 ownershipTransfer 设置，任何客户端可以请求成为 PhotonView 的拥有者。

- void TransferOwnership (PhotonPlayer newOwner)

转移这个 PhotonView（和游戏对象）的所有权到另外一个玩家。

- void TransferOwnership (int newOwnerId)

转移这个 PhotonView（和游戏对象）的所有权到另外一个玩家。

- void SerializeView (PhotonStream stream, PhotonMessageInfo info)
- void DeserializeView (PhotonStream stream, PhotonMessageInfo info)
- void RefreshRpcMonoBehaviourCache ()

可以被用来在 PhotonNetwork.UseRpcMonoBehaviourCache 为 true 时刷新游戏对象上的 MonoBehaviours 的列表。

- void RPC (string methodName, PhotonTargets target, params object[ ] parameters)

在这个房间的远程客户端上调用这个游戏对象的 RPC 方法（或者在所有客户端上都调用，包括这个客户端）。

- void RpcSecure (string methodName, PhotonTargets target, bool encrypt, params object[ ] parameters)

在这个房间的远程客户端上调用这个游戏对象的 RPC 方法（或者在所有客户端上都调用，包括这个客户端）。

- void RPC (string methodName, PhotonPlayer targetPlayer, params object[ ] parameters)

在这个房间的远程客户端上调用这个游戏对象的 RPC 方法（或者在所有客户端上都调用，包括这个客户端）。

- void RpcSecure (string methodName, PhotonPlayer targetPlayer, bool encrypt, params object[ ] parameters)

在这个房间的远程客户端上调用这个游戏对象的 RPC 方法（或者在所有客户端上都调用，包括这个客户端）。

- override string ToString ()

Static Public Member Functions | 静态公共成员函数

- static PhotonView Get (Component component)
- static PhotonView Get (GameObject gameObj)
- static PhotonView Find (int viewID)

Public Attributes | 公共字段

- int ownerId
- int group = 0
- bool OwnerShipWasTransfered

用于检查该 photonView 的所有权是否在生命周期内被设置的布尔值。当加入晚时用来检查事件是否不匹配拥有者，以及发送者是否需要地址。

- int prefixBackup = -1
- Component observed
- ViewSynchronization synchronization
- OnSerializeTransform onSerializeTransformOption = OnSerializeTransform.PositionAndRotation
- OnSerializeRigidBody onSerializeRigidBodyOption = OnSerializeRigidBody.All
- OwnershipOption ownershipTransfer = OwnershipOption.Fixed

定义这个 PhotonView 的所有权是否是固定的，可以被请求或简单地获取。

- List< Component > ObservedComponents
- int instantiationId

Properties | 属性

- int prefix [get, set]
- object[ ] instantiationData [get, set]

这个就是在调用 PhotonNetwork.Instantiate（如果是被用来生成这个预制体）时被传递的 instantiationData。

- int viewID [get, set]

PhotonView 的 ID。在网络游戏中识别它（每个房间内）。

- bool isSceneView [get]

如果 PhotonView 和场景（游戏对象）一起被加载或用 InstantiateSceneObject 方法实例化，则为 true。

- PhotonPlayer owner [get]

PhotonView 的所有者是创建视觉游戏对象的玩家。场景中的对象则没有拥有者。

- int OwnerActorNr [get]
- bool isOwnerActive [get]
- int CreatorActorNr [get]
- bool isMine [get]

如果该 PhotonView 是“我的”且可以被这个客户端控制，则为 true。

8.36.1 Detailed Description | 详细描述

PUN 的网络 NetworkView 替代类。和 NetworkView 一样的使用方法。

8.36.2 Member Function Documentation | 成员函数文档

8.36.2.1 void PhotonView.DeserializeView (PhotonStream stream, PhotonMessageInfo info)

8.36.2.2 static PhotonView PhotonView.Find (int viewID) [static]

8.36.2.3 static PhotonView PhotonView.Get (Component component) [static]

8.36.2.4 static PhotonView PhotonView.Get (GameObject gameObj) [static]

8.36.2.5 void PhotonView.RefreshRpcMonoBehaviourCache ( )

可以被用来在 PhotonNetwork.UseRpcMonoBehaviourCache 为 true 时刷新游戏对象上的 MonoBehaviours 的列表。

设置 PhotonNetwork.UseRpcMonoBehaviourCache 为 true 来启用缓冲。使用 this.GetComponents<MonoBehaviour>()来获取 MonoBehaviours 的列表 ,用于调用 RPCs（潜在地）。

当 PhotonNetwork.UseRpcMonoBehaviourCache 为 false 时，这个方法没有效果，因为该列表在 RPC 调用时被刷新。

8.36.2.6 void PhotonView.RequestOwnership ( )

根据 PhotonView 的 ownershipTransfer 设置，任何客户端可以请求成为 PhotonView 的拥有者。

如果 ownershipTransfer 设置允许，请求所有权可以让你控制 PhotonView。当前的所有者可能不得不实现 IPunCallbacks.OnOwnershipRequest 回调函数来回应所有权请求。

PhotonView 的所有者/控制者也是发送游戏对象位置更新的客户端。

8.36.2.7 void PhotonView.RPC (string methodName, PhotonTargets target, params object[ ] parameters)

在这个房间的远程客户端上调用这个游戏对象的 RPC 方法（或者在所有客户端上都调用，包括这个客户端）。

Remote Procedure Calls 是一个 PUN 制作多人游戏的必备工具。它使你可以让同房间内的每个客户端可以调用一个指定的方法。

RPC 调用可以以“所有的”或“其他的”客户端为目标。通常，以“所有的”的客户端为目标会在发送 RPC 后在本地立即执行。“ViaServer”选项会将 RPC 发送到服务器，并在被发送回来的时候在这个客户端上执行。当然，调用会受客户端的延迟所影响。

每一个调用自动追踪到同一个被用于始发客户端的 PhotonView（以及游戏对象）。

详见：Remote Procedure Calls。

Parameters | 参数

methodName	有 RPC 字段的适合的方法的名称。
target	目标组以及 RPC 发送的方式。
parameters	RPC 方法的参数（必须匹配这个调用！）。

8.36.2.8 void PhotonView.RPC (string methodName, PhotonPlayer targetPlayer, params object[ ] parameters)

在这个房间的远程客户端上调用这个游戏对象的 RPC 方法（或者在所有客户端上都调用，包括这个客户端）。

Remote Procedure Calls 是一个 PUN 制作多人游戏的必备工具。它使你可以让同房间内的每个客户端可以调用一个指定的方法。

该方法允许你使 RPC 在指定玩家的客户端上调用。当然，调用会受这个客户端和远程客户端的延迟影响。

每一个调用自动追踪到同一个被用于始发客户端的 PhotonView（以及游戏对象）。

详见：Remote Procedure Calls。

Parameters | 参数

methodName	有 RPC 字段的适合的方法的名称。
targetPlayer	目标组以及 RPC 发送的方式。
parameters	RPC 方法的参数（必须匹配这个调用！）。

8.36.2.9 void PhotonView.RpcSecure (string methodName, PhotonTargets target, bool encrypt, params object[ ] parameters)

在这个房间的远程客户端上调用这个游戏对象的 RPC 方法（或者在所有客户端上都调用，包括这个客户端）。

Remote Procedure Calls 是一个 PUN 制作多人游戏的必备工具。它使你可以让同房间内的每个客户端可以调用一个指定的方法。

RPC 调用可以以“所有的”或“其他的”客户端为目标。通常，以“所有的”的客户端为目标会在发送 RPC 后在本地立即执行。“ViaServer”选项会将 RPC 发送到服务器，并在被发送回来的时候在这个客户端上执行。当然，调用会受客户端的延迟所影响。

每一个调用自动追踪到同一个被用于始发客户端的 PhotonView（以及游戏对象）。

详见：Remote Procedure Calls。

Parameters | 参数

methodName	有 RPC 字段的适合的方法的名称。
target	目标组以及 RPC 发送的方式。
parameters	RPC 方法的参数（必须匹配这个调用！）。

8.36.2.10 void PhotonView.RpcSecure (string methodName, PhotonPlayer targetPlayer, bool encrypt, params object[ ] parameters)

在这个房间的远程客户端上调用这个游戏对象的 RPC 方法（或者在所有客户端上都调用，包括这个客户端）。

Remote Procedure Calls 是一个 PUN 制作多人游戏的必备工具。它使你可以让同房间内的每个客户端可以调用一个指定的方法。

RPC 调用可以以“所有的”或“其他的”客户端为目标。通常，以“所有的”的客户端为目标会在发送 RPC 后在本地立即执行。“ViaServer”选项会将 RPC 发送到服务器，并在被发送回来的时候在这个客户端上执行。当然，调用会受客户端的延迟所影响。

每一个调用自动追踪到同一个被用于始发客户端的 PhotonView（以及游戏对象）。

详见：Remote Procedure Calls。

Parameters | 参数

methodName	有 RPC 字段的适合的方法的名称。
target	目标组以及 RPC 发送的方式。
parameters	RPC 方法的参数（必须匹配这个调用！）。

8.36.2.11 void PhotonView.SerializeView (PhotonStream stream, PhotonMessageInfo info)

8.36.2.12 override string PhotonView.ToString ( )

8.36.2.13 void PhotonView.TransferOwnership (PhotonPlayer newOwner)

转移这个 PhotonView（和游戏对象）的所有权到另外一个玩家。

PhotonView 的所有者/控制者也是发送游戏对象位置更新的客户端。

8.36.2.14 void PhotonView.TransferOwnership (int newOwnerId)

转移这个 PhotonView（和游戏对象）的所有权到另外一个玩家。

PhotonView 的所有者/控制者也是发送游戏对象位置更新的客户端。

8.36.3 Member Data Documentation | 成员数据文档

8.36.3.1 int PhotonView.group = 0

8.36.3.2 int PhotonView.instantiationId

8.36.3.3 Component PhotonView.observed

8.36.3.4 List<Component> PhotonView.ObservedComponents

8.36.3.5 OnSerializeRigidBody PhotonView.onSerializeRigidBodyOption = OnSerializeRigidBody.All

8.36.3.6 OnSerializeTransform PhotonView.onSerializeTransformOption = OnSerializeTransform.PositionAndRotation

8.36.3.7 int PhotonView.ownerId

8.36.3.8 OwnershipOption PhotonView.ownershipTransfer = OwnershipOption.Fixed

定义这个 PhotonView 的所有权是否是固定的，可以被请求或简单地获取。

注意，您不能在运行时编辑此值.。选项是在枚举 OwnershipOption 描述。当前的所有者

不得不实现 IPunCallbacks.OnOwnershipRequest 回调函数来对所有权的请求作出反应。

8.36.3.9 bool PhotonView.OwnerShipWasTransfered

用于检查该 photonView 的所有权是否在生命周期内被设置的布尔值。当加入晚时用来检查事件是否不匹配拥有者，以及发送者是否需要地址。

如果所有权已被转移则为 true；否则为 false。

8.36.3.10 int PhotonView.prefixBackup = -1

8.36.3.11 ViewSynchronization PhotonView.synchronization

8.36.4 Property Documentation | 属性文档

8.36.4.1 int PhotonView.CreatorActorNr [get]

8.36.4.2 object [ ] PhotonView.instantiationData [get], [set]

这个就是在调用 PhotonNetwork.Instantiate（如果是被用来生成这个预制体）时被传递的 instantiationData。

8.36.4.3 bool PhotonView.isMine [get]

如果该 PhotonView 是“我的”且可以被这个客户端控制，则为 true。

PUN 有一个定义谁可以控制和摧毁每一个 PhotonView 的所有权概念。如果所有者和本地的 PhotonPlayer 匹配则为 true。如果这是一个在主客户端上的场景 photonview 也为 true。

8.36.4.4 bool PhotonView.isOwnerActive [get]

8.36.4.5 bool PhotonView.isSceneView [get]

如果 PhotonView 和场景（游戏对象）一起被加载或用 InstantiateSceneObject 方法实例化，则为 true。

场景对象不被某个特别的玩家所拥有，而是属于场景。因此当它们的创造者离开游戏时它们不会被销毁，并且当前的主客户端可以控制它们（无论是什么）。ownerId 是 0（玩家 IDs 是 1 以上的整型）。

8.36.4.6 PhotonPlayer PhotonView.owner [get]

PhotonView 的所有者是创造该视图游戏对象的玩家。场景里的对象没有拥有者。

PhotonView 的拥有者/控制者也是发送游戏对象位置更新的客户端。

所有权可以被转移到另一个拥有 PhotonView.TransferOwnership 的玩家，或任何可以通过调用 PhotonView 的 RequestOwnership 方法请求所有权的玩家。当前的所有者不得不实现 IPunCallbacks.OnOwnershipRequest 回调函数来对所有权的请求作出反应。

8.36.4.7 int PhotonView.OwnerActorNr [get]

8.36.4.8 int PhotonView.prefix [get], [set]

8.36.4.9 int PhotonView.viewID [get], [set]

PhotonView 的 ID。在网络游戏中识别它（每个房间内）。

8.37 PingMonoEditor Class Reference | 类参考

使用来自 System.Net.Sockets 的 C# Socket 类 (Unity 通常做的那样).

继承 PhotonPing.

Public Member Functions | 公共成员函数

- override bool StartPing (string ip)

发送一个"Photon Ping"到服务器。

- override bool Done ()
- override void Dispose ()

8.37.1 Detailed Description | 详细描述

使用来自 System.Net.Sockets 的 C# Socket 类 (Unity 通常做的那样).

与 Windows 8 Store/Phone API 不兼容。

8.37.2 Member Function Documentation | 成员函数文档

8.37.2.1 override void PingMonoEditor.Dispose ( )

8.37.2.2 override bool PingMonoEditor.Done ( )

8.37.2.3 override bool PingMonoEditor.StartPing (string ip)

发送一个"Photon Ping"到服务器。

Parameters | 参数

ip	以 IPv4 或 IPv6 协议的地址格式。一个包含一个'.'的地址将被解释为 IPv4。

Returns | 返回值

如果 Photon Ping 可以被发送则为 true。

8.38 Photon.PunBehaviour Class Reference | 类参考

这个类提供了一个.photonView 属性和所有 PUN 能调用的回调函数/事件。重写那些你想要使用的时间/方法。

继承 Photon.MonoBehaviour 和 IPunCallbacks 接口。

Public Member Functions | 公共成员函数

- virtual void OnConnectedToPhoton ()

当初始连接已被建立时调用，但在您可以使用服务器之前。当 PUN 准备好时调用 OnJoinedLobby()或 OnConnectedToMaster()。

- virtual void OnLeftRoom ()

当本地用户/客户端离开房间时调用。

- virtual void OnMasterClientSwitched (PhotonPlayer newMasterClient)

在当前主客户端离开时切换到一个新主客户端，切换后调用。

- virtual void OnPhotonCreateRoomFailed (object[ ] codeAndMsg)

当 CreateRoom()调用失败时被调用。参数提供 ErrorCode 和消息（作为数组）。

- virtual void OnPhotonJoinRoomFailed (object[ ] codeAndMsg)

当 JoinRoom()调用失败时被调用。参数提供 ErrorCode 和消息（作为数组）。

- virtual void OnCreatedRoom ()

当这个客户端创建一个房间并进入该房间时被调用。OnJoinedRoom()将也会被调用。

- virtual void OnJoinedLobby ()

在主服务器上进入大厅时被调用。实际的房间列表更新将调用 OnReceivedRoomListUpdate()。

- virtual void OnLeftLobby ()

离开大厅后被调用。

- virtual void OnFailedToConnectToPhoton (DisconnectCause cause)

如果一个连接到 Photon 服务器请求失败（在连接被建立之前）被调用，接着会调用

OnDisconnectedFromPhoton()。

- virtual void OnDisconnectedFromPhoton ()

在从 Photon 服务器断连后被调用。

- virtual void OnConnectionFail (DisconnectCause cause)

当未知因素导致连接失败（在建立连接之后）时被调用，接着调用 OnDisconnectedFromPhoton()。

- virtual void OnPhotonInstantiate (PhotonMessageInfo info)

在一个游戏对象（及其子类）通过使用 PhotonNetwork.Instantiate 方法被实例化时在任何脚本上被调用。

- virtual void OnReceivedRoomListUpdate ()

在主服务器上的大厅内（PhotonNetwork.insideLobby）房间列表的任何更新都会调用该函数。

- virtual void OnJoinedRoom ()

当进入一个房间（通过创建或加入）时被调用。在所有客户端（包括主客户端）上被调用。

- virtual void OnPhotonPlayerConnected (PhotonPlayer newPlayer)

当一个远程玩家进入房间时调用。这个 PhotonPlayer 在这个时候已经被添加 playerlist 玩家列表。

- virtual void OnPhotonPlayerDisconnected (PhotonPlayer otherPlayer)

当一个远程玩家离开房间时调用。这个 PhotonPlayer 此时已经从 playerlist 玩家列表删除。

- virtual void OnPhotonRandomJoinFailed (object[ ] codeAndMsg)

在一个 JoinRandom()请求失败后调用。参数提供 ErrorCode 错误代码和消息。

- virtual void OnConnectedToMaster ()

在到主服务器连接被建立和认证后调用，但是只有当 PhotonNetwork.autoJoinLobby 是 false 时才调用。

- virtual void OnPhotonMaxCccuReached ()

由于并发用户限制（暂时）到达了，该客户端会被服务器拒绝并断连。

- virtual void OnPhotonCustomRoomPropertiesChanged (Hashtable propertiesThatChanged)

当一个房间的自定义属性更改时被调用。propertiesThatChanged 改变的属性包含所有通过 Room.SetCustomProperties 设置的。

- virtual void OnPhotonPlayerPropertiesChanged (object[ ] playerAndUpdatedProps)

当自定义玩家属性更改时调用。玩家和更改的属性被传递为对象数组 object[]。

- virtual void OnUpdatedFriendList ()

当服务器发送一个对 FindFriends 请求的响应并更新 PhotonNetwork.Friends 时调用。

- virtual void OnCustomAuthenticationFailed (string debugMessage)

当自定义身份验证失败时调用。接下来就会断开连接！

- virtual void OnCustomAuthenticationResponse (Dictionary< string, object > data)

当您的自定义身份验证服务用附加数据响应时调用。

- virtual void OnWebRpcResponse (OperationResponse response)

当一个 WebRPC 的回应可用时被 PUN 调用。详见 PhotonNetwork.WebRPC。

- virtual void OnOwnershipRequest (object[ ] viewAndPlayer)

当另一个玩家从你（现在的所有者）这里请求一个 PhotonView 的所有权时调用。

- virtual void OnLobbyStatisticsUpdate ()

当主服务器发送一个游戏大厅统计更新、更新 PhotonNetwork.LobbyStatistics 时调用。

Additional Inherited Members | 附加继承成员

8.38.1 Detailed Description | 详细描述

这个类提供了一个.photonView 属性和所有 PUN 能调用的回调函数/事件。重写那些你想要使用的时间/方法。

通过扩展这个类，你可以通过重写来实现个人的方法。

Visual Studio和MonoDevelop会在你输入"override"时提供一系列可以重写的方法。**你的实现不一定要调用"base.method()"。**

这个类实现了 IPunCallbacks，用于定义所有的 PUN 回调函数。不要在你的类里面实现 IPunCallbacks。而是实现 PunBehaviour 或自己的方法。

8.38.2 Member Function Documentation | 成员函数文档

8.38.2.1 virtual void Photon.PunBehaviour.OnConnectedToMaster ( ) [virtual]

在到主服务器连接被建立和认证后调用，但是只有当 PhotonNetwork.autoJoinLobby 是 false 时才调用。

如果你设置 PhotonNetwork.autoJoinLobby 为 true，取而代之调用的是 OnJoinedLobby()。

即使没有在游戏大厅内，你也可以加入房间和创建房间。在这种情况下使用了默认的大厅。可用房间列表将不可用，除非你通过 PhotonNetwork.joinLobby 加入一个游戏大厅。

实现了 IPunCallbacks.

8.38.2.2 virtual void Photon.PunBehaviour.OnConnectedToPhoton ( ) [virtual]

当初始连接已被建立时调用，但在您可以使用服务器之前。当 PUN 准备好时调用 OnJoinedLobby()或 OnConnectedToMaster()。

这个回调函数只在检测服务器是否可以被完全连接时有用（技术上）。通常，实现 OnFailedToConnectToPhoton()和 OnDisconnectedFromPhoton()就够了。

当 PUN 准备好时调用 OnJoinedLobby()或 OnConnectedToMaster()。当被调用时，低层次的连接被建立起来并且 PUN 会在后台发送你的 AppId，用户信息等。从主服务器向游戏服务器转换时不会被调用。

实现了 IPunCallbacks.

8.38.2.3 virtual void Photon.PunBehaviour.OnConnectionFail (DisconnectCause cause) [virtual]

当未知因素导致连接失败（在建立连接之后）时被调用，接着调用 OnDisconnectedFromPhoton()。

如果服务器不能立即连接，就会调用 OnFailedToConnectToPhoton。错误的原因会以 StatusCode 的形式提供。

实现了 IPunCallbacks.

8.38.2.4 virtual void Photon.PunBehaviour.OnCreatedRoom ( ) [virtual]

当这个客户端创建了一个房间并进入它时调用。OnJoinedRoom()也会被调用。这个回调只在创建房间的客户端调用（详见 PhotonNetwork.CreateRoom）。

由于任何客户端在任何时候都可能会关闭（或断开连接），一个房间的创造者有一定的几率不执行 OnCreatedRoom。

如果你需要特定的房间属性或一个“开始信号”，实现 OnMasterClientSwitched()并使新的主客户端检查房间的状态更加安全。

案例: void OnCreatedRoom() { ... }

实现了 IPunCallbacks.

8.38.2.5 virtual void Photon.PunBehaviour.OnCustomAuthenticationFailed (string debugMessage) [virtual]

当自定义身份验证失败时调用。接下来就会断开连接！

自定义身份验证可能会失败，由于用户输入、坏令牌/密匙。如果身份验证是成功的，这种方法不被调用。实现 OnJoinedLobby()或 OnConnectedToMaster()（像往常一样）。

在游戏开发过程中，它也可能由于服务器端的配置错误而失败。在这些情况下，记录 debugMessage 调试消息是非常重要的。

除非您为应用程序（在仪表板）设置一个自定义的身份验证服务，否则将不会被调用！

Parameters | 参数

debugMessage	包含一个授权失败原因的调试信息。这个必须要在开发期间被修复。 实现了 IPunCallbacks.

8.38.2.6 virtual void Photon.PunBehaviour.OnCustomAuthenticationResponse (Dictionary< string, object > data) [virtual]

当您的自定义身份验证服务用附加数据响应时调用。

自定义身份验证服务可以在响应中包含一些自定义数据。当存在时，该数据是在回调函数中作为字典使其可用。而你的数据的键值必须是字符串，值类型可以是字符串或数字（以 JSON 形式）。你需要额外确定，这个值类型是你期望的。数字成为（目前）Int64。

例子: void OnCustomAuthenticationResponse(Dictionary<string, object> data) { ... }

实现了 IPunCallbacks.

8.38.2.7 virtual void Photon.PunBehaviour.OnDisconnectedFromPhoton ( ) [virtual]

从 Photon 服务器断开连接后被调用。

在某些情况下，其他回调函数在 OnDisconnectedFromPhoton 被调用之前被调用。例如：OnConnectionFail()和 OnFailedToConnectToPhoton()。

示例: void OnDisconnectedFromPhoton() { ... }

实现了 IPunCallbacks.

8.38.2.8 virtual void Photon.PunBehaviour.OnFailedToConnectToPhoton (DisconnectCause cause) [virtual]

如果一个连接到 Photon 服务器请求失败（在连接被建立之前）被调用，接着会调用 OnDisconnectedFromPhoton()。

当完全没有连接可以被建立时这个回调函数被调用。和 OnConnectionFail 不同的是 OnConnectionFail 是在一个存在的连接失败时被调用的。

实现了 IPunCallbacks.

8.38.2.9 virtual void Photon.PunBehaviour.OnJoinedLobby ( ) [virtual]

在主服务器上进入一个大厅时调用。实际的房间列表的更新会调用 OnReceivedRoomListUpdate()。注意：当 PhotonNetwork.autoJoinLobby 是 false 时，OnConnectedToMaster()将会被调用并且房间列表将不可用。

而在大堂的房间列表是在固定的时间间隔内自动更新（这是你不能修改的）。当 OnReceivedRoomListUpdate()在 OnJoinedLobby()之后被调用后，房间列表变得可用。

例子: void OnJoinedLobby() { ... }

实现了 IPunCallbacks.

8.38.2.10 virtual void Photon.PunBehaviour.OnJoinedRoom ( ) [virtual]

当进入一个房间（通过创建或加入）时被调用。在所有客户端（包括主客户端）上被调用。

这种方法通常用于实例化玩家角色。如果一场比赛必须“积极地”被开始，你也可以调用一个由用户的按键或定时器触发的 PunRPC 。

当这个被调用时，你通常可以通过 PhotonNetwork.playerList 访问在房间里现有的玩家。

同时，所有自定义属性 Room.customProperties 应该已经可用。检查 Room.playerCount 就知道房间里是否有足够的玩家来开始游戏。

实现了 IPunCallbacks.

8.38.2.11 virtual void Photon.PunBehaviour.OnLeftLobby ( ) [virtual]

离开大厅后被调用。

当你离开大厅时，CreateRoom 和 JoinRandomRoom 自动引用默认的大堂。

例如: void OnLeftLobby() { ... }

实现了 IPunCallbacks.

8.38.2.12 virtual void Photon.PunBehaviour.OnLeftRoom ( ) [virtual]

当本地用户/客户离开房间时被调用。

当离开一个房间时，PUN 将你带回主服务器。在您可以使用游戏大厅和创建/加入房间之前，OnJoinedLobby()或 OnConnectedToMaster()会再次被调用。

例如: void OnLeftRoom() { ... }

实现了 IPunCallbacks.

8.38.2.13 virtual void Photon.PunBehaviour.OnLobbyStatisticsUpdate ( ) [virtual]

当主服务器发送一个游戏大厅统计更新、更新 PhotonNetwork.LobbyStatistics 时调用。

这个回调有两个前提条件：EnableLobbyStatistics 必须设置为 true，在客户端连接之前。并且客户端必须连接到主服务器，提供关于大堂的信息。

实现了 IPunCallbacks.

8.38.2.14 virtual void Photon.PunBehaviour.OnMasterClientSwitched (PhotonPlayer newMasterClient) [virtual]

在切换到一个新的主客户端后，且在当前客户端离开时被调用。

当这个客户进入某个房间时，这是不被调用的。当这个方法被调用时前主客户端仍在玩家列表。

例如: void OnMasterClientSwitched(PhotonPlayer newMasterClient) { ... }

实现了 IPunCallbacks.

8.38.2.15 virtual void Photon.PunBehaviour.OnOwnershipRequest (object[ ] viewAndPlayer) [virtual]

当另一个玩家从你（现在的所有者）这里请求一个 PhotonView 的所有权时调用。参数 viewAndPlayer 包含：

PhotonView view = viewAndPlayer[0] as PhotonView;

PhotonPlayer requestingPlayer = viewAndPlayer[1] as PhotonPlayer;

void OnOwnershipRequest(object[] viewAndPlayer) {} //

Parameters | 参数

viewAndPlayer	PhotonView 是 viewAndPlayer[0]，请求的玩家是 viewAndPlayer[1]。实现了 IPunCallbacks.

8.38.2.16 virtual void Photon.PunBehaviour.OnPhotonCreateRoomFailed (object[ ] codeAndMsg) [virtual]

当一个 CreateRoom()方法调用失败时被调用。

ErrorCode 和消息以数组的形式作为参数。最有可能是因为房间的名称已经在使用（其他客户端比你更快）。如果 PhotonNetwork.logLevel >= PhotonLogLevel.Informational 为真，PUN 会记录一些信息。

Parameters | 参数

codeAndMsg	codeAndMsg[0]是一个 short 类型的 ErrorCode，codeAndMsg[1]是一个字符串类型的调试信息。实现了 IPunCallbacks.

8.38.2.17 virtual void Photon.PunBehaviour.OnPhotonCustomRoomPropertiesChanged (Hashtable propertiesThatChanged)[virtual]

当一个房间的自定义属性更改时被调用。propertiesThatChanged 改变的属性包含所有通过 Room.SetCustomProperties 设置的。自从 v1.25 版本这个方法就有一个参数：Hashtable propertiesThatChanged。更改属性必须由 Room.SetCustomProperties 完成，导致这个回调函数局限在本地。

Parameters | 参数

propertiesThatChanged	实现了 IPunCallbacks.

8.38.2.18 virtual void Photon.PunBehaviour.OnPhotonInstantiate (PhotonMessageInfo info) [virtual]

在一个游戏对象（及其子类）通过使用 PhotonNetwork.Instantiate 方法被实例化时在任何脚本上被调用。

PhotonMessageInfo 参数提供关于谁创建的对象和什么时候创建的（基于 PhotonNetworking.time）的信息。

实现了 IPunCallbacks.

8.38.2.19 virtual void Photon.PunBehaviour.OnPhotonJoinRoomFailed (object[ ] codeAndMsg) [virtual]

当一个 JoinRoom()调用失败时被调用。参数以数组的形式提供错误代码 ErrorCode 和消息。

最有可能的错误是房间不存在或房间已满（一些其他的客户端比你更快）。如果满足条件 PhotonNetwork.logLevel >= PhotonLogLevel.Informational，PUN 会记录一些信息。

Parameters | 参数

codeAndMsg	codeAndMsg[0]是一个 short 类型的 ErrorCode，codeAndMsg[1]是一个字符串类型的调试信息。实现了 IPunCallbacks.

8.38.2.20 virtual void Photon.PunBehaviour.OnPhotonMaxCccuReached ( ) [virtual]

由于并发用户限制（暂时）到达了，该客户端会被服务器拒绝并断连。

当这种情况发生时，用户可能会等一会儿再试一次。在 OnPhotonMaxCcuReached()触发时你不能创建或加入房间，因为客户端会断开连接。你可以用一个新的许可证提高 CCU 的限制（当你使用自己的主机时）或扩展订阅（当使用 Photon 云服务时）。Photon 云会在达到 CCU 极限时给你发邮件。这也是在仪表板（网页）中可见的。

实现了 IPunCallbacks.

8.38.2.21 virtual void Photon.PunBehaviour.OnPhotonPlayerConnected (PhotonPlayer newPlayer) [virtual]

当一个远程玩家进入房间时调用。这个 PhotonPlayer 在这个时候已经被添加 playerlist 玩家列表。

如果你的游戏开始时就有一定数量的玩家，这个回调在检查 Room.playerCount 并发现你是否可以开始游戏时会很有用。

实现了 IPunCallbacks.

8.38.2.22 virtual void Photon.PunBehaviour.OnPhotonPlayerDisconnected (PhotonPlayer otherPlayer) [virtual]

当一个远程玩家离开房间时调用。这个 PhotonPlayer 此时已经从 playerlist 玩家列表删除。

当你的客户端调用 PhotonNetwork.leaveRoom 时，PUN 将在现有的客户端上调用此方法。当远程客户端关闭连接或被关闭时，这个回调函数会在经过几秒钟的暂停后被执行。

实现了 IPunCallbacks.

8.38.2.23 virtual void Photon.PunBehaviour.OnPhotonPlayerPropertiesChanged (object[ ]

playerAndUpdatedProps)[virtual]

当自定义玩家属性更改时调用。玩家和更改的属性被传递为对象数组 object[]。自从 v1.25 版本这个方法就有一个参数：object[] playerAndUpdatedProps，其中包含两个条目。

[0]是被影响的 PhotonPlayer 玩家。

[1]是那些改变了的属性的哈希表。

我们使用一个对象数组 object[]是由于 Unity 的 GameObject.SendMessage 方法的限制（该方法只有一个可选的参数）。

更改属性必须由 PhotonPlayer.SetCustomProperties 完成，导致这个回调局限在本地。

用例:

```
void OnPhotonPlayerPropertiesChanged(object[] playerAndUpdatedProps) {
    PhotonPlayer player = playerAndUpdatedProps[0] as PhotonPlayer;
    Hashtable props = playerAndUpdatedProps[1] as Hashtable;
    //分别将传入的对象数组参数赋值给临时变量，方便实现需要的操作
}
```

Parameters | 参数

playerAndUpdatedProps	包含 PhotonPlayer 和已改变的属性。实现了 IPunCallbacks.

8.38.2.24 virtual void Photon.PunBehaviour.OnPhotonRandomJoinFailed (object[ ] codeAndMsg) [virtual]

在一个 JoinRandom()请求失败后调用。参数提供 ErrorCode 错误代码和消息。

最有可能所有的房间都是满的或没有房间是可用的。

当使用多个大厅（通过 JoinLobby 或 TypedLobby），另一个大厅可能有更多/合适的房间。

如果 PhotonNetwork.logLevel >= PhotonLogLevel.Informational 为真，PUN 记录一些信息。

Parameters | 参数

codeAndMsg	codeAndMsg[0] 是 short 类型的 ErrorCode， codeAndMsg[1] 是 string 类型的调试信息 debug msg.实现了 IPunCallbacks.

8.38.2.25 virtual void Photon.PunBehaviour.OnReceivedRoomListUpdate ( ) [virtual]

在主服务器上的大厅内（PhotonNetwork.insideLobby）房间列表的任何更新都会调用该函数。PUN 通过 PhotonNetwork.GetRoomList()提供房间列表。

每一项都是一个 RoomInfo，其中可能包括自定义属性（提供你在创建一个房间时定义的那些 lobbylisted）。

不是所有类型的游戏大厅都会提供一系列的房间给客户端。有些游戏大厅是沉默的并且专门做服务器端的匹配（例如英雄联盟的游戏大厅就没有房间列表，所有的房间都是通过服务器

匹配的，自定义则是会提供房间列表）。

实现了 IPunCallbacks.

8.38.2.26 virtual void Photon.PunBehaviour.OnUpdatedFriendList ( ) [virtual]

当服务器发送一个对 FindFriends 请求的响应并更新 PhotonNetwork.Friends 时调用。

好友列表可作为 PhotonNetwork.Friends、列出的姓名、在线状态和玩家所在的房间（如果有）。

实现了 IPunCallbacks.

8.38.2.27 virtual void Photon.PunBehaviour.OnWebRpcResponse (OperationResponse response) [virtual]

当一个 WebRPC 的回应可用时被 PUN 调用。详见 PhotonNetwork.WebRPC。

重要：如果 Photon 可以使用你的网页服务 response.ReturnCode 是 0。回应的内容是你的网页服务发送的。你可以从中创建一个 WebResponse 实例。

用例: WebRpcResponse webResponse = new WebRpcResponse(operationResponse);

请注意:OperationResponse 类是在需要被使用的一个命名空间里，即 using ExitGames.Client.Photon; //包含 OperationResponse（和其他的类）。Photon 的 OperationResponse.ReturnCode 返回代码：0 代表“OK”；-3 代表“网页服务没有配置”（详见仪表盘/ 网页钩子）；-5 代表“网页服务现在有 RPC 路径/名称”（至少是 Azure）

用例: void OnWebRpcResponse(OperationResponse response) { ... }

实现了 IPunCallbacks.

8.39 PunRPC Class Reference | PunRPC 类参考

RPC 属性不同的名称替换。用于标记方法作为可远程调用的。

继承属性。

8.39.1 Detailed Description | 详细描述

RPC 属性不同的名称替换。用于标记方法作为可远程调用的。

8.40 RaiseEventOptions Class Reference | 类参考

聚集几个 RaiseEvent 操作不经常使用的选项。使用细节详见字段说明。

Public Attributes | 公共属性

- EventCaching CachingOption

定义服务器是否应该简单地发送事件，将其放入缓存或移除类似此事件的事件。

- byte InterestGroup

利益分组的号码，将其设置为 0 会发送给所有用户，除此外的 1 以上的分组，客户端必须要先订阅该分组。

- int[ ] TargetActors

需要发送该事件的 PhotonPlayer.IDs 列表。这样你就可以实现指向特定用户的事件了。

- ReceiverGroup Receivers

将事件发送到所有，主客户端或别的客户端（默认）。要注意主客户端，因为主客户端可能会在得到事件之前断连，那么该事件就丢失了。

- byte SequenceChannel

事件是按“通道”排列的。如果你有独立于其他的事件，它们可以进入另一个序列或通道。

- bool ForwardToWebhook

事件可以被转发到 Webhooks，Webhooks 可以评估和使用事件来跟踪游戏的状态。

- bool Encrypt

Static Public Attributes | 静态的公共属性

- static readonly RaiseEventOptions Default = new RaiseEventOptions()

默认选项：CachingOption：DoNotCache，InterestGroup：0、targetActors：null，receivers：Others，sequenceChannel：0。

8.40.1 Detailed Description | 详细描述

聚集几个 RaiseEvent 操作不经常使用的选项。使用细节详见字段说明。

8.40.2 Member Data Documentation | 成员数据文档

8.40.2.1 EventCaching RaiseEventOptions.CachingOption

定义服务器是否应该简单地发送事件，将其放入缓存或移除类似此事件的事件。

当使用选项：SliceSetIndex,、SlicePurgeIndex 或 SlicePurgeUpToIndex 时，设置一个CacheSliceIndex。除了 SequenceChannel 以外的所有其他选项被忽略。

8.40.2.2 readonly RaiseEventOptions RaiseEventOptions.Default = new RaiseEventOptions() [static]

默认选项：CachingOption：DoNotCache，InterestGroup：0、targetActors：null，receivers：Others，sequenceChannel：0。

8.40.2.3 bool RaiseEventOptions.Encrypt

8.40.2.4 bool RaiseEventOptions.ForwardToWebhook

事件可以被转发到 Webhooks，Webhooks 可以评估和使用事件来跟踪游戏的状态。

8.40.2.5 byte RaiseEventOptions.InterestGroup

利益分组的号码，将其设置为 0 会发送给所有用户，除此外的 1 以上的分组，客户端必须要先订阅该分组。

8.40.2.6 ReceiverGroup RaiseEventOptions.Receivers

将事件发送到所有，主客户端或别的客户端（默认）。要注意主客户端，因为主客户端可能会在得到事件之前断连，那么该事件就丢失了。

8.40.2.7 byte RaiseEventOptions.SequenceChannel

事件是按“通道”排列的。如果你有独立于其他的事件，它们可以进入另一个序列或通道。

8.40.2.8 int [ ] RaiseEventOptions.TargetActors

需要发送该事件的 PhotonPlayer.IDs 列表。这样你就可以实现指向特定用户的事件了。

8.41 Region Class Reference | 区域类参考

Public Member Functions | 公共成员函数

- override string ToString ()

Static Public Member Functions | 静态公共成员函数

- static CloudRegionCode Parse (string codeAsString)

Public Attributes | 公共属性

- CloudRegionCode Code

- string HostAndPort
- int Ping

8.41.1 Member Function Documentation | 成员函数文档

8.41.1.1 static CloudRegionCode Region.Parse (string codeAsString) [static]

8.41.1.2 override string Region.ToString ( )

8.41.2 Member Data Documentation | 成员数据文档

8.41.2.1 CloudRegionCode Region.Code

8.41.2.2 string Region.HostAndPort

8.41.2.3 int Region.Ping

8.42 Room Class Reference | 房间类参考

这个类描述了 PUN 加入（或已加入）的房间。与 RoomInfo 的属性相反，房间属性是可以设置的，且你可以关闭或隐藏“你的”房间。

继承 RoomInfo。

Public Member Functions | 公共成员函数

- void SetCustomProperties (Hashtable propertiesToSet, Hashtable expectedValues=null, bool webForward=false)

用新的/已更新的键-值对更新当前房间的自定义属性。

- void SetPropertiesListedInLobby (string[ ] propsListedInLobby)

如果选择房间时不是所有的属性都需要，使你可以定义游戏大厅里可用的属性。

- void ClearExpectedUsers ()

试图从服务器的 Slot Reservation 列表中删除所有当前期望的用户。

- override string ToString ()

以字符串的形式返回一个这个 Room 实例的总结。

- new string ToStringFull ()

以更长的字符串返回一个这个 Room 实例的总结，包括自定义属性。

Properties | 属性

- new string name [get, set]

房间的名称。房间/比赛的唯一的标识符（每个负载均衡组）。

- new bool open [get, set]

定义方式是否可以被加入。这不会影响游戏大厅里的列表，但是如果该房间没有开放的话则无法被加入，并且会从随机匹配的房间中排除。由于比赛条件，发现的比赛可能会在玩家加入前被关闭。只需重新连接到主服务器，再找到另一个比赛即可。使用"visible"属性来不列出房间。

- new bool visible [get, set]

定义该房间是否被列入游戏大厅。房间可以被隐身创建，或改变为不可见。要改变房间是否可以被加入，使用“open”属性。

- string[ ] propertiesListedInLobby [get, set]

应该被推送到游戏大厅并在大厅中列出的自定义属性列表。

- bool autoCleanUp [get]

获取该房间是否在玩家离开时使用 autoCleanUp 来移除所有（包括已缓存的）RPCs 和已实例化的游戏对象。

- new int maxPlayers [get, set]

设置该房间的玩家数量限制。这个属性也会在大厅里展示。如果房间满了(players count == maxplayers),那么加入该房间将会失败。

- new int playerCount [get]

该房间里的玩家总数。

- string[ ] expectedUsers [get]

期待加入该房间的玩家列表。在匹配时，Photon 会为这些 UserIDs 从 MaxPlayers 中预留位置。

Additional Inherited Members | 附加继承成员

8.42.1 Detailed Description | 详细描述

这个类描述了 PUN 加入（或已加入）的房间。与 RoomInfo 的属性相反，房间属性是可以设置的，且你可以关闭或隐藏“你的”房间。

8.42.2 Member Function Documentation | 成员函数文档

8.42.2.1 void Room.ClearExpectedUsers ( )

试图从服务器的 Slot Reservation 列表中删除所有当前期望的用户。

注意该操作可以和新的/其他用户加入冲突。他们可能会在该客户端调用 ClearExpectedUsers 之前或之后添加用户到期待列表中。

当服务器发送一个成功更新时，这个房间的 expectedUsers 值将随之更新。

内部：这种方法将设置一个房间的 ExpectedUsers 属性包装起来。

8.42.2.2 void Room.SetCustomProperties (Hashtable propertiesToSet, Hashtable expectedValues = null, bool webForward = false)

用新的/已更新的键-值对更新这个玩家的自定义属性。

自定义属性是一个对同房间内的所有玩家可用的键值对集合（哈希表）。它们可以关联到所在的房间或个别玩家，当只有某件事物的当前值是有用的时非常有用。例如：房间地图。所有的键必须是字符串。

Room 和 PhotonPlayer 类都有 SetCustomProperties 方法。同样的，两个类都通过 customProperties 提供访问当前键值对。

总是使用 SetCustomProperties 来改变值。为了减少网络流量，只设置实际改变了的值。

当新的属性被添加或存在的值被更新时，其他值将不会被改变，所以只需提供改变了的值或新增的。

要删除一个该房间里的已命名的（自定义）属性，使用 null 来作为值即可。

在本地，SetCustomProperties 将无延迟地更新其缓存。其他客户端通过 Photon（服务器）恰当的操作来被更新。

Check and Swap | 检查和交换

SetCustomProperties 可以选择做一个服务器端的 Check-And-Swap (CAS)：只有在期望的值是正确的时候才会被更新。expectedValues 可以和 propertiesToSet 不同的键/值。所以你可以检查一些键并设置另一个键的值（如果检查成功了）。

如果客户端已知的属性是错误的或过期的，则不能通过 CAS 设置值。CAS 被用来避免玩

家同时设置值时很有用。举个例子：如果所有的玩家都尝试拾取一些卡片或物品，只有一个玩家可以获取它。有了 CAS，只有第一个 SetProperties 会在服务器端得到执行，其他的（同时被发送）会失败。

服务器将广播成功地改变了值，本地的 customProperties “缓存”只有在往返后得到更新（如果有什么变化）。

你可以做一个"webForward"：Photon 将发送改变了的属性到为你的应用定义的 WebHook。

OfflineMode | 离线模式

当 PhotonNetwork.offlineMode 为 true 时，expectedValues 和 webForward 参数会被忽略。在离线模式里，本地的 customProperties 值被立即更新（无需往返服务器）。

Parameters | 参数

propertiesToSet	要设置的新属性。
expectedValues	至少一个属性键/值集，服务器端检查。键和值必须正确。离线模式里被忽略。
webForward	设置为 true，把设置属性推送到一个在仪表盘里为应用定义的 WebHook。离线模式里被忽略。

8.42.2.3 void Room.SetPropertiesListedInLobby (string[ ] propsListedInLobby)

如果选择房间时不是所有的属性都需要，使你可以定义游戏大厅里可用的属性。

限制发送到大厅里用户的属性数量是有道理的，因为这提高了速度和稳定性。

Parameters | 参数

propsListedInLobby	推送到大厅的自定义房间属性名称数组。

8.42.2.4 override string Room.ToString ( )

以字符串的形式返回一个这个 Room 实例的总结。

Returns | 返回值

返回这个 Room 实例的总结。

8.42.2.5 new string Room.ToStringFull ( )

以更长的字符串返回一个这个 Room 实例的总结，包括自定义属性。

Returns | 返回值

返回这个 Room 实例的总结。

8.42.3 Property Documentation | 属性文档

8.42.3.1 bool Room.autoCleanUp [get]

获取该房间是否在玩家离开时使用 autoCleanUp 来移除所有（包括已缓存的）RPCs 和已实例化的游戏对象。

8.42.3.2 string [ ] Room.expectedUsers [get]

期待加入该房间的玩家列表。在匹配时，Photon 会为这些 UserIDs 从 MaxPlayers 中预

留位置。

Photon 中相应的功能称为“Slot Reservation”，并且可以在文档页中找到该功能。你可以在 PhotonNetwork 方法 CreateRoom、JoinRoom 和 JoinOrCreateRoom 中定义一个 expectedUsers 的数组，来在房间里为这些玩家预留位置。

8.42.3.3 new int Room.maxPlayers [get], [set]

置该房间的玩家数量限制。这个属性也会在大厅里展示。如果房间满了(players count == maxplayers),那么加入该房间将会失败。

8.42.3.4 new string Room.name [get], [set]

房间的名称。房间/比赛的唯一的标识符（每个负载均衡组）。

8.42.3.5 new bool Room.open [get], [set]

定义方式是否可以被加入。这不会影响游戏大厅里的列表，但是如果该房间没有开放的话则无法被加入，并且会从随机匹配的房间中排除。由于比赛条件，发现的比赛可能会在玩家加入前被关闭。只需重新连接到主服务器，再找到另一个比赛即可。使用"visible"属性来不列出房间。

8.42.3.6 new int Room.playerCount [get]

这个房间里的玩家总数。

8.42.3.7 string [ ] Room.propertiesListedInLobby [get], [set]

应该被推送到游戏大厅并在大厅中列出的自定义属性列表。

8.42.3.8 new bool Room.visible [get], [set]

定义该房间是否被列入游戏大厅。房间可以被隐身创建，或改变为不可见。要改变房间是

否可以被加入，使用“open”属性。

8.43 RoomInfo Class Reference | RoomInfo 类参考

简化版的房间，只有房间列表和加入房间所必须的信息，用于游戏大厅内的列表。这些属性是可设置的（open、 MaxPlayers 等等）。

继承 Room。

Public Member Functions | 公共成员函数

- override bool Equals (object other)

使 RoomInfo 具有可比性（通过名称）。

- override int GetHashCode ()

协同 Equals 方法，使用名称的 HashCode 作为返回值。

- override string ToString ()

简单的 printingin 方法。

- string ToStringFull ()

简单的 printingin 方法。

Protected Attributes | 受保护的字段

- byte maxPlayersField = 0

属性支持字段。

- string[ ] expectedUsersField

属性支持字段。

- bool openField = true

属性支持字段。

- bool visibleField = true

属性支持字段。

- bool autoCleanUpField = PhotonNetwork.autoCleanUpPlayerObjects

属性支持字段。False 除非 GameProperty 被设置为 true（否则它不会被发送）。

- string nameField

属性支持字段。

Properties | 属性

- bool removedFromList [get, set]

用于游戏大厅内部，以标示不再列出的房间。

- Hashtable customProperties [get]

房间的自定义属性的只读“缓存”。通过 Room.SetCustomProperties 来设置（对 RoomInfo 类不可用！）。

- string name [get]

房间的名称。房间/比赛的唯一的标识符（每个负载均衡组）。

- int playerCount [get, set]

仅在游戏大厅内部使用，用来展示房间里的玩家数量（当你还没有进入的时候）。

- bool isLocalClientInside [get, set]

如果本地客户端已经在游戏里了或仍将在游戏服务器上加入，这两种状态为 true（在游戏

大厅时总是为 false）。

- byte maxPlayers [get]

设置该房间的玩家数量限制。这个属性也会在大厅里展示。如果房间满了(players count == maxplayers),那么加入该房间将会失败。

- bool open [get]

定义方式是否可以被加入。这不会影响游戏大厅里的列表，但是如果该房间没有开放的话则无法被加入，并且会从随机匹配的房间中排除。由于比赛条件，发现的比赛可能会在玩家加入前被关闭。只需重新连接到主服务器，再找到另一个比赛即可。使用"visible" 属性来不列出房间。

- bool visible [get]

定义该房间是否被列入游戏大厅。房间可以被隐身创建，或改变为不可见。要改变房间是否可以被加入，使用“open”属性。

8.43.1 Detailed Description | 详细描述

简化版的房间，只有房间列表和加入房间所必须的信息，用于游戏大厅内的列表。这些属性是可设置的（open、 MaxPlayers 等等）。

这个类抽象了关于可用房间的信息，由主服务器的大厅发送。由于所有值都是只读，没有一个是同步的（只是由服务器通过事件更新）。

8.43.2 Member Function Documentation | 成员函数文档

8.43.2.1 override bool RoomInfo.Equals (object other)

使 RoomInfo 具有可比性（通过名称）。

8.43.2.2 override int RoomInfo.GetHashCode ( )

协同 Equals 方法，使用名称的 HashCode 作为返回值。

8.43.2.3 override string RoomInfo.ToString ( )

简单的 printingin 方法。

Returns | 返回值

返回这个 RoomInfo 实例的总结。

8.43.2.4 string RoomInfo.ToStringFull ( )

简单的 printingin 方法。

Returns | 返回值

返回这个 RoomInfo 实例的总结。

8.43.3 Member Data Documentation | 成员数据文档

8.43.3.1 bool RoomInfo.autoCleanUpField = PhotonNetwork.autoCleanUpPlayerObjects [protected]

属性支持字段。False 除非 GameProperty 被设置为 true（否则它不会被发送）。

8.43.3.2 string [ ] RoomInfo.expectedUsersField [protected]

属性支持字段。

8.43.3.3 byte RoomInfo.maxPlayersField = 0 [protected]

属性支持字段。

8.43.3.4 string RoomInfo.nameField [protected]

属性支持字段。

8.43.3.5 bool RoomInfo.openField = true [protected]

属性支持字段。

8.43.3.6 bool RoomInfo.visibleField = true [protected]

属性支持字段。

8.43.4 Property Documentation | 属性文档

8.43.4.1 Hashtable RoomInfo.customProperties [get]

房间的自定义属性的只读“缓存”。通过 Room.SetCustomProperties 来设置（对 RoomInfo 类不可用！）。

所有键都是字符串类型的，且该值取决于游戏/应用程序。

8.43.4.2 bool RoomInfo.isLocalClientInside [get], [set]

如果本地客户端已经在游戏里了或仍将在游戏服务器上加入，这两种状态为 true（在游戏大厅时总是为 false）。

8.43.4.3 byte RoomInfo.maxPlayers [get]

设置该房间的玩家数量限制。这个属性也会在大厅里展示。如果房间满了(players count == maxplayers),那么加入该房间将会失败。

作为 RoomInfo 的一部分这个属性不能被设置。作为（玩家所加入的）Room 的一部分，设置者将更新服务器和所有的客户端。

8.43.4.4 string RoomInfo.name [get]

房间的名称。房间/比赛的唯一的标识符（每个负载均衡组）。

8.43.4.5 bool RoomInfo.open [get]

定义方式是否可以被加入。这不会影响游戏大厅里的列表，但是如果该房间没有开放的话则无法被加入，并且会从随机匹配的房间中排除。由于比赛条件，发现的比赛可能会在玩家加入前被关闭。只需重新连接到主服务器，再找到另一个比赛即可。使用"IsVisible" 属性来不列出房间。

作为 RoomInfo 的一部分这个属性不能被设置。作为（玩家所加入的）Room 的一部分，设置者将更新服务器和所有的客户端。

8.43.4.6 int RoomInfo.playerCount [get], [set]

仅在游戏大厅内部使用，用来展示房间里的玩家数量（当你还没有进入的时候）。

8.43.4.7 bool RoomInfo.removedFromList [get], [set]

用于游戏大厅内部，以标示不再列出的房间。

8.43.4.8 bool RoomInfo.visible [get]

定义该房间是否被列入游戏大厅。房间可以被隐身创建，或改变为不可见。要改变房间是否可以被加入，使用“open”属性。

作为 RoomInfo 的一部分这个属性不能被设置。作为（玩家所加入的）Room 的一部分，设置者将更新服务器和所有的客户端。

8.44 RoomOptions Class Reference | 房间选项类参考

当你创建房间时把需要的常用房间属性包装起来。更多细节请阅读个别条目。

Public Attributes | 公共字段

- byte MaxPlayers

任何时候可以在房间里的最大玩家数量。0 代表“没有限制”。

- int PlayerTtl

房间里的“演员”的 Time To Live (TTL)。如果客户端断连，这个演员首先变得无效，并在超时后移除。以毫秒为单位。

- int EmptyRoomTtl

当最后的玩家离开房间时房间的 Time To Live (TTL)。将房间保存在缓存中以免玩家稍后重新加入房间。以毫秒为单位。

- Hashtable CustomRoomProperties

房间的自定义属性设置。使用字符串键！

- string[ ] CustomRoomPropertiesForLobby = new string[0]

定义在大堂中列出的自定义房间属性.。

- string[ ] Plugins

通知服务器的预期插件设置。

Properties | 属性

- bool IsVisible [get, set]

定义这个房间是否在游戏大厅内被列出。如果不列出，它也不能被随机加入。

- bool IsOpen [get, set]

定义这个房间是否可以被加入。

- bool CleanupCacheOnLeave [get, set]

当用户离开时从房间移除该用户的事件和属性。

- bool SuppressRoomEvents [get]

告诉服务器跳过加入和离开玩家的房间事件。

- bool PublishUserId [get, set]

定义房间里的玩家 UserIds 是否“发布”。如果玩家想要在另一局游戏中一起玩，对于 FindFriends 方法很有用。

- bool isVisible [get, set]
- bool isOpen [get, set]
- byte maxPlayers [get, set]
- bool cleanupCacheOnLeave [get, set]
- Hashtable customRoomProperties [get, set]
- string[ ] customRoomPropertiesForLobby [get, set]
- string[ ] plugins [get, set]
- bool suppressRoomEvents [get]
- bool publishUserId [get, set]

8.44.1 Detailed Description | 详细描述

当你创建房间时把需要的常用房间属性包装起来。更多细节请阅读个别条目。

这直接映射到 Room 类中的字段.。

8.44.2 Member Data Documentation | 成员数据文档

8.44.2.1 Hashtable RoomOptions.CustomRoomProperties

房间的自定义属性设置。使用字符串键！

自定义房间属性是您需要定义游戏设置的任何键值对。你的键越短越好。例如：Map、Mode（当使用"Map"时键可以是 ‘m’ ），TileSet（键可能是 ‘t’ ）。

8.44.2.2 string [ ] RoomOptions.CustomRoomPropertiesForLobby = new string[0]

定义在游戏大厅中列出的自定义房间属性.。

为在大厅中的客户端提供可用的自定义房间属性命名。谨慎使用。除非定制属性对于匹配或用户信息至关重要，否则不应该发送给大厅，这会导致大厅的拥堵和延迟。

默认：没有自定义属性被发送到大堂。

8.44.2.3 int RoomOptions.EmptyRoomTtl

当最后的玩家离开房间时房间的 Time To Live (TTL)。将房间保存在缓存中以免玩家稍后重新加入房间。以毫秒为单位。

8.44.2.4 byte RoomOptions.MaxPlayers

任何时候可以在房间里的最大玩家数量。0 代表“没有限制”。

8.44.2.5 int RoomOptions.PlayerTtl

房间里的“演员”的 Time To Live (TTL)。如果客户端断连，这个演员首先变得无效，并在超时后移除。以毫秒为单位。

8.44.2.6 string [ ] RoomOptions.Plugins

通知服务器的预期插件设置。

在插件不匹配返回错误代码 PluginMismatch 32757(0x7FFF - 10)时操作将失败。

设置 string[]{}意味着客户端不需要安装插件。注：为了向后兼容 null 省略任何检查。

8.44.3 Property Documentation | 属性文档

8.44.3.1 bool RoomOptions.CleanupCacheOnLeave [get], [set]

当用户离开时从房间移除该用户的事件和属性。

当房间里的玩家不能在房间里放置东西且只是完全消失时，这是合理的。当你禁用这个时，如果房间保持在无限期使用，事件历史会变得太长而难以加载。默认值：true。清除缓存和离开用户的属性。

8.44.3.2 bool RoomOptions.cleanupCacheOnLeave [get], [set]

8.44.3.3 Hashtable RoomOptions.customRoomProperties [get], [set]

8.44.3.4 string [ ] RoomOptions.customRoomPropertiesForLobby [get], [set]

8.44.3.5 bool RoomOptions.IsOpen [get], [set]

定义这个房间是否可以被加入。

如果一个房间被关闭，没有玩家可以加入该房间。举例来说，当 3 个可能的玩家提前开始了游戏，且不希望任何人在游戏期间加入时，关闭房间就是合理的。房间仍然可以在游戏大厅里列出（设置 IsVisible 来控制房间是否在大厅隐藏）。

8.44.3.6 bool RoomOptions.isOpen [get], [set]

8.44.3.7 bool RoomOptions.IsVisible [get], [set]

定义这个房间是否在游戏大厅内被列出。如果不列出，它也不能被随机加入。

一个不可见的房间将被排除在被发送到大厅里的客户端的房间列表外。一个隐身的房间可以通过名称加入，但是不能被随机匹配到。

用这个来“隐藏”一个房间和模拟“私人房间”。玩家可以交换一个 roomname，通过隐身创建来避免别人加入。

8.44.3.8 bool RoomOptions.isVisible [get], [set]

8.44.3.9 byte RoomOptions.maxPlayers [get], [set]

8.44.3.10 string [ ] RoomOptions.plugins [get], [set]

8.44.3.11 bool RoomOptions.PublishUserId [get], [set]

定义房间里的玩家 UserIds 是否“发布”。如果玩家想要在另一局游戏中一起玩，对于 FindFriends 方法很有用。

当你设置这个为 true 时，Photon 将在对应的房间里发布玩家的 UserIds。这样，你就可以使用 PhotonPlayer.userId 来访问任何玩家的 userID。对于 FindFriends 方法很有用，在房间里设置"expected users"来预留位置时也会用到（详见 PhotonNetwork.JoinRoom 案例）

8.44.3.12 bool RoomOptions.publishUserId [get], [set]

8.44.3.13 bool RoomOptions.SuppressRoomEvents [get]

告诉服务器跳过加入和离开玩家的房间事件。

使用这个属性来使客户端不知道房间里的其他玩家。如果你有一些服务器逻辑来更新玩家，这可以节省一些流量，但它也可以限制客户端的可用性。

如果你使用这个属性 PUN 将损坏，所以它不可设置。

8.44.3.14 bool RoomOptions.suppressRoomEvents [get]

8.45 UnityEngine.SceneManagement.SceneManager

Class Reference | 类参考

为老版本 Unity SceneManager 的最小化实现，直到 v5.2 版本。

Static Public Member Functions | 静态公共成员函数

- static void LoadScene (string name)
- static void LoadScene (int buildIndex)

8.45.1 Detailed Description | 详细描述

为老版本 Unity SceneManager 的最小化实现，直到 v5.2 版本。

8.45.2 Member Function Documentation | 成员函数文档

8.45.2.1 static void UnityEngine.SceneManagement.SceneManager.LoadScene (string name) [static]

8.45.2.2 static void UnityEngine.SceneManagement.SceneManager.LoadScene (int buildIndex) [static]

8.46 SceneManagerHelper Class Reference | 类参考

Properties | 属性

- static string ActiveSceneName [get]
- static int ActiveSceneBuildIndex [get]

8.46.1 Property Documentation | 属性文档

8.46.1.1 int SceneManagerHelper.ActiveSceneBuildIndex [static], [get]

8.46.1.2 string SceneManagerHelper.ActiveSceneName [static], [get]

8.47 ServerSettings Class Reference | 类参考

连接相关设置的集合，由 PhotonNetwork.ConnectUsingSettings 内部使用。

继承 ScriptableObject.

Public Types | 公共类型

- enum HostingOption {

HostingOption.NotSet = 0, HostingOption.PhotonCloud = 1,

HostingOption.SelfHosted = 2, HostingOption.OfflineMode = 3,

HostingOption.BestRegion = 4 }

Public Member Functions | 公共成员函数

- void UseCloudBestRegion (string cloudAppid)
- void UseCloud (string cloudAppid)
- void UseCloud (string cloudAppid, CloudRegionCode code)
- void UseMyServer (string serverAddress, int serverPort, string application)
- override string ToString ()

Public Attributes | 公共属性

- HostingOption HostType = HostingOption.NotSet
- ConnectionProtocol Protocol = ConnectionProtocol.Udp
- string ServerAddress = ""
- int ServerPort = 5055

- string AppID = ""
- string VoiceAppID = ""
- CloudRegionCode PreferredRegion
- CloudRegionFlag EnabledRegions = (CloudRegionFlag)(-1)
- bool JoinLobby
- bool EnableLobbyStatistics
- List< string > RpcList = new List<string>()
- bool DisableAutoOpenWizard

8.47.1 Detailed Description | 详细描述

连接相关设置的集合，由 PhotonNetwork.ConnectUsingSettings 内部使用。

8.47.2 Member Enumeration Documentation | 成员枚举文档

8.47.2.1 enum ServerSettings.HostingOption

Enumerator | 枚举器

NotSet | 没设置

PhotonCloud | Photon 云端

SelfHosted | 自己托管

OfflineMode | 离线模式

BestRegion | 最佳域

8.47.3 Member Function Documentation | 成员函数文档

8.47.3.1 override string ServerSettings.ToString ( )

8.47.3.2 void ServerSettings.UseCloud (string cloudAppid)

8.47.3.3 void ServerSettings.UseCloud (string cloudAppid, CloudRegionCode code)

8.47.3.4 void ServerSettings.UseCloudBestRegion (string cloudAppid)

8.47.3.5 void ServerSettings.UseMyServer (string serverAddress, int serverPort, string application)

8.47.4 Member Data Documentation | 成员数据文档

8.47.4.1 string ServerSettings.AppID = ""

8.47.4.2 bool ServerSettings.DisableAutoOpenWizard

8.47.4.3 CloudRegionFlag ServerSettings.EnabledRegions = (CloudRegionFlag)(-1)

8.47.4.4 bool ServerSettings.EnableLobbyStatistics

8.47.4.5 HostingOption ServerSettings.HostType = HostingOption.NotSet

8.47.4.6 bool ServerSettings.JoinLobby

8.47.4.7 CloudRegionCode ServerSettings.PreferredRegion

8.47.4.8 ConnectionProtocol ServerSettings.Protocol = ConnectionProtocol.Udp

8.47.4.9 List<string> ServerSettings.RpcList = new List<string>()

8.47.4.10 string ServerSettings.ServerAddress = ""

8.47.4.11 int ServerSettings.ServerPort = 5055

8.47.4.12 string ServerSettings.VoiceAppID = ""

8.48 PhotonAnimatorView.SynchronizedLayer Class Reference | 类参考

Public Attributes | 公共字段

- SynchronizeType SynchronizeType
- int LayerIndex

8.48.1 Member Data Documentation | 成员数据文档

8.48.1.1 int PhotonAnimatorView.SynchronizedLayer.LayerIndex

8.48.1.2 SynchronizeType PhotonAnimatorView.SynchronizedLayer.SynchronizeType

8.49 PhotonAnimatorView.SynchronizedParameter Class Reference | 类参考

Public Attributes | 公共字段

- ParameterType Type
- SynchronizeType SynchronizeType
- string Name

8.49.1 Member Data Documentation | 成员数据文档

8.49.1.1 string PhotonAnimatorView.SynchronizedParameter.Name

8.49.1.2 SynchronizeType

PhotonAnimatorView.SynchronizedParameter.SynchronizeType

8.49.1.3 ParameterType PhotonAnimatorView.SynchronizedParameter.Type

8.50 TypedLobby Class Reference | 类参考

指服务器上的特定的游戏大厅（和类型）。

由 TypedLobbyInfo 继承。

Public Member Functions | 公共成员函数

- TypedLobby ()
- TypedLobby (string name, LobbyType type)
- override string ToString ()

Public Attributes | 公共字段

- string Name

这个游戏添加到的大厅名称。默认：null，被附加到默认大厅。每个 lobbyName +lobbyType 组合都是唯一的大厅，所以当几种类型存在时可以使用相同的名称。

- LobbyType Type

这个游戏添加到的（命名的）大厅的类型。

Static Public Attributes | 静态公共字段

- static readonly TypedLobby Default = new TypedLobby()

Properties | 属性

- bool IsDefault [get]

8.50.1 Detailed Description | 详细描述

指服务器上的特定的游戏大厅（和类型）。

名称和类型是一个游戏大厅的唯一标识符。

通过 PhotonNetwork.JoinLobby(TypedLobby lobby)方法来加入一个游戏大厅。

当前游戏大厅被保存在 PhotonNetwork.lobby 里。

8.50.2 Constructor & Destructor Documentation | 构造与析构文档

8.50.2.1 TypedLobby.TypedLobby ( )

8.50.2.2 TypedLobby.TypedLobby (string name, LobbyType type)

8.50.3 Member Function Documentation | 成员函数文档

8.50.3.1 override string TypedLobby.ToString ( )

8.50.4 Member Data Documentation | 成员数据文档

8.50.4.1 readonly TypedLobby TypedLobby.Default = new TypedLobby()

[static]

8.50.4.2 string TypedLobby.Name

这个游戏添加到的大厅名称。默认：null，被附加到默认大厅。每个 lobbyName +lobbyType 组合都是唯一的大厅，所以当几种类型存在时可以使用相同的名称。

8.50.4.3 LobbyType TypedLobby.Type

这个游戏添加到的（命名的）大厅的类型。

8.50.5 Property Documentation | 属性文档

8.50.5.1 bool TypedLobby.IsDefault [get]

8.51 TypedLobbyInfo Class Reference | 类参考

继承 TypedLobby.

Public Member Functions | 公共成员函数

- override string ToString ()

Public Attributes | 公共字段

- int PlayerCount
- int RoomCount

Additional Inherited Members | 附加继承成员

8.51.1 Member Function Documentation | 成员函数文档

8.51.1.1 override string TypedLobbyInfo.ToString ( )

8.51.2 Member Data Documentation | 成员数据文档

8.51.2.1 int TypedLobbyInfo.PlayerCount

8.51.2.2 int TypedLobbyInfo.RoomCount

8.52 WebRpcResponse Class Reference | 类参考

读取 WebRpc 的操作回应并提供对最常见值的便捷访问。

Public Member Functions | 公共成员函数

- WebRpcResponse (OperationResponse response)

读取一个 WebRpc 的 OperationResponse 值。

- string ToStringFull ()

将回应转化成更容易读取的字符串。

Properties | 属性

- string Name [get, set]

被调用的 WebRpc 名称。

- int ReturnCode [get, set]

WebService 回应 WebRpc 的 ReturnCode。

- string DebugMessage [get, set]

可能是空字符串或 null。

- Dictionary< string, object > Parameters [get, set]

网络服务回应 WebRpc 所返回的其他键/值对。

8.52.1 Detailed Description | 详细描述

读取 WebRpc 的操作回应并提供对最常见值的便捷访问。

详见 PhotonNetwork.WebRpc 方法。

创建一个 WebRpcResponse 来访问常见结果值。

operationResponse.OperationCode 应该是: OperationCode.WebRpc.

8.52.2 Constructor & Destructor Documentation | 构造与析构文档

8.52.2.1 WebRpcResponse.WebRpcResponse (OperationResponse response)

读取一个 WebRpc 的 OperationResponse 值。

8.52.3 Member Function Documentation | 成员函数文档

8.52.3.1 string WebRpcResponse.ToStringFull ( )

将回应转化成更容易读取的字符串。

Returns | 返回值

代表结果的字符串。

8.52.4 Property Documentation | 属性文档

8.52.4.1 string WebRpcResponse.DebugMessage [get], [set]

可能是空字符串或 null。

8.52.4.2 string WebRpcResponse.Name [get], [set]

被调用的 WebRpc 名称。

8.52.4.3 Dictionary<string, object> WebRpcResponse.Parameters [get], [set]

网络服务回应 WebRpc 所返回的其他键/值对。

8.52.4.4 int WebRpcResponse.ReturnCode [get], [set]

WebService 回应 WebRpc 的 ReturnCode。

0 是常用的成功信号。

-1 告诉你：WebRpc 服务没有 ReturnCode。

其他的 ReturnCodes 由单独的 WebRpc 和服务定义。

第 9 章文件文档

9.1 general.md 文件参考

C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/_Doc/general.md File Reference | 文件参考

9.2 main.md 文件参考

C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/_Doc/main.md File Reference | 文件参考

9.3 optionalGui.md 文件参考

C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/_Doc/optionalGui.md File Reference | 文件参考

9.4 photonStatsGui.md 文件参考

C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/_Doc/photonStatsGui.md File Reference | 文件参考

9.5 publicApi.md 文件参考

C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/_Doc/publicApi.md File Reference | 文件参考

9.6 CustomTypes.cs 文件参考

C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/CustomTypes.cs File Reference | 文件参考

设置 Unity 的指定类型支持。可以是一个教你如何注册自己用来发送的自定义类型的蓝图。

Classes | 类

- class **CustomTypes**

内部使用类，包含对不同 Unity 指定类的解序列化/序列化方法。将其添加到 Photon 的序列化协议中，从而允许你在事件中发送它们，等等。

9.6.1 Detailed Description | 详细描述

设置 Unity 的指定类型支持。可以是一个教你如何注册自己用来发送的自定义类型的蓝图。

9.7 Enums.cs 文件参考

C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/Enums.cs File Reference

总结了几个常用的枚举。

Classes | 类

- class EncryptionDataParameters

Enumerations | 枚举器

- enum PhotonNetworkingMessage {

PhotonNetworkingMessage.OnConnectedToPhoton,

PhotonNetworkingMessage.OnLeftRoom,

PhotonNetworkingMessage.OnMasterClientSwitched,

PhotonNetworkingMessage.OnPhotonCreateRoomFailed,

PhotonNetworkingMessage.OnPhotonJoinRoomFailed,

PhotonNetworkingMessage.OnCreatedRoom,

PhotonNetworkingMessage.OnJoinedLobby,

PhotonNetworkingMessage.OnLeftLobby,

PhotonNetworkingMessage.OnDisconnectedFromPhoton,

PhotonNetworkingMessage.OnConnectionFail,

PhotonNetworkingMessage.OnFailedToConnectToPhoton,

PhotonNetworkingMessage.OnReceivedRoomListUpdate,

PhotonNetworkingMessage.OnJoinedRoom,

PhotonNetworkingMessage.OnPhotonPlayerConnected,

PhotonNetworkingMessage.OnPhotonPlayerDisconnected,

PhotonNetworkingMessage.OnPhotonRandomJoinFailed,

PhotonNetworkingMessage.OnConnectedToMaster,

PhotonNetworkingMessage.OnPhotonSerializeView,

PhotonNetworkingMessage.OnPhotonInstantiate,

PhotonNetworkingMessage.OnPhotonMaxCccuReached,

PhotonNetworkingMessage.OnPhotonCustomRoomPropertiesChanged,

PhotonNetworkingMessage.OnPhotonPlayerPropertiesChanged,

PhotonNetworkingMessage.OnUpdatedFriendList,

PhotonNetworkingMessage.OnCustomAuthenticationFailed,

PhotonNetworkingMessage.OnCustomAuthenticationResponse,

PhotonNetworkingMessage.OnWebRpcResponse,

PhotonNetworkingMessage.OnOwnershipRequest,

PhotonNetworkingMessage.OnLobbyStatisticsUpdate }

这个枚举定义了 PUN 用来作为回调函数的 MonoMessages 集。由 PunBehaviour 实现。

- enum PhotonLogLevel { PhotonLogLevel.ErrorsOnly,

PhotonLogLevel.Informational, PhotonLogLevel.Full }

用于定义由 PUN 类创建的日志输出的级别。无论是错误日志，信息（更多的一些）或全部。

- enum PhotonTargets {

PhotonTargets.All, PhotonTargets.Others, PhotonTargets.MasterClient,

PhotonTargets.AllBuffered,PhotonTargets.OthersBuffered,

PhotonTargets.AllViaServer, PhotonTargets.AllBufferedViaServer }

RPCs 的“目标”选项枚举。这些定义了哪些远程客户端得到你的 RPC 调用。

- enum CloudRegionCode {

CloudRegionCode.eu = 0, CloudRegionCode.us = 1, CloudRegionCode.asia = 2,

CloudRegionCode.jp = 3,CloudRegionCode.au = 5, CloudRegionCode.usw = 6,

CloudRegionCode.sa = 7, CloudRegionCode.cae = 8,

CloudRegionCode.none = 4 }

当前可用的 Photon Cloud regions 枚举。

- enum CloudRegionFlag {

CloudRegionFlag.eu = 1 << 0, CloudRegionFlag.us = 1 << 1,

CloudRegionFlag.asia = 1 << 2, CloudRegionFlag.jp = 1 << 3,

CloudRegionFlag.au = 1 << 4, CloudRegionFlag.usw = 1 << 5,

CloudRegionFlag.sa = 1 << 6, CloudRegionFlag.cae = 1 << 7 }

可用区域旗帜枚举。用于作为最佳 Region pinging 的“已启用”旗帜。

- enum ConnectionState {

ConnectionState.Disconnected, ConnectionState.Connecting,

ConnectionState.Connected, ConnectionState.Disconnecting,

ConnectionState.InitializingApplication }

客户端的高级连接状态。最好使用更加详细的 ClientState。

- enum EncryptionMode { EncryptionMode.PayloadEncryption,

EncryptionMode.DatagramEncryption = 10 }

定义通信如何加密。

9.7.1 Detailed Description | 详细描述

总结了几个常用的枚举。

9.7.2 Enumeration Type Documentation | 枚举类型文档

9.7.2.1 enum CloudRegionCode

当前可用的 Photon Cloud regions 枚举。

这被用于 PhotonNetwork.ConnectToRegion。

Enumerator | 枚举器

eu 位于阿姆斯特丹的欧洲服务器。

us 美国服务器（东海岸）。

asia 新加坡亚洲服务器。

jp 位于东京日本服务器。

au 澳大利亚墨尔本服务器。

usw 美国西部。

sa 美国南部。

cae 加拿大东部，蒙特利尔。

none 没有选择区域。

9.7.2.2 enum CloudRegionFlag

可用区域旗帜枚举。用于作为最佳 Region pinging 的“已启用”旗帜。

请注意，这些枚举值跳过 CloudRegionCode.none，且其值是按照严格秩序排列（功率 2）。

Enumerator | 枚举器

eu

us

asia

jp

au

usw

sa

cae

9.7.2.3 enum ConnectionState

客户端的高级连接状态。最好使用更加详细的 ClientState。

Enumerator | 枚举器

Disconnected | 已断开

Connecting | 正在连接

Connected | 已连接

Disconnecting | 正在断开连接

InitializingApplication | 初始化应用

9.7.2.4 enum EncryptionMode

定义通信如何加密。

Enumerator | 枚举器

PayloadEncryption 这是默认的加密模式 :消息只会按需求加密(当你把参数 "encrypt" 设置为 true 时发送操作 ）。

DatagramEncryption 为 UDP 设计的加密方式，连接获取设置且所有进一步的数据得到几乎完整的加密。按需加密消息（就像 PayloadEncryption 里面那样）被跳过。

该模式必须 AuthOnce 或 AuthOnceWss 作为 AuthMode!

9.8 Extensions.cs 文件参考

C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/Extensions.cs File Reference | 文件参考

Classes | 类

- class Extensions

这个静态类为几个现有的类（如 Vector3、float 及其他）定义了一些有用的扩展方法。

- class GameObjectExtensions

少量的扩展方法，使 PUN 更容易兼容 Unity 的多版本。

Typedefs | 类型定义

- using Hashtable = ExitGames.Client.Photon.Hashtable
- using SupportClassPun = ExitGames.Client.Photon.SupportClass

9.8.1 Typedef Documentation | 类型定义文档

9.8.1.1 using Hashtable = ExitGames.Client.Photon.Hashtable

9.8.1.2 using SupportClassPun = ExitGames.Client.Photon.SupportClass

9.9 FriendInfo.cs 文件参考

C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/FriendInfo.cs File Reference | 文件参考

Classes | 类

- class FriendInfo

用于存储关于朋友在线状态以及他/她在哪个房间的信息。

9.10 GizmoType.cs 文件参考

C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/GizmoType.cs File Reference | 文件参考

Classes | 类

- class ExitGames.Client.GUI.GizmoTypeDrawer

Namespaces | 命名空间

- package ExitGames.Client.GUI

Enumerations | 枚举

- enum ExitGames.Client.GUI.GizmoType

{ ExitGames.Client.GUI.GizmoType.WireSphere,

ExitGames. Client.GUI.GizmoType.Sphere,

ExitGames.Client.GUI.GizmoType.WireCube,

ExitGames.Client.GUI. GizmoType.Cube }

9.11 LoadbalancingPeer.cs 文件参考

C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/LoadbalancingPeer.cs File Reference | 文件参考

Classes | 类

- class **LoadBalancingPeer**

PUN 内部使用。一个 LoadbalancingPeer 提供操作，枚举定义需要使用负载均衡服务器应用，该应用也使用 Photon 云。

- class **OpJoinRandomRoomParams**
- class **EnterRoomParams**
- class ErrorCode

ErrorCode 定义了与 Photon 客户端/服务器通信相关的默认代码。

- class ActorProperties

常量类。这些（字节）值为演员/玩家定义的“众所周知”的属性。PUN 在内部使用这些常量。

- class GamePropertyKey

常量类。这些（字节）的值是用于 Photon 负载均衡的“著名”的房间/游戏属性。

Pun 在内部使用这些常量

- class EventCode

常量类。这些值是由 Photon 的负载均衡定义的事件。Pun 在内部使用这些常量。

- class ParameterCode

常量类。操作和事件的参数代码。Pun 在内部使用这些常量。

- class OperationCode

常量类。包含操作码。Pun 在内部使用这些常量。

- class RoomOptions

当您创建房间时，需要添加的常用房间属性。更多细节请阅读个别条目。

- class RaiseEventOptions

聚集几个 RaiseEvent 操作不经常使用的选项。有关使用细节详见字段说明。

- class TypedLobby

指服务器上的特定大厅（和类型）。

- class TypedLobbyInfo

- class AuthenticationValues

用于 Photon 用户认证的容器。在你连接前设置 AuthValues。

Enumerations | 枚举

- enum JoinMode : byte { JoinMode.Default = 0, JoinMode.CreateIfNotExists = 1,

JoinMode.JoinOrRejoin = 2,JoinMode.RejoinOnly = 3 }

为 OpJoinRoom 和 OpJoinOrCreate 定义可能的值。它告诉服务器房间是否只能正常被

加入、隐式地创建或在网络服务上被发现的回合制游戏。

- enum MatchmakingMode : byte { MatchmakingMode.FillRoom = 0,

  MatchmakingMode.SerialMatching = 1,

  MatchmakingMode.RandomMatching = 2 }

  OpJoinRandom 匹配规则选项。

- enum ReceiverGroup : byte { ReceiverGroup.Others = 0, ReceiverGroup.All = 1,

  ReceiverGroup.MasterClient= 2 }

  轻量版-OpRaiseEvent 让你选择在房间里的哪些演员应该接收事件。默认情况下，事件被发送到“Others”，但你可以否决这个。

- enum EventCaching : byte {

  EventCaching.DoNotCache = 0, EventCaching.MergeCache = 1,

  EventCaching.ReplaceCache = 2, EventCaching.RemoveCache = 3,

  EventCaching.AddToRoomCache = 4, EventCaching.AddToRoomCacheGlobal = 5,

  EventCaching.RemoveFromRoomCache = 6,

  EventCaching.RemoveFromRoomCacheForActorsLeft = 7,

  EventCaching.SliceIncreaseIndex = 10, EventCaching.SliceSetIndex = 11,

  EventCaching.SlicePurgeIndex = 12, EventCaching.SlicePurgeUpToIndex = 13 }

  轻量版-OpRaiseEvent 允许你缓存事件并自动将其发送给加入房间里的玩家。事件缓存每个事件代码和玩家：事件 100（例如！）可以每玩家存储一次。缓存事件可以被修改、替换和删除。

- enum PropertyTypeFlag : byte { PropertyTypeFlag.None = 0x00, PropertyTypeFlag.Game = 0x01, PropertyTypeFlag.Actor = 0x02, PropertyTypeFlag.GameAndActor = Game j Actor }

"types of properties"的旗帜，在 OpGetProperties 中被用作过滤器。

- enum LobbyType : byte { LobbyType.Default = 0, LobbyType.SqlLobby = 2, LobbyType.AsyncRandomLobby= 3 }

可用的大厅类型选项。大厅类型可能会在某些 Photon 版本中实现，且不会在旧服务器上可用。

- enum AuthModeOption { AuthModeOption.Auth, AuthModeOption.AuthOnce, AuthModeOption.AuthOnceWss }

认证模式的选项。从“classic”的在每个服务器上授权来 AuthOnce（在 NameServer 上）。

- enum CustomAuthenticationType : byte { CustomAuthenticationType.Custom = 0, CustomAuthenticationType.Steam = 1, CustomAuthenticationType.Facebook = 2, CustomAuthenticationType.Oculus = 3, CustomAuthenticationType.PlayStation = 4, CustomAuthenticationType.Xbox = 5, CustomAuthenticationType.None = byte.MaxValue }

与 Photon 一起使用的可选“Custom Authentication”服务选项。连接到 Photon 后使用 OpAuthenticate。

9.11.1 Enumeration Type Documentation | 枚举类型文档

9.11.1.1 enum AuthModeOption

认证模式的选项。从“classic”的在每个服务器上授权来 AuthOnce（在 NameServer 上）。

Enumerator | 枚举器

Auth | 认证

AuthOnce | 认证一次

AuthOnceWss

9.11.1.2 enum CustomAuthenticationType : byte

与 Photon 一起使用的可选“Custom Authentication”服务选项。连接到 Photon 后使用 OpAuthenticate。

Enumerator | 枚举

Custom 使用一个自定义的认证服务。目前唯一实施的选项。

Steam 通过 Steam 账户来对用户进行身份验证。设置相应认证值！

Facebook 通过 Facebook 账户来对用户进行身份验证。设置相应认证值！

Oculus 通过 Oculus 账户和口令来对用户进行身份验证。

PlayStation 通过 PSN 账户和口令来对用户进行身份验证。

Xbox 通过 Xbox 账户和 XSTS 口令来对用户进行身份验证。

None 禁用自定义认证。意味着不为连接提供任何 AuthenticationValues（更准确地说：OpAuthenticate）。

9.11.1.3 enum EventCaching : byte

轻量版-OpRaiseEvent 允许你缓存事件并自动将其发送给加入房间里的玩家。事件缓存每个事件代码和玩家：事件 100（例如！）可以每玩家存储一次。缓存事件可以被修改、替换和删除。

缓存只能结合 ReceiverGroup 的 Others 和 All 选项工作。

Enumerator | 枚举器

DoNotCache 默认值（不发送）。

MergeCache 将此事件的键与已缓存事件的合并。

ReplaceCache 用此事件的内容取代这个 EventCode 的事件缓存。

RemoveCache 从缓存中删除此事件（由 eventCode）。

AddToRoomCache 将事件添加到房间的缓存中。

AddToRoomCacheGlobal 为演员 0 将事件添加到缓存（在缓存中成为“globally owned”事件）。

RemoveFromRoomCache 从房间的缓存中移除拟合事件。

RemoveFromRoomCacheForActorsLeft 移除已经离开房间的玩家的事件（清理）。

SliceIncreaseIndex 增加片缓存的索引。

SliceSetIndex 设置缓存片的索引。你必须为其设置 RaiseEventOptions.CacheSliceIndex。

SlicePurgeIndex 用索引清除缓存片。从缓存中删除一个片。你必须为其设置 RaiseEventOptions.CacheSliceIndex。

SlicePurgeUpToIndex 清除指定索引以及低于该索引的所有缓存片。你必须为其设置

RaiseEventOptions.CacheSliceIndex。

9.11.1.4 enum JoinMode : byte

为 OpJoinRoom 和 OpJoinOrCreate 定义可能的值。它告诉服务器房间是否只能正常被加入、隐式地创建或在网络服务上被发现的回合制游戏。

这些值不是由游戏直接使用，但是会被隐性设置。

Enumerator | 枚举器

Default 定期加入。房间必须存在。

CreateIfNotExists 加入或创建房间，如果房间不存在则创建。例如在用于 OpJoinOrCreate 操作中时。

JoinOrRejoin 这个房间可能是内存不足，应装从基于回合制的网络服务中被加载（如果可能的话）。

RejoinOnly 只有重新连接将被允许。如果用户还没有在房间里，这将失败。

9.11.1.5 enum LobbyType : byte

可用的大厅类型选项。大厅类型可能会在某些 Photon 版本中实现，且不会在旧服务器上可用。

Enumerator | 枚举器

Default 这个大厅会被使用，除非另一个游戏大厅被游戏或 JoinRandom 方法定义。房间列表将被发送，且 JoinRandomRoom 可以通过匹配属性来刷选。

SqlLobby 这个大厅类型和默认大厅一样列出房间，但是 JoinRandom 有一个类似 SQL 的参数"where"语句来刷选。该语句允许更大、更少、或其他组合。

AsyncRandomLobby 该游戏大厅不会发送游戏列表。只被用于OpJoinRandomRoom。当只有不活跃的用户离开时它会保持房间可用一会儿。

9.11.1.6 enum MatchmakingMode : byte

OpJoinRandom 的匹配规则选项。

Enumerator | 枚举器

FillRoom 填充房间（第一时间），以尽可能快的把玩家聚集到一起。默认情况下，MaxPlayers > 0 是最合理的，且游戏只能以更多的玩家开始。

SerialMatching 以可用的房间顺序分发玩家，但需要考虑到过滤器。没有过滤器，房间得到均等的玩家。

RandomMatching 加入一个（完全）随机的房间。预期的属性必须匹配，但除此之外，任何可用的房间可能被选中。

9.11.1.7 enum PropertyTypeFlag : byte

"types of properties"的旗帜，在 OpGetProperties 中被用作过滤器。

Enumerator | 枚举器

None (0x00) 无属性类型的标志类型。

Game (0x01) 游戏附加属性的标志类型。

Actor (0x02) 演员相关属性的标志类型。

GameAndActor (0x01) 游戏和演员属性的标志类型。等价于’ Game’ 。

9.11.1.8 enum ReceiverGroup : byte

轻量版-OpRaiseEvent 让你选择在房间里的哪些演员应该接收事件。默认情况下，事件被

发送到“Others”，但你可以否决这个。

Enumerator | 枚举器

Others 默认值（不发送）。任何其他人都能得到我的事件。

All 每个在当前的房间人（包括这个玩家）将得到这个事件。

MasterClient 服务器仅将此事件发送到 actorNumber 最低的演员。“主客户端”没有特殊的权利，只是一个在这个房间里最长时间的客户端。

9.12 NetworkingPeer.cs 文件参考

C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking /Plugins/PhotonNetwork/NetworkingPeer.cs File Reference

Classes | 类

- class **NetworkingPeer**

实现用于 PUN 的 Photon 负载均衡。这个类被 PhotonNetwork 内部使用，且不打算作为公共 API。

Typedefs | 类型定义

- using Hashtable = ExitGames.Client.Photon.Hashtable
- using SupportClassPun = ExitGames.Client.Photon.SupportClass

Enumerations | 枚举

- enum ClientState {

ClientState.Uninitialized, ClientState.PeerCreated, ClientState.Queued,

ClientState.Authenticated,ClientState.JoinedLobby,

ClientState.DisconnectingFromMasterserver,ClientState.ConnectingToGameserver,

ClientState.ConnectedToGameserver,ClientState.Joining, ClientState.Joined,

ClientState.Leaving, ClientState.DisconnectingFromGameserver,

ClientState.ConnectingToMasterserver,

ClientState.QueuedComingFromGameserver, ClientState.Disconnecting,

ClientState.Disconnected,ClientState.ConnectedToMaster,

ClientState.ConnectingToNameServer, ClientState.ConnectedToNameServer,

ClientState.DisconnectingFromNameServer,ClientState.Authenticating }

详细连接/网络对等状态。PUN 在“幕后”实现了负载均衡和认证流程，所以一些状态将自动提前到一些跟踪状态。这些状态被注释为”（将改变）”。

- enum DisconnectCause {

DisconnectCause.DisconnectByServerUserLimit = StatusCode.DisconnectByServerUserLimit,

DisconnectCause.ExceptionOnConnect = StatusCode.ExceptionOnConnect,

DisconnectCause.DisconnectByServerTimeout = StatusCode.DisconnectByServer,

DisconnectCause.DisconnectByServerLogic = StatusCode.DisconnectByServerLogic,

DisconnectCause.Exception = StatusCode.Exception,

DisconnectCause.InvalidAuthentication = ErrorCode.InvalidAuthentication,

DisconnectCause.MaxCcuReached = ErrorCode.MaxCcuReached,

DisconnectCause.InvalidRegion = ErrorCode.InvalidRegion,

DisconnectCause.SecurityExceptionOnConnect =

StatusCode.SecurityExceptionOnConnect,

DisconnectCause.DisconnectByClientTimeout = StatusCode.TimeoutDisconnect,

DisconnectCause.InternalReceiveException = StatusCode.ExceptionOnReceive,

DisconnectCause.AuthenticationTicketExpired = 32753 }

总结断开连接的原因。用于：OnConnectionFail 和 OnFailedToConnectToPhoton。

- enum ServerConnection { ServerConnection.MasterServer,

ServerConnection.GameServer, ServerConnection.NameServer }

可用的服务器（类型）内部使用的字段：server。

9.12.1 Typedef Documentation | 类型定义文档

9.12.1.1 using Hashtable = ExitGames.Client.Photon.Hashtable

9.12.1.2 using SupportClassPun = ExitGames.Client.Photon.SupportClass

9.12.2 Enumeration Type Documentation | 枚举类型文档

9.12.2.1 enum ServerConnection

可用的服务器（类型）内部使用的字段：server。

Photon 使用 3 种不同的服务器角色：Name Server, Master Server and Game Server.

Enumerator | 枚举器

MasterServer 该服务器是匹配完成的地方，客户端可以在主服务器上的游戏大厅里获得房间列表。

GameServer 此服务器处理多个房间来执行和中继玩家之间的信息（在房间里）。

NameServer 此服务器最初用于获取特定区域的主服务器的地址（IP）。不用于 Photon 的 OnPremise（自托管）。

9.13 PhotonClasses.cs 文件参考

C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/PhotonClasses.cs File Reference | 文件参考

将较小的不需要自己的文件的类封装起来。

Classes | 类

- interface IPunObservable

定义 OnPhotonSerializeView 方法来更容易地正确实现可观察脚本。

- interface IPunCallbacks

这个接口被用于作为 PUN 所有回调函数的定义，除了 OnPhotonSerializeView。最好单独实现。

- interface IPunPrefabPool

定义了一个对象池必须要实现的所有方法，这样 PUN 才可以使用。

- class Photon.MonoBehaviour

这个类添加了 photonView 属性，当你的游戏仍然采用 networkView 时记录一个警告。

- class Photon.PunBehaviour

该类提供.photonView 属性和 PUN 能调用的所有回调/事件。重写那些你想使用的事件/方法。

- struct PhotonMessageInfo

关于一个特定的消息、RPC 或更新信息的容器类。

- class **PunEvent**

定义用于 PUN 的 Photon 事件代码。

- class PhotonStream

该容器被用于 OnPhotonSerializeView()中，用来要么提供 PhotonView 的输入数据，要么由你为其提供数据。

- class HelpURL

即将上线的 Unity5.1 版本 HelpURL 的空实现。该类只是为了字段兼容。

- class UnityEngine.SceneManagement.SceneManager

为老版本 Unity SceneManager 的最小化实现，直到 v5.2 版本。

- class SceneManagerHelper

- class WebRpcResponse

读取 WebRpc 的操作回应并提供对最常见值的便捷访问。

Namespaces | 命名空间

- package Photon

- package UnityEngine.SceneManagement

Typedefs

- using Hashtable = ExitGames.Client.Photon.Hashtable
- using SupportClassPun = ExitGames.Client.Photon.SupportClass
- using Photon.Hashtable = ExitGames.Client.Photon.Hashtable

9.13.1 Detailed Description | 详细描述

将较小的不需要自己的文件的类封装起来。

9.13.2 Typedef Documentation | 类型定义文档

9.13.2.1 using Hashtable = ExitGames.Client.Photon.Hashtable

9.13.2.2 using SupportClassPun = ExitGames.Client.Photon.SupportClass

9.14 PhotonHandler.cs 文件参考

C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/PhotonHandler.cs File Reference | 文件参考

Classes | 类

- class **PhotonHandler**

Monobehaviour 内部允许 Photon 运行 Update 循环。

Typedefs | 类型定义

- using Debug = UnityEngine.Debug

- using Hashtable = ExitGames.Client.Photon.Hashtable
- using SupportClassPun = ExitGames.Client.Photon.SupportClass

9.14.1 Typedef Documentation | 类型定义文档

9.14.1.1 using Debug = UnityEngine.Debug

9.14.1.2 using Hashtable = ExitGames.Client.Photon.Hashtable

9.14.1.3 using SupportClassPun = ExitGames.Client.Photon.SupportClass

9.15 PhotonLagSimulationGui.cs 文件参考

C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/PhotonLagSimulationGui.cs File Reference

Optional GUI 的一部分。

Classes | 类

- class PhotonLagSimulationGui

这个 MonoBehaviour 是一个 Photon 客户端的网络模拟功能的基本 GUI。它可以修改滞后（固定延迟），抖动（随机滞后）和数据包丢失。

9.15.1 Detailed Description | 详细描述

Optional GUI 的一部分。

9.16 PhotonNetwork.cs 文件参考

C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photo

n Unity Networking/Plugins/PhotonNetwork/PhotonNetwork.cs File Reference

Classes | 类

- class PhotonNetwork

使用 PhotonNetwork 插件的主类. 这是一个静态类。

Typedefs | 类型定义

- using Debug = UnityEngine.Debug
- using Hashtable = ExitGames.Client.Photon.Hashtable

9.16.1 Typedef Documentation | 类型定义文档

9.16.1.1 using Debug = UnityEngine.Debug

9.16.1.2 using Hashtable = ExitGames.Client.Photon.Hashtable

9.17 PhotonPlayer.cs 文件参考

C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/PhotonPlayer.cs File Reference

Classes | 类

- class PhotonPlayer

总结了一个房间内的“玩家”，由 actorID 来确定身份（在房间里）。

Typedefs | 类型定义

- using Hashtable = ExitGames.Client.Photon.Hashtable

9.17.1 Typedef Documentation | 类型定义文档

9.17.1.1 using Hashtable = ExitGames.Client.Photon.Hashtable

9.18 PhotonStatsGui.cs 文件参考

C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/PhotonStatsGui.cs File Reference

Optional GUI 的一部分。

Classes | 类

- class PhotonStatsGui

展示 Photon 连接的流量和健康统计的基础 GUI，通过快捷键 shift+tab 来开关。

9.18.1 Detailed Description | 详细描述

Optional GUI 的一部分。

9.19 PhotonStreamQueue.cs 文件参考

C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/PhotonStreamQueue.cs File Reference | PhotonStreamQueue.cs 文件参考

Classes | 类

- class PhotonStreamQueue

PhotonStreamQueue 帮助你在更高的频率获取对象状态，然后是由 PhotonNetwork.sendRate 决定的频率，再然后当 Serialize()被调用时马上发送所有获取到的状态。在接收端，你可以调用 Deserialize()，然后数据流将以记录的与序列化相同的顺序和 timeStep 推出接收的对象状态。

9.20 PhotonView.cs 文件参考

C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/PhotonView.cs File Reference | PhotonView.cs 文件参考

Classes | 类

- class PhotonView

PUN 的网络 NetworkView 替代类。和 NetworkView 一样的使用方法。

Enumerations | 枚举

- enum ViewSynchronization { ViewSynchronization.Off, ViewSynchronization.ReliableDeltaCompressed, ViewSynchronization.Unreliable, ViewSynchronization.UnreliableOnChange }
- enum OnSerializeTransform { OnSerializeTransform.OnlyPosition, OnSerializeTransform.OnlyRotation, OnSerializeTransform.OnlyScale,OnSerializeTransform.PositionAndRotation,

OnSerializeTransform.All }

- enum OnSerializeRigidBody { OnSerializeRigidBody.OnlyVelocity, OnSerializeRigidBody.OnlyAngularVelocity, OnSerializeRigidBody.All }
- enum OwnershipOption { OwnershipOption.Fixed, OwnershipOption.Takeover, OwnershipOption.Request }

定义了 PhotonView 的所有权转移是怎样被处理的选项。

9.20.1 Enumeration Type Documentation | 枚举类型文档

9.20.1.1 enum OnSerializeRigidBody

Enumerator | 枚举器

OnlyVelocity

OnlyAngularVelocity

All

9.20.1.2 enum OnSerializeTransform

Enumerator | 枚举器

OnlyPosition

OnlyRotation

OnlyScale

PositionAndRotation

All

9.20.1.3 enum OwnershipOption

定义了 PhotonView 的所有权转移是怎样被处理的选项。

此设置影响了 RequestOwnership 和 TransferOwnership 在运行时的工作方式。

Enumerator | 枚举

Fixed 所有权是固定的。实例化的对象粘着它们的创造者，场景对象永远属于主客户端。

Takeover 所有权可以从目前的所有者那里拿走而无法反对。

Request 所有权可以被 PhotonView.RequestOwnership 方法请求，但目前所有者必须要同意放弃所有权才行。目前所有者必须实现 IPunCallbacks.OnOwnershipRequest 来回应所有权请求。

9.20.1.4 enum ViewSynchronization

Enumerator | 枚举器

Off

ReliableDeltaCompressed

Unreliable

UnreliableOnChange

9.21 PingCloudRegions.cs 文件参考

C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/PingCloudRegions.cs File Reference | PingCloudRegions.cs 文件参考

Classes | 类

- class PingMonoEditor

使用来自 System.Net.Sockets 的 C# Socket 类 (Unity 通常做的那样).

- class PhotonPingManager

Typedefs | 类型定义

- using Debug = UnityEngine.Debug
- using SupportClassPun = ExitGames.Client.Photon.SupportClass

9.21.1 Typedef Documentation | 类型定义文档

9.21.1.1 using Debug = UnityEngine.Debug

9.21.1.2 using SupportClassPun = ExitGames.Client.Photon.SupportClass

9.22 Room.cs 文件参考

C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/Room.cs File Reference

Classes | 类

- class Room

这个类描述了 PUN 加入（或已加入）的房间。与 RoomInfo 的属性相反，房间属性是可以设置的，且你可以关闭或隐藏“你的”房间。

9.23 RoomInfo.cs 文件参考

C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photo

n Unity Networking/Plugins/PhotonNetwork/RoomInfo.cs File Reference

Classes | 类

- class RoomInfo

简化版的房间，只有房间列表和加入房间所必须的信息，用于游戏大厅内的列表。这些属性是可设置的（open、 MaxPlayers 等等）。

9.24 RPC.cs 文件参考

C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/RPC.cs File Reference

Classes | 类

- class PunRPC

RPC 属性不同的名称替换。用于标记方法作为可远程调用的。

9.24.1 Detailed Description | 详细描述

RPC 属性不同的名称替换。用于标记方法作为可远程调用的。

9.25 RpcIndexComponent.cs 文件参考

C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/RpcIndexComponent.cs File Reference | RpcIndexComponent.cs 文件参考

过时的.导入时在这里覆盖旧文件。

9.25.1 Detailed Description | 详细描述

过时的.导入时在这里覆盖旧文件。

9.26 ServerSettings.cs 文件参考

C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/ServerSettings.cs File Reference | ServerSettings.cs 文件参考

可编程对象定义一个服务器设置。一个实例被创建为 **PhotonServerSettings**。

Classes | 类

- class Region
- class ServerSettings

连接相关设置的集合，由 PhotonNetwork.ConnectUsingSettings 内部使用。

9.26.1 Detailed Description | 详细描述

可编程对象定义一个服务器设置。一个实例被创建为 **PhotonServerSettings**。

9.27 SocketWebTcp.cs 文件参考

C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/SocketWebTcp.cs File Reference | SocketWebTcp.cs 文件参考

9.28 PhotonAnimatorView.cs 文件参考

C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/Views/PhotonAnimatorView.cs File Reference

Classes | 类

- class PhotonAnimatorView

这个类帮助你同步 Mecanim 动画，只需添加该组件到你的游戏对象上，并确保 PhotonAnimatorView 被添加到观察组件的列表里。

- class PhotonAnimatorView.SynchronizedParameter
- class PhotonAnimatorView.SynchronizedLayer

9.29 PhotonRigidbody2DView.cs 文件参考

C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/Views/PhotonRigidbody2DView.cs File Reference

Classes | 类

- class PhotonRigidbody2DView

这个类帮助你同步 2d 物理刚体组件的速度。注意只有速度被同步，因为 Unity 的物理引擎的不确定性（例如，其结果在所有的电脑上不尽相同）-对象的实际位置可能在同步之外。如果你想要让该对象的位置在所有客户端上都一样，你也应该添加一个 PhotonTransformView

来同步位置。简单地添加该组件到你的游戏对象上，并确保 PhotonRigidbody2DView 被添加到观察组件的列表。

9.30 PhotonRigidbodyView.cs 文件参考

C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/Views/PhotonRigidbodyView.cs File Reference

Classes | 类

- class PhotonRigidbodyView

这个类帮助你同步物理刚体组件的速度。注意只有速度被同步，因为 Unity 的物理引擎的不确定性（例如，其结果在所有的电脑上不尽相同）-对象的实际位置可能在同步之外。如果你想要让该对象的位置在所有客户端上都一样，你也应该添加一个 PhotonTransformView 来同步位置。简单地添加该组件到你的游戏对象上，并确保 PhotonRigidbodyView 被添加到观察组件的列表。

9.31 PhotonTransformView.cs 文件参考

C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/Views/PhotonTransformView.cs File Reference

Classes | 类

- class PhotonTransformView

这个类帮助你同步游戏对象的 position、rotation 和 scale。它还给你许多不同的选择来使同步值显得平滑，即使当数据只是每秒发送几次。只需要简单地把该组件添加到你的游戏对象上，并确保 PhotonTransformView 被添加到观察组件的列表。

9.32 PhotonTransformViewPositionControl.cs 文件参考

C:/Dev/photon-sdk-dotnet/Unity/ PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/Views/PhotonTransformViewPositionControl.cs File Reference

Classes

- class PhotonTransformViewPositionControl

9.33 PhotonTransformViewPositionModel.cs 文件参考

C:/Dev/photon-sdk-dotnet/Unity/ PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/Views/PhotonTransformViewPositionModel.cs File Reference

Classes | 类

- class PhotonTransformViewPositionModel

9.34 PhotonTransformViewRotationControl.cs 文件参考

C:/Dev/photon-sdk-dotnet/Unity/ PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/Views/PhotonTransformViewRotationControl.cs File Reference | 文件参考

Classes | 类

- class PhotonTransformViewRotationControl

9.35 PhotonTransformViewRotationModel.cs 文件参考

C:/Dev/photon-sdk-dotnet/Unity/ PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/Views/PhotonTransformViewRotationModel.cs File Reference | 文件参考

Classes | 类

- class PhotonTransformViewRotationModel

9.36 PhotonTransformViewScaleControl.cs 文件参考

C:/Dev/photon-sdk-dotnet/Unity/ PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/Views/PhotonTransformViewScaleControl.c

s File Reference | 文件参考

Classes | 类

- class PhotonTransformViewScaleControl

9.37 PhotonTransformViewScaleModel.cs 文件参考

C:/Dev/photon-sdk-dotnet/Unity/ PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/Views/PhotonTransformViewScaleModel.cs

File Reference | 文件参考

Classes | 类

- class PhotonTransformViewScaleModel