# 游戏联机对战引擎

# 客户端 SDK

## 产品文档

腾讯云

# 文档目录

# 客户端 SDK
# SDK 下载

最近更新时间：2022-03-29 14:13:23

> 注意：
> 由于产品逻辑已无法满足游戏行业技术发展，游戏联机对战引擎 MGOBE 将于2022年6月1日下线，请您在2022年5月31日前完成服务迁移。

| 平台 | 更新时间 | 版本 | SDK下载 | 文档 |
| --- | --- | --- | --- | --- |
| 微信小程序/微信小游戏/QQ 小游戏/字节小游戏/百度小游戏 /OPPO 小游戏/vivo 小游戏/H5 小游戏（不限于以上平台，兼容 JavaScript SDK 都可以） | 2021/02/05 | v1.3.9 | 下载 | • 快速入门<br>• JS SDK 使用流程 |
| Unity SDK 更新 | 2021/03/03 | v1.3.3 | 下载 | • C# SDK 使用流程<br>• Unity 游戏项目 |

# JS SDK 使用流程

最近更新时间：2022-03-29 14:13:28

> **注意：**
> 由于产品逻辑已无法满足游戏行业技术发展，游戏联机对战引擎 MGOBE 将于2022年6月1日下线，请您在2022年5月31日前完成服务迁移。

## 操作场景

本文档指导您如何使用游戏联机对战引擎 JS SDK。

## 前提条件

- 已在游戏联机对战引擎控制台创建游戏，并开通联机对战服务，详情可参见 开通服务。
- 已获取游戏 gameID 和 secretKey，您可在游戏概览的基本信息里查看。JS SDK 需要对这两个参数进行校验。

## 操作步骤

### 设置 request 和 socket 域名

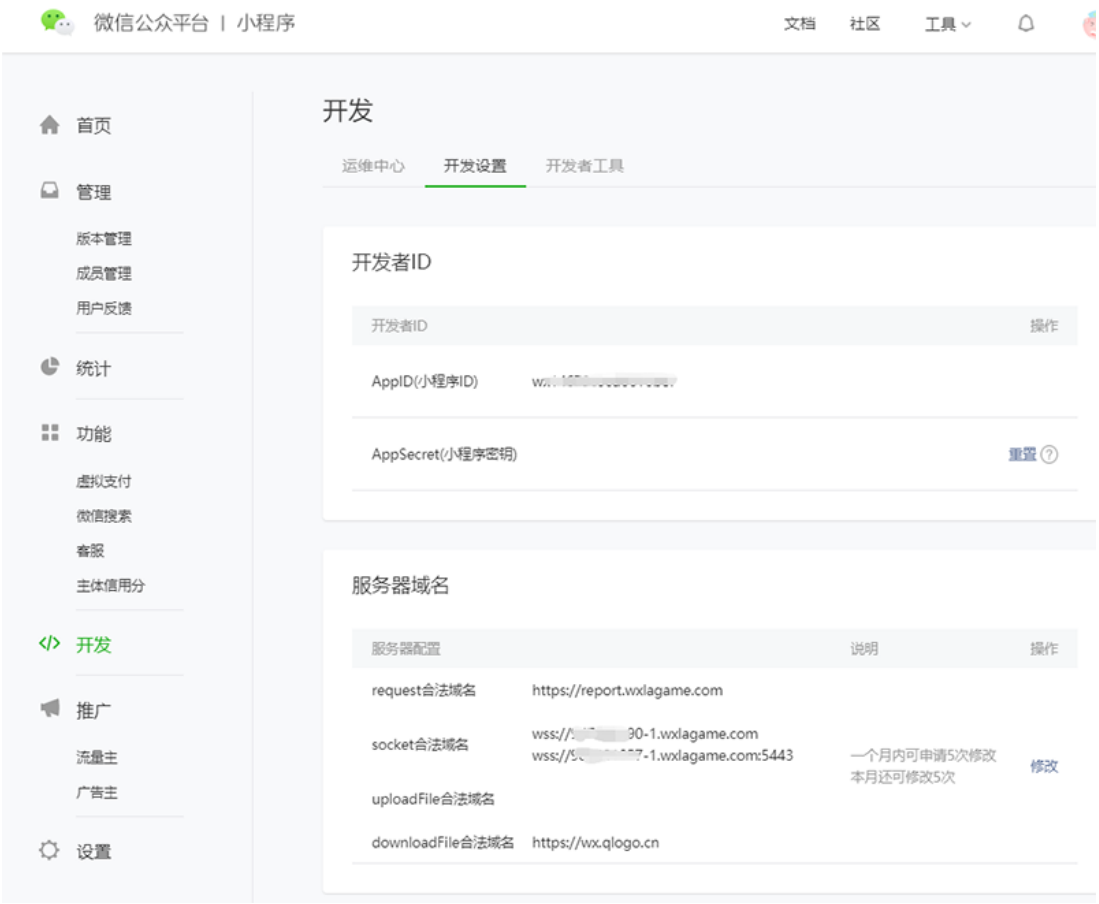
> **注意**
> 出于安全考虑，微信小程序/小游戏会限制请求域名，所有的 HTTPS、WebSocket、上传、下载请求域名都需要在 微信公众平台 进行配置。因此，在正式接入游戏联机对战引擎 JS SDK 前，需要您在微信公众平台配置合法域名。

1. 需要配置的域名包含一条 request 域名和两条 socket 域名记录。您在游戏联机对战引擎控制台上获取域名后，需要配置该域名的默认端口、 5443 端口两条记录。

```
// request 域名
report.wxlagame.com
// socket 域名
xxx.wxlagame.com
xxx.wxlagame.com:5443
```

2. 进入 游戏联机对战引擎控制台，将控制台获取的游戏域名信息复制保存。
3. 登录 微信公众平台，选择左侧菜单栏**开发>开发设置**。

4. 进入开发设置详情页，在 “服务器域名” 中添加合法域名记录。如下图所示：




## 导入 JS SDK

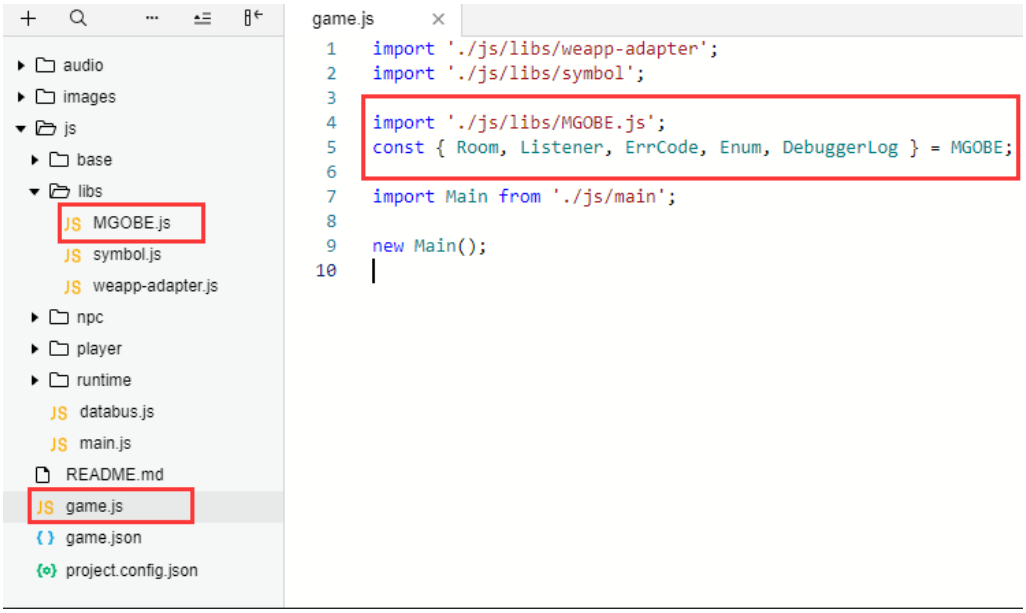
JS SDK 文件包含 MGOBE.js 和 MGOBE.d.ts，即源代码文件和定义文件。在 MGOBE.js 中，JS SDK 接口被全局注入到 Window 对象下。因此，只需要在使用 JS SDK 接口之前执行 MGOBE.js 文件即可。单击进入 JS SDK 下载 页面。

### 微信小游戏原生环境

在微信原生环境中，您只需将 MGOBE.js 放到项目下任意位置，在 game.js 中 import JS SDK 文件后即可使用 MGOBE 的方法。导入示例代码如下：

```
// 只需要在使用 MGOBE 之前 import 一次该文件
import "./js/libs/MGOBE.js";
// 直接使用 MGOBE
const { Room, Listener, ErrCode, ENUM, DebuggerLog } = MGOBE;
```

界面示例如下图所示：




您也可以使用 import/from、require 语法显式导入 MGOBE 模块。

import/from 代码示例如下：

```
// 使用 import/from
import * as MGOBE from "./js/libs/MGOBE.js";
const { Room, Listener, ErrCode, ENUM, DebuggerLog } = MGOBE;

// 或者
import { Room, Listener, ErrCode, ENUM, DebuggerLog } from "./js/libs/MGOBE.js";
```

require 代码示例如下：

```
// 使用 require
const MGOBE = require("./js/libs/MGOBE.js");
const { Room, Listener, ErrCode, ENUM, DebuggerLog } = MGOBE;

// 或者
const { Room, Listener, ErrCode, ENUM, DebuggerLog } = require("./js/libs/MGOBE.js");
```

**TypeScript 环境**

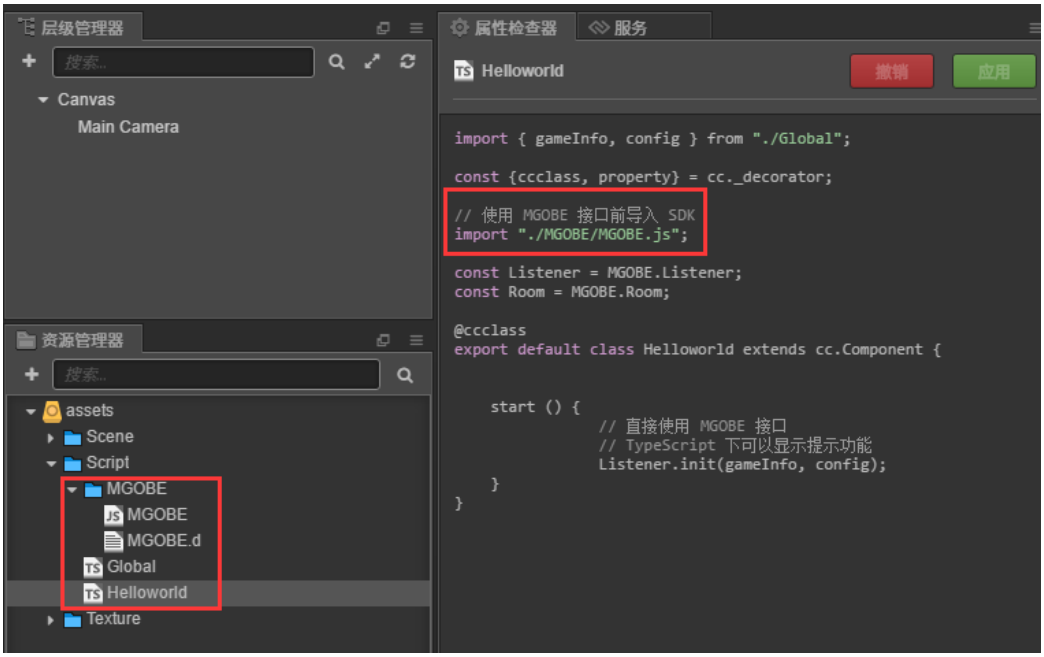
在 Laya、Cocos 等支持直接使用 TypeScript 进行开发的集成环境中，您可以使用 TypeScript 自带的 import 语法导入 JS SDK。由于 TypeScript 支持 .d.ts 定义文件，为了方便开发，您可以将 MGOBE.js 和 MGOBE.d.ts 一同复制到项目中，再调用 import 语句即可。以 Cocos Creator 和 LayaAir IDE 为例：

**Cocos Creator：**

```
import { gameInfo, config } from "./Global";

const {ccclass, property} = cc._decorator;

// 使用 MGOBE 接口前导入 JS SDK
import "./MGOBE/MGOBE.js";

const Listener = MGOBE.Listener;
const Room = MGOBE.Room;

@ccclass
```

```
export default class Helloworld extends cc.Component {


start () {
// 直接使用 MGOBE 接口
// TypeScript 下可以显示提示功能
Listener.init(gameInfo, config);
}
}
```

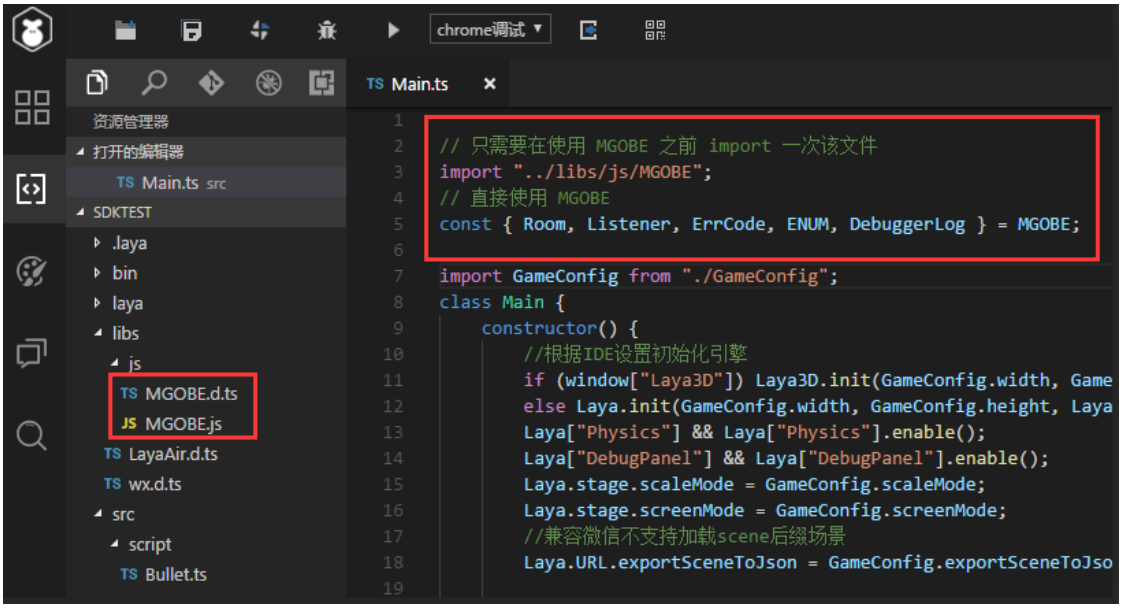
界面示例如下图所示：




LayaAir IDE：

```
// 只需要在使用 MGOBE 之前 import 一次该文件
import "../libs/js/MGOBE";
// 直接使用 MGOBE
const { Room, Listener, ErrCode, ENUM, DebuggerLog } = MGOBE;
```

界面示例如下图所示：




此外，在 LayaAir IDE 中，您也可以直接在 bin/index.js 中直接使用 loadLib 函数导入 MGOBE.js，让 JS SDK 文件先执行一遍即可。
在 TypeScript 环境中，您也可以使用 import/from 语法导入 MGOBE.js，但由于 TS 导入 .d.ts 的优先级高于导入 .js，所以您需要将 MGOBE.js 和 MGOBE.d.ts 文件放在不同文件夹，并使用 import/from 导入 .js 文件。

> ⚠ 注意
> 使用 import/from 语法方式导入 .js 将无法使用 .d.ts 提示。

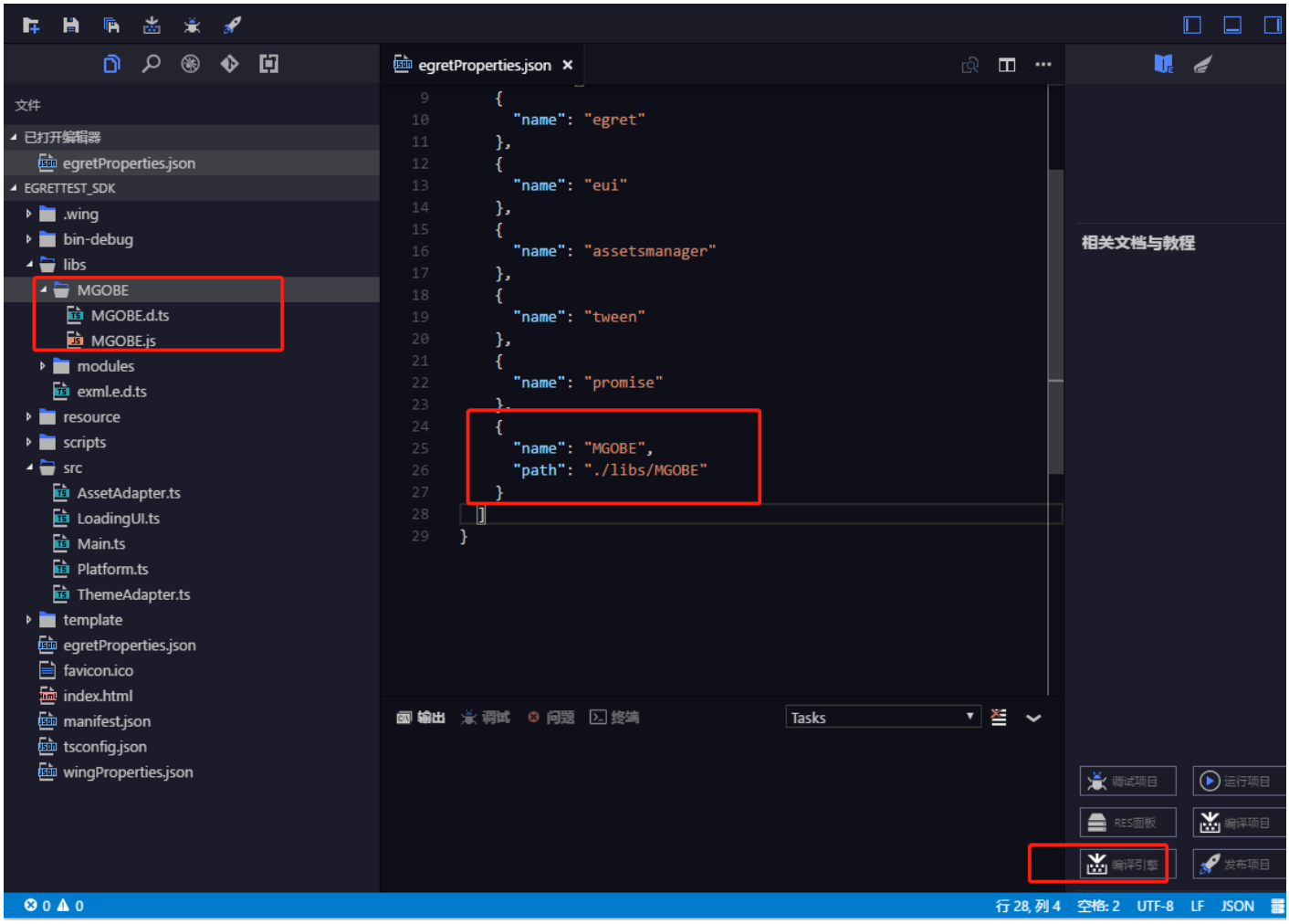
```
// import/from 导入 .js，无法使用 .d.ts 提示
import * as MGOBE from "../libs/js/MGOBE";
const { Room, Listener, ErrCode, ENUM, DebuggerLog } = MGOBE;
```

### Egret 环境

1. 使用 Egret Wing 打开项目，在 libs 文件夹下，创建 MGOBE 文件夹，将 MGOBE.js、MGOBE.d.ts 拷贝到 MGOBE 文件夹。
2. 编辑 egretProperties.json 文件，在 modules 数组中新增 MGOBE 库的描述。

```
{
"name": "MGOBE",
"path": "./libs/MGOBE"
}
```

3. 运行编译引擎，完成 MGOBE SDK 的导入工作。在项目代码中可以直接使用 MGOBE 对象。




### 初始化监听器 Listener

在使用 MGOBE API 接口时，主要用到 Listener 对象和 Room 对象。导入 JS SDK 后，需要先初始化 Listener 对象。

```
const gameInfo = {
openId: 'xxxxxx',
gameId: "xxxxxx",// 替换为控制台上的“游戏 ID”
secretKey: 'xxxxxx',// 替换为控制台上的“游戏 key”
};

const config = {
url: 'xxx.wxlagame.com',// 替换为控制台上的“域名”
reconnectMaxTimes: 5,
reconnectInterval: 1000,
resendInterval: 1000,
resendTimeout: 10000,
};

Listener.init(gameInfo, config, event => {
if (event.code === 0) {
// 初始化成功
// 继续在此添加代码
// ...
}
});
```

调用 Listener.init 时，需要传入游戏信息 gameInfo 和游戏配置 config 两个参数。

- gameInfo.gameId、gameInfo.secretKey 和 config.url 都需要根据控制台上的信息传入。
- gameInfo.openId 为玩家唯一 ID，例如，微信小游戏平台上的 OpenID。
- 其它字段由您自定义。

每个字段的具体含义您可参考 Listener 对象。

初始化成功后才能继续调用其他接口。因此，下文的 API 调用代码都应该在初始化回调函数内调用。

### 实例化 Room 对象

在开发游戏过程中的大部分业务接口都位于 Room 对象中。由于每个玩家只能加入一个房间，在游戏生命周期中可以实例化一个 Room 对象进行接口调用：

```
const room = new Room();
```

实例化 Room 后，可以调用 getMyRoom 接口来检查玩家是否已经加房，适用于应用重启后需要恢复玩家状态的场景。

```
// 查询玩家自己的房间
Room.getMyRoom(event => {
if (event.code === 0) {
// 设置房间信息到 room 实例
room.initRoom(event.data.roomInfo);
return console.log("玩家已在房间内：", event.data.roomInfo.name);
}

if (event.code === 20011) {
return console.log("玩家不在房间内");
}

return console.log("调用失败");
});
```

后续的创建房间、加房、匹配等接口调用直接利用 room 实例即可。

> ⚠ 注意
> getMyRoom、getRoomList、getRoomByRoomId 接口是 Room 对象的静态方法，您需要使用 Room.getMyRoom、Room.getRoomList、Room.getRoomByRoomId 进行调用。Room 的实例无法直接访问 getMyRoom、getRoomList、getRoomByRoomId。

### Room 接收广播

一个房间对象会有很多广播事件与其相关，例如该房间有新成员加入、房间属性变化、房间开始对战等广播。Room 实例需要在 Listener 中注册广播监听，并且实现广播的回调函数。

#### 注册广播监听

```
// 监听
Listener.add(room);
```

#### 设置广播回调函数

```
// 广播：房间有新玩家加入
room.onJoinRoom = event => console.log("新玩家加入", event.data.joinPlayerId);
// 广播：房间有玩家退出
room.onLeaveRoom = event => console.log("玩家退出", event.data.leavePlayerId);
```

```
// 广播：房间被解散
room.onDismissRoom = event => console.log("房间被解散");
// 其他广播
// ...
```

其他广播接口您可参考 Room 对象。

#### 移除监听

如果不想再接收该 Room 实例的广播，可以从 Listener 中移除：

```
// 移除监听
Listener.remove(room);
```

### 游戏对战

如果玩家已经加入房间，可以通过帧同步功能进行游戏对战。游戏过程中用到的接口有发送帧消息、帧广播，您可以利用这两个接口进行帧同步。帧广播的开始、结束需要使用房间的 startFrameSync、stopFrameSync 接口。

#### 开始帧同步

使用 room.startFrameSync 接口可以开启帧广播。房间内任意一个玩家成功调用该接口将导致全部玩家开始接收帧广播。

```
// 发起请求
room.startFrameSync({}, event => {
if (event.code === ErrCode.EC_OK) {
console.log("请求成功");
}
});
// 广播：开始帧同步
room.onStartFrameSync = event => console.log("开始帧同步");
```

#### 发送帧消息

玩家收到帧同步开始广播后可以发送帧消息，后台会将每个玩家的帧消息组合后广播给每个玩家。

```
const frame = { x: 100, y: 100, dir: 30, id: "xxxxxxxx" };
room.sendFrame({ data: frame }, event => {
if (event.code === ErrCode.EC_OK) {
console.log("发送成功");
}
});
```

#### 接收帧广播

您可设置 room.onRecvFrame 广播回调函数获得帧广播数据。

```
// 广播：收到帧消息
room.onRecvFrame = event => {
console.log("帧广播", event.data);
};
```

#### 停止帧同步

使用 room.stopFrameSync 接口可以停止帧广播。房间内任意一个玩家成功调用该接口将导致全部玩家停止接收帧广播。

```
// 发起请求
room.stopFrameSync({}, event => {
```

```
if (event.code === ErrCode.EC_OK) {
console.log("请求成功");
}
});
// 广播：停止帧同步
room.onStopFrameSync = event => console.log("停止帧同步");
```

# C# SDK 使用流程

最近更新时间：2022-03-29 14:13:36

> ⚠ 注意：
> 由于产品逻辑已无法满足游戏行业技术发展，游戏联机对战引擎 MGOBE 将于2022年6月1日下线，请您在2022年5月31日前完成服务迁移。

## 操作场景

本文档指导您如何使用游戏联机对战引擎 C# SDK。

## 前提条件

已在 Unity Editor 中根据指引导入 MGOBE，详情请参见 Unity 游戏项目。

### 初始化监听器 Listener

在使用 MGOBE API 接口时，主要用到 Listener 对象和 Room 对象。导入 SDK 后，需要先初始化 Listener 对象。

```
// 初始化监听器 Listener
static void Listener () {
var gameInfo = new GameInfoPara {
GameId = "xxxxxxxxxxxx",// 替换为控制台上的“游戏 ID”
OpenId = "xxxxxxxxxxxx", // 开发者定义的玩家唯一 ID
SecretKey = "xxxxxxxxxxxxx" // 替换为控制台上的“游戏 key”
};

var config = new ConfigPara {
Url = "xxx.wxlagame.com", // 替换为控制台上的“域名”
ReconnectMaxTimes = 5,
ReconnectInterval = 1000,
ResendInterval = 1000,
ResendTimeout = 10000,
};

Listener.Init (gameInfo, config, (ResponseEvent eve) => {
if (eve.Code == ErrCode.EcOk) {
// 初始化成功
// 继续在此添加代码
// ...
}
});
}
```

调用 Listener.Init 时，需要传入游戏信息 gameInfo 和游戏配置 config 两个参数。

- gameInfo.GameId、gameInfo.SecretKey 和 config.Url 都需要根据控制台上的信息传入。
- gameInfo.OpenId 为玩家唯一 ID，例如，微信小游戏平台上的 OpenID。
- 其它字段由您自定义。

每个字段的具体含义您可参考 Listener 对象。

初始化成功后才能继续调用其他接口。因此，下文的 API 调用代码都应该在初始化回调函数内调用。

### 实例化 Room 对象

在开发游戏过程中的大部分业务接口都位于 Room 对象中。由于每个玩家只能加入一个房间，在游戏生命周期中可以实例化一个 Room 对象进行接口调用：

```
var room = new Room (null);
```

实例化 Room 后，可以调用 getMyRoom 接口来检查玩家是否已经加房，适用于应用重启后需要恢复玩家状态的场景。

```
// 查询玩家自己的房间
Room.GetMyRoom (eve => {
if (eve.Code == 0) {
var data = (GetRoomByRoomIdRsp) eve.Data;
// 设置房间信息到 room 实例
room.InitRoom (data.RoomInfo);
Debug.Log ("玩家已在房间内：" + data.RoomInfo.Name);
return;
}

if (eve.Code == 20011) {
Debug.Log ("玩家不在房间内");
return;
}

Debug.Log ("调用失败");
});
```

后续的创建房间、加房、匹配等接口调用直接利用 room 实例即可。

> 说明
> GetMyRoom、GetRoomList、GetRoomByRoomId 接口是 Room 对象的静态方法，您需要使用 Room.GetMyRoom、Room.GetRoomList、Room.GetRoomByRoomId 进行调用。Room 的实例无法直接访问 GetMyRoom、GetRoomList、GetRoomByRoomId。

### Room 接收广播

一个房间对象会有很多广播事件与其相关，例如该房间有新成员加入、房间属性变化、房间开始对战等广播。Room 实例需要在 Listener 中注册广播监听，并且实现广播的回调函数。

#### 注册广播监听

```
// 监听
Listener.Add(room);
```

#### 设置广播回调函数

```
// 广播：房间有新玩家加入
room.OnJoinRoom = eve => {
var data = (JoinRoomBst) eve.Data;
Debug.Log ("新玩家加入" + data.JoinPlayerId);
};

// 广播：房间有玩家退出
room.OnLeaveRoom = eve => {
var data = (LeaveRoomBst) eve.Data;
Debug.Log ("玩家退出" + data.LeavePlayerId);
};

// 广播：房间被解散
room.OnDismissRoom = eve => {
```

```
var data = (JoinRoomBst) eve.Data;
Debug.Log ("房间被解散");
};

// 其他广播
// ...
```

其他广播接口您可参考 Room 对象。

**移除监听**

如果不想再接收该 Room 实例的广播，可以从 Listener 中移除：

```
// 移除监听
Listener.Remove (room);
```

### 创建房间

可以通过创建房间、加入房间、匹配等方式进入房间，进入房间后可以继续通过房间消息、帧同步、实时服务器相关接口进行游戏。
创建房间示例代码如下：

```
PlayerInfoPara playerInfoPara = new PlayerInfoPara
{
Name = "测试 Name",
CustomProfile = "测试 CustomProfile",
CustomPlayerStatus = 12345,
};

CreateRoomPara createRoomPara = new CreateRoomPara
{
RoomName = "测试 RoomName",
RoomType = "测试 RoomType",
MaxPlayers = 10,
IsPrivate = true,
CustomProperties = "测试 CustomProperties",
PlayerInfo = playerInfoPara,
};

room.CreateRoom(createRoomPara, eve =>
{
if (eve.Code == 0)
{
// 创建成功
Debug.LogFormat ("创建成功 {0}", eve.Data);

// 使用 room 调用其他 API，如 room.StartFrameSync
// ...

return;
}

if (eve.Code == 20010) {
Debug.Log ("玩家已在房间内");
return;
}
```

```
Debug.Log ("调用失败");
});
```

### 游戏对战

如果玩家已经加入房间，可以通过帧同步功能进行游戏对战。游戏过程中用到的接口有发送帧消息、帧广播，您可以利用这两个接口进行帧同步。帧广播的开始、结束需要使用房间的 startFrameSync、stopFrameSync 接口。

#### 开始帧同步

使用 room.startFrameSync 接口可以开启帧广播。房间内任意一个玩家成功调用该接口将导致全部玩家开始接收帧广播。

```
// 开始帧同步
static void startFrameSync () {
room.StartFrameSync (eve => {
if (eve.Code == ErrCode.EcOk) {
Debug.Log ("请求成功");
}
});
// 广播：开始帧同步
room.OnStartFrameSync = eve => Debug.Log ("开始帧同步");
}
```

#### 发送帧消息

玩家收到帧同步开始广播后可以发送帧消息，后台会将每个玩家的帧消息组合后广播给每个玩家。

```
// 用户自定义 Frame 类
[Serializable]
internal class Frame {
[SerializeField]
public int x;
[SerializeField]
public int y;
[SerializeField]
public string id;
}

// 发送帧消息
static void sendFrame () {
var frame = new Frame {
x = 1,
y = 1,
id = "xxxxxxxxx"
};
var para = new SendFramePara {
Data = frame.ToString ()
};
room.SendFrame (para, eve => {
if (eve.Code == ErrCode.EcOk) {
Debug.Log ("发送成功");
}
});
}
```

#### 接收帧广播

您可设置 room.OnRecvFrame 广播回调函数获得帧广播数据。

```
// 广播：收到帧消息
room.OnRecvFrame = eve => {
RecvFrameBst bst = (RecvFrameBst) eve.Data;
Debug.LogFormat ("帧广播: {0}", eve.Data);
};
```

**停止帧同步**

使用 room.stopFrameSync 接口可以停止帧广播。房间内任意一个玩家成功调用该接口将导致全部玩家停止接收帧广播。

```
room.StopFrameSync (eve => {
if (eve.Code == ErrCode.EcOk) {
Debug.Log ("请求成功");
}
});

// 广播：停止帧同步
room.OnStopFrameSync = eve => {
Debug.LogFormat ("停止帧同步: {0}", eve.Data);
};
```

> **注意**
> C# SDK 可正常使用但后续不再进行更新维护。

# API
# SDK 模板类型说明

最近更新时间：2022-03-29 14:16:05

> 注意：
> 由于产品逻辑已无法满足游戏行业技术发展，游戏联机对战引擎 MGOBE 将于2022年6月1日下线，请您在2022年5月31日前完成服务迁移。

SDK 在使用过程中会收到两类消息，即响应消息和广播消息。

- 响应消息指由客户端主动发起请求后，服务器返回的响应，消息类型为 ResponseEvent。
- 广播消息指服务器主动向客户端发起的消息，消息类型为 BroadcastEvent。

每种响应、广播的数据由下文“响应消息”、“广播消息”定义。
客户端向服务器发起请求后，可以设置响应回调函数，回调函数类型由下文“响应回调函数”定义。

### 响应消息 MGOBE.types.ResponseEvent

**MGOBE.types.ResponseEvent 的 TypeScript 定义如下：**

```
interface ResponseEvent<T> {
code: number;
msg: string;
seq: string;
data?: T;
}
```

**参数说明**

| 参数名 | 类型 | 描述 |
| --- | --- | --- |
| code | number | 消息错误码 |
| msg | string | 错误信息 |
| seq | string | 请求序列号 |
| data | object | 消息数据，由各消息回调接口定义 |

SDK 使用 TypeScript 的模板类型定义了 data 字段，具体的 data 结构由 API 各接口定义。

- 如

  MGOBE.types.ResponseEvent<MGOBE.types.CreateRoomRsp>

  定义了创建房间的响应消息，其中 data 的类型为

  MGOBE.types.CreateRoomRsp

  。
- 由于有些响应消息没有 data 内容，API 将使用

  MGOBE.types.ResponseEvent<null>

  来表示这类响应消息。

### 广播消息 MGOBE.types.BroadcastEvent

**MGOBE.types.BroadcastEvent 的 TypeScript 定义如下：**

```
interface BroadcastEvent<T> {
data?: T;
}
```

**参数说明**

| 参数名 | 类型 | 描述 |
| --- | --- | --- |
| data | object | 消息数据，由各消息回调接口定义 |

**广播消息是由服务端主动发起，只含有 data 一个字段。Room 对象各个广播接口中有具体 data 定义。**
**如**

MGOBE.types.BroadcastEvent<MGOBE.types.DismissRoomBst>

**定义了解散房间广播消息，其中 data 的类型为**
MGOBE.types.DismissRoomBst

。

### 响应回调函数 MGOBE.types.ReqCallback

**MGOBE.types.ReqCallback 的 TypeScript 定义如下：**

```
ReqCallback<T> = (event: MGOBE.types.ResponseEvent<T>) => any;
```

**响应函数是使用 SDK 接口向后台发起请求后，后台返回消息时的回调函数。上述定义表明该函数的入参是响应消息 MGOBE.types.ResponseEvent，函数体由您自定义。**

# Player 对象

最近更新时间：2022-03-29 14:16:11

> **注意：**
> 由于产品逻辑已无法满足游戏行业技术发展，游戏联机对战引擎 MGOBE 将于2022年6月1日下线，请您在2022年5月31日前完成服务迁移。

Player 对象为 MGOBE 的子属性，用于访问玩家的基本信息，例如玩家 ID、openId 等。

### 对象描述

玩家信息。

### 参数描述

| 属性名 | 类型 | 描述 |
| --- | --- | --- |
| id | string | 玩家 ID |
| openId | string | 玩家 openId |
| name | string | 玩家昵称 |
| teamId | string | 队伍 ID |
| customPlayerStatus | number | 自定义玩家状态 |
| customProfile | string | 自定义玩家属性 |
| commonNetworkState | MGOBE.types.NetworkState | 房间网络状态 |
| relayNetworkState | MGOBE.types.NetworkState | 帧同步网络状态 |

> **说明**
> - 该对象记录了玩家的基本信息，默认全部为空。成功初始化 Listener 后，ID、openId 属性将生效。
> - 玩家进入房间后，该对象的属性与 roomInfo.playerList 中当前玩家信息保持一致。
> - 玩家 ID 是 MGOBE 后台生成的 ID，openId 是您初始化时候使用的 ID。openId 只有初始化 Listener 的时候使用，其它接口的“玩家 ID”均指 MGOBE 后台生成的 ID。

# Listener 对象

最近更新时间：2022-03-29 14:16:17

> **注意：**
> 由于产品逻辑已无法满足游戏行业技术发展，游戏联机对战引擎 MGOBE 将于2022年6月1日下线，请您在2022年5月31日前完成服务迁移。

Listener 对象为 MGOBE 的子属性，该对象方法全为静态方法，不需要实例化。该对象主要用于给 Room 对象的实例绑定广播事件监听。

## init

### 接口描述

初始化监听器。

### 参数描述

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| gameInfo | MGOBE.types.GameInfoPara | 游戏信息 |
| config | MGOBE.types.ConfigPara | 游戏配置 |
| callback | MGOBE.types.ReqCallback<null> | 初始化回调函数 |

> **说明**
> - 该方法为静态方法。初始化 Listener 时需要传入 gameInfo 和 config 两个参数。
> - 初始化结果在 callback 中异步返回，错误码为0表示初始化成功。

### 返回值说明

无

### 使用示例

```
const gameInfo = {
gameId: "xxxxx",
openId: 'xxxxxx',
secretKey: 'xxxxxx',
};

const config = {
url: 'xxxxxxx.com',
reconnectMaxTimes: 5,
reconnectInterval: 1000,
resendInterval: 1000,
resendTimeout: 10000,
};

const room = new MGOBE.Room();

Listener.init(gameInfo, config, event => {
if (event.code === MGOBE.ErrCode.EC_OK) {
console.log("初始化成功");
// 初始化后才能添加监听
Listener.add(room);
} else {
console.log("初始化失败");
```

```
}
});
```

### add

**接口描述**

为 Room/Group 实例添加广播监听。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| entity | Room 或 Group | 需要监听的房间/队组对象 |

> **说明**
> - 该方法为静态方法。实例化 Room/Group 对象之后，需要通过该方法给 Room/Group 注册广播事件监听。
> - Listener 完成初始化之后才能添加监听。

**返回值说明**

无

**使用示例**

```
const room = new MGOBE.Room();
// 初始化后才能添加监听
Listener.add(room);
```

### remove

**接口描述**

为 Room/Group 实例移除广播监听。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| entity | Room 或 Group | 需要移除监听的房间/队组对象 |

> **说明**
> 该方法为静态方法。如果不再需要监听某个 Room/Group 对象的广播事件，可以通过该方法进行移除。

**返回值说明**

无

**使用示例**

```
const room = new MGOBE.Room();
// 监听
Listener.add(room);
// 移除监听
Listener.remove(room);
```

### clear

**接口描述**

移除全部 Room/Group 对象的广播监听。

**参数描述**

无

> 说明
> 该方法为静态方法。

**返回值说明**

无

**使用示例**

```
Listener.clear();
```

# Room 对象

## 概览

最近更新时间：2022-03-29 14:16:42

> 注意：
> 由于产品逻辑已无法满足游戏行业技术发展，游戏联机对战引擎 MGOBE 将于2022年6月1日下线，请您在2022年5月31日前完成服务迁移。

### Room 对象属性概览表

#### 构造器

| 名称 | 说明 |
| --- | --- |
| constructor | 构造器 |

#### 房间管理相关接口

| 名称 | 说明 |
| --- | --- |
| roomInfo | 房间信息 |
| networkState | 该属性为只读属性，用于获取客户端本地 SDK 网络状态 |
| initRoom | 初始化 Room 实例的房间信息，即更新 roomInfo 属性 |
| onUpdate | 房间信息更新回调接口 |
| createRoom | 创建房间 |
| createTeamRoom | 创建团队房间 |
| joinRoom | 加入房间 |
| joinTeamRoom | 加入团队房间 |
| leaveRoom | 退出房间 |
| dismissRoom | 解散房间 |
| changeRoom | 修改房间信息 |
| changeCustomPlayerStatus | 修改玩家自定义状态 |
| removePlayer | 移除房间内玩家 |
| getRoomDetail | 获取 Room 实例的房间信息 |
| getRoomByRoomId | 根据房间 ID 获取房间 |
| getMyRoom | 查询玩家所在的房间信息 |
| getRoomList | 获取房间列表 |
| onJoinRoom | 新玩家加入房间广播回调接口 |
| onLeaveRoom | 玩家退出房间广播回调接口 |
| onDismissRoom | 房间被解散广播回调接口 |
| onChangeRoom | 房主修改房间信息广播回调接口 |
| onRemovePlayer | 房间内玩家被移除广播回调接口 |
| onChangePlayerNetworkState | 房间内玩家网络状态变化广播回调接口 |
| onChangeCustomPlayerStatus | 玩家自定义状态变化广播回调接口 |

**匹配相关接口**

| 名称 | 说明 |
| --- | --- |
| matchPlayers | 玩家在线匹配 |
| matchGroup | 组队匹配 |
| matchRoom | 房间匹配 |
| cancelPlayerMatch | 取消玩家匹配 |
| onMatch | 匹配结束广播 |
| onCancelMatch | 取消匹配广播 |

**帧同步相关接口**

| 名称 | 说明 |
| --- | --- |
| startFrameSync | 开始帧同步 |
| stopFrameSync | 停止帧同步 |
| sendFrame | 发送帧同步数据 |
| requestFrame | 请求补帧 |
| onRecvFrame | 房间帧消息广播回调接口 |
| onStartFrameSync | 开始帧同步广播回调接口 |
| onStopFrameSync | 停止帧同步广播回调接口 |
| onAutoRequestFrameError | 自动补帧失败回调接口 |
| retryAutoRequestFrame | 重试自动补帧 |

**消息发送相关接口**

| 名称 | 说明 |
| --- | --- |
| sendToClient | 发送消息给房间内玩家 |
| onRecvFromClient | 收到房间内其他玩家消息广播回调接口 |
| sendToGameSvr | 发送消息给实时服务器 |
| onRecvFromGameSvr | 收到实时服务器消息广播回调接口 |

# 构造器

最近更新时间：2022-03-29 14:16:49

> **注意：**
> 由于产品逻辑已无法满足游戏行业技术发展，游戏联机对战引擎 MGOBE 将于2022年6月1日下线，请您在2022年5月31日前完成服务迁移。

## constructor

### 接口描述

构造器。

### 参数描述

| 参数名 | 类型/值 | 描述 | 可选 |
| --- | --- | --- | --- |
| roomInfo | MGOBE.types.RoomInfo | 房间信息 | 是 |

> **说明**
> - 实例化 Room 对象时可以传入一个 MGOBE.types.RoomInfo 对象 roomInfo，后续接口调用都将基于该 roomInfo，例如修改该房间的属性、接收该房间的广播。
> - 如果不传 roomInfo 参数，您可以通过直接调用 initRoom、createRoom、joinRoom 等方法获取 roomInfo。
> - Room 对象会自动维护内部的 roomInfo 属性保持最新，您可以直接通过访问该属性获得最新的房间信息。

### 返回值说明

无。

### 使用示例

```
// 示例1：不传 roomInfo 参数
// 该 Room 实例没有房间信息，room1.getRoomDetail 将查询玩家所在的房间
const room1 = new MGOBE.Room();

// 示例2：传 roomInfo 参数
// 该 Room 实例代表 ID 为 xxx 的房间，room2.getRoomDetail 将查询 xxx 房间信息
const roomInfo = {
id: "xxx",
// 其他字段
// ...
};
const room2 = new MGOBE.Room(roomInfo);
```

# 房间管理相关接口

最近更新时间：2022-03-29 14:16:57

> **注意：**
> 由于产品逻辑已无法满足游戏行业技术发展，游戏联机对战引擎 MGOBE 将于2022年6月1日下线，请您在2022年5月31日前完成服务迁移。

## roomInfo 属性

**对象描述**

房间信息。

**参数描述**

无

> **说明**
> roomInfo 为 Room 实例的属性，类型为 MGOBE.types.RoomInfo，调用 Room 相关的接口会导致该属性发生变化。

**使用示例**

```
// 打印 Room 实例的 roomInfo 属性
// 如需更新该属性，可以使用 getRoomDetail 方法
console.log(room.roomInfo);
```

## networkState 属性

**对象描述**

该属性为只读属性，用于获取客户端本地 SDK 网络状态。

**参数描述**

无

> **说明**
> - 该属性类型为 { COMMON: boolean, RELAY: boolean } 。COMMON 表示房间网络状态；RELAY 表示帧同步网络状态。为 true 时表示网络已连接，为 false 时表示网络未连接。
> - 该网络状态与玩家信息中的网络状态（Player.commonNetworkState/Player.relayNetworkState）概念不同，前者表示本地 socket 状态，后者表示玩家在 MGOBE 后台中的状态。
> - 本地 socket 网络状态变化时，onUpdate 将被触发。

**返回值说明**

无

**使用示例**

```
// 打印 Room 实例的 networkState 属性
console.log(room.networkState);

// 判断本地连接是否正常
if (room.networkState.COMMON) {
console.log("连接正常");
}
```

## initRoom

**接口描述**

初始化 Room 实例的房间信息，即更新 roomInfo 属性。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| roomInfo | MGOBE.types.RoomInfo 或 { id: string } | 初始化参数，id 表示房间 id |

> 说明
> - initRoom 会更新 Room 实例的 roomInfo，接受 MGOBE.types.RoomInfo 或 { id: string; } 类型的参数。
> - 如果参数为 MGOBE.types.RoomInfo 类型，SDK 将自动更新 WebSocket 连接。如果参数为 { id: string; } 类型，需要调用 getRoomDetail 或 joinRoom 方法，才能更新 WebSocket 连接，否则可能无法及时收到房间广播。
> - 如果不传参数，该方法将清空 Room 实例的 roomInfo 属性，此时调用 getRoomDetail 方法将查询玩家所在的房间。
> - 当玩家要加入指定 ID 房间时，需要使用该接口初始化 Room 实例的 roomInfo 属性，然后才能通过调用 joinRoom 方法加入该 Room 实例所代表的房间。

**返回值说明**

无

**使用示例**

```
const room = new Room();

// 示例1：不传 roomInfo 参数
// 该 Room 实例房间信息被清除，room.getRoomDetail 将查询玩家所在的房间
room.initRoom();

// 示例2：指定房间 ID
// 该 Room 实例代表 ID 为 xxx 的房间，room.getRoomDetail 将查询 xxx 房间信息
const roomInfo = { id: "xxx" };
room.initRoom(roomInfo);
```

## onUpdate

**接口描述**

房间信息更新回调接口。

**参数描述**

| 参数名 | 类型/值 | 描述 | 可选 |
| --- | --- | --- | --- |
| room | Room | 更新的 Room 实例 | 是 |

> 说明
> - onUpdate 表明 Room 实例的 roomInfo 信息发生变化，这种变化原因包括各种房间操作、房间广播、本地网络状态变化等。
> - 您可以在该接口中更新游戏画面，或者使用 networkState 属性判断网络状态。

**返回值说明**

无

**使用示例**

```
room.onUpdate = (_) => {
console.log(_ === room); // true, 参数 _ 等于 room
console.log("房间信息更新", room.roomInfo);
};
```

### createRoom

**接口描述**

创建房间。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| createRoomPara | MGOBE.types.CreateRoomPara | 创建房间参数 |
| callback | MGOBE.types.ReqCallback<MGOBE.types.CreateRoomRsp> | 响应回调函数 |

> 说明
> - createRoom 调用结果将在 callback 中异步返回。操作成功后，roomInfo 属性将更新。
> - 创建房间成功后，玩家自动进入该房间，因此无法继续调用 joinRoom、matchPlayers 等方法，可以利用房间 ID 邀请其他玩家进入该房间。

**返回值说明**

无

**使用示例**

```
const playerInfo = {
name: "Tom",
customPlayerStatus: 1,
customProfile: "https://xxx.com/icon.png",
};

const createRoomPara = {
roomName: "房间名",
maxPlayers: 4,
roomType: "2V2",
isPrivate: false,
customProperties: "WAIT",
playerInfo,
};

room.createRoom(createRoomPara, event => console.log(event));
```

### createTeamRoom

**接口描述**

创建团队房间。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| createTeamRoomPara | MGOBE.types.CreateTeamRoomPara | 创建团队房间参数 |
| callback | MGOBE.types.ReqCallback<MGOBE.types.CreateRoomRsp> | 响应回调函数 |

> 说明
> - createTeamRoom 调用结果将在 callback 中异步返回。操作成功后，roomInfo 属性将更新。
> - 创建房间成功后，玩家自动进入该房间，因此无法继续调用 joinRoom、matchPlayers 等方法。
> - 参数中的“房间最大玩家数量”要求能被“队伍数量”整除，创建成功后每个队伍的“队伍最小人数”为1，“队伍最大人数”为整除结果。

**返回值说明**

无

**使用示例**

```
const playerInfo = {
name: "Tom",
customPlayerStatus: 1,
customProfile: "https://xxx.com/icon.png",
};

const createTeamRoomPara = {
roomName: "房间名",
maxPlayers: 4,
roomType: "2V2",
isPrivate: false,
customProperties: "WAIT",
playerInfo,
teamNumber: 2,
};

room.createTeamRoom(createTeamRoomPara, event => console.log(event));
```

### joinRoom

**接口描述**

加入房间。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| joinRoomPara | MGOBE.types.JoinRoomPara | 加入房间参数 |
| callback | MGOBE.types.ReqCallback<MGOBE.types.JoinRoomRsp> | 响应回调函数 |

> 说明
> - joinRoom 调用结果将在 callback 中异步返回。
> - 该接口加入的房间是 Room 实例所代表的房间，如果该 Room 实例的 roomInfo 不存在 roomId，则需要使用 roomId 通过 initRoom 方法初始化 Room 实例。
> - 加房成功后，房间内全部成员（包含调用者）都会收到一条玩家加入房间广播 onJoinRoom，roomInfo 属性将更新。

**返回值说明**

无

**使用示例**

```
const playerInfo = {
name: "Tom",
```

```
customPlayerStatus: 1,
customProfile: "https://xxx.com/icon.png",
};

const joinRoomPara = {
playerInfo,
};

// 示例1：加入指定 ID 的房间
const room1 = new MGOBE.Room();
room1.initRoom({ id: "xxx" });
room1.joinRoom(joinRoomPara, event => console.log(event));

// 示例2：加入没有房间信息的房间
const room2 = new MGOBE.Room();
// 加房失败，找不到房间信息
room2.joinRoom(joinRoomPara, event => console.log(event));
```

## joinTeamRoom

**接口描述**

加入团队房间。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| joinTeamRoomPara | MGOBE.types.JoinTeamRoomPara | 加入团队房间参数 |
| callback | MGOBE.types.ReqCallback<MGOBE.types.JoinRoomRsp> | 响应回调函数 |

> 说明
> - joinTeamRoom 调用结果将在 callback 中异步返回。
> - 与 joinRoom 类似，该接口加入的房间是 Room 实例所代表的房间。teamId 为 roomInfo.teamList 中定义的队伍 ID。

**返回值说明**

无

**使用示例**

```
const playerInfo = {
name: "Tom",
customPlayerStatus: 1,
customProfile: "https://xxx.com/icon.png",
};

const joinTeamRoomPara = {
playerInfo,
teamId: "1",
};

room.joinTeamRoom(joinTeamRoomPara, event => console.log(event));
```

## leaveRoom

**接口描述**

退出房间。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| para | object | 预留参数，传{}即可 |
| callback | MGOBE.types.ReqCallback<MGOBE.types.LeaveRoomRsp> | 响应回调函数 |

> 说明
> - leaveRoom 调用结果将在 callback 中异步返回。退出成功后，房间内剩余成员（不含调用者）都会收到一条玩家退出房间广播 onLeaveRoom，roomInfo 属性将更新，roomInfo.playerList 中将没有该玩家信息。
> - 退房后，如果房间内还剩下其他玩家，则该 room 实例仍然代表退房前的房间，可以继续调用 room.initRoom() 清除房间信息。

**返回值说明**

无

**使用示例**

```
room.leaveRoom({}, event => {
if (event.code === 0) {
// 退房成功
console.log("退房成功", room.roomInfo.id);
// 可以使用 initRoom 清除 roomInfo
room.initRoom();
}
});
```

## dismissRoom

**接口描述**

解散房间。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| para | object | 预留参数，传{}即可 |
| callback | MGOBE.types.ReqCallback<MGOBE.types.DismissRoomRsp> | 响应回调函数 |

> 说明
> - dismissRoom 调用结果将在 callback 中异步返回。解散成功后，房间内全部成员（不含调用者）都会收到一条解散房间广播 onDismissRoom，roomInfo 属性将更新。
> - 只有房主有权限解散房间。

**返回值说明**

无

**使用示例**

```
room.dismissRoom({}, event => {
if (event.code === 0) {
```

```
console.log("解散成功");
}
});
```

### changeRoom

**接口描述**

修改房间信息。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| changeRoomPara | MGOBE.types.ChangeRoomPara | 修改房间参数 |
| callback | MGOBE.types.ReqCallback<MGOBE.types.ChangeRoomRsp> | 响应回调函数 |

> 说明
> - changeRoom 调用结果将在 callback 中异步返回。修改成功后，房间内全部成员都会收到一条修改房间广播 onChangeRoom，roomInfo 属性将更新。
> - 只有房主有权限修改房间

**返回值说明**

无

**使用示例**

```
const changeRoomPara = {
roomName: "房间名",
owner: "xxxxxx",
isPrivate: false,
customProperties: "xxxxxx",
};

room.changeRoom(changeRoomPara, event => console.log(event));
```

### changeCustomPlayerStatus

**接口描述**

修改玩家自定义状态。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| changeCustomPlayerStatusPara | MGOBE.types.ChangeCustomPlayerStatusPara | 修改玩家状态参数 |
| callback | MGOBE.types.ReqCallback<MGOBE.types.ChangeCustomPlayerStatusRsp> | 响应回调函数 |

> 说明
> - 修改玩家状态是修改 PlayerInfo 中的 customPlayerStatus 字段，玩家的状态由您自定义。
> - 修改成功后，房间内全部成员都会收到一条修改玩家状态广播 onChangeCustomPlayerStatus，roomInfo 属性将更新。
> - 每个玩家只能修改自己的状态，调用结果将在 callback 中异步返回。

**返回值说明**

**无**

**使用示例**

```
const changeCustomPlayerStatusPara = {
customPlayerStatus: 2,
};

room.changeCustomPlayerStatus(changeCustomPlayerStatusPara, event => console.log(event));
```

## changeRoomPlayerProfile

**接口描述**

修改玩家自定义属性。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| changeRoomPlayerProfilePara | MGOBE.types.ChangeRoomPlayerProfilePara | 修改玩家属性参数 |
| callback | MGOBE.types.ReqCallback<MGOBE.types.ChangeRoomPlayerProfileRsp> | 响应回调函数 |

**说明**

- 修改玩家属性是修改 PlayerInfo 中的 customProfile 字段，玩家的属性由您自定义。
- 修改成功后，房间内全部成员都会收到一条修改玩家属性广播 onChangeRoomPlayerProfile，roomInfo 属性将更新。
- 每个玩家只能修改自己的属性，调用结果将在 callback 中异步返回。

**返回值说明**

**无**

**使用示例**

```
const changeRoomPlayerProfilePara = {
customProfile: "{name: 'name_example'}",
};

room.changeRoomPlayerProfile(changeRoomPlayerProfilePara, event => console.log(event));
```

## removePlayer

**接口描述**

移除房间内玩家。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| removePlayerPara | MGOBE.types.RemovePlayerPara | 移除房间内玩家参数 |
| callback | MGOBE.types.ReqCallback<MGOBE.types.RemovePlayerRsp> | 响应回调函数 |

**说明**

- 调用结果将在 callback 中异步返回。移除玩家成功后，房间内全部成员都会收到一条移除玩家广播 onRemovePlayer，roomInfo 属性将更新。
- 只有房主有权限移除其他玩家

**返回值说明**

无

**使用示例**

```
const removePlayerPara = {
removePlayerId: "xxxxxx",
};

room.removePlayer(removePlayerPara, event => console.log(event));
```

## getRoomDetail

**接口描述**

获取 Room 实例的房间信息。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| callback | MGOBE.types.ReqCallback<MGOBE.types.GetRoomByRoomIdRsp> | 响应回调函数 |

> 说明
> - 该接口获取的是 Room 实例的房间信息，调用结果将在 callback 中异步返回。
> - 如果该 Room 实例中的 roomInfo 属性没有 ID，该接口将查询玩家所在的房间。
> - 如果 roomInfo 属性含有 ID，则查询该 ID 对应的房间信息。
> - 操作成功后，roomInfo 属性将更新。
> - 如果需要获取指定 ID 的房间信息，可以使用 getRoomByRoomId 方法。

**返回值说明**

无

**使用示例**

```
// 示例1：查询 room 实例的信息
room.getRoomDetail(event => {
if (event.code === 0) {
console.log("房间名", event.data.roomInfo.name);
}
});

// 示例2：查询玩家所在房间信息
room.initRoom();// 或者 room.initRoom({ id: "" });
room.getRoomDetail(event => {
if (event.code === 0) {
console.log("房间名", event.data.roomInfo.name);
}
});
```

## getRoomByRoomId

**接口描述**

根据房间 ID 获取房间。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| getRoomByRoomIdPara | MGOBE.types.GetRoomByRoomIdPara | 获取房间参数 |
| callback | MGOBE.types.ReqCallback<MGOBE.types.GetRoomByRoomIdRsp> | 响应回调函数 |

说明
- 调用结果将在 callback 中异步返回。
- 该接口为 Room 的静态方法，只能通过 Room.getRoomByRoomId 方式调用，Room 实例无法直接访问该方法。
- 如果参数中的 roomId 为空字符串，将查询玩家所在的房间。

**返回值说明**

无

**使用示例**

```
// 示例1：查询指定房间id的信息
const getRoomByRoomIdPara1 = {
roomId: "800000",
};
MGOBE.Room.getRoomByRoomId(getRoomByRoomIdPara1, event => console.log(event));

// 示例2：查询玩家所在房间信息
const getRoomByRoomIdPara2 = {
roomId: "",
};
MGOBE.Room.getRoomByRoomId(getRoomByRoomIdPara2, event => console.log(event));
```

### getMyRoom

**接口描述**

查询玩家所在的房间信息。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| callback | MGOBE.types.ReqCallback<MGOBE.types.GetRoomByRoomIdRsp> | 响应回调函数 |

说明
- 调用结果将在 callback 中异步返回。
- 该接口为 Room 的静态方法，只能通过 Room.getMyRoom 方式调用，Room 实例无法直接访问该方法。

**返回值说明**

无

**使用示例**

```
MGOBE.Room.getMyRoom(event => {
if (event.code === 0) {
console.log("玩家在房间内", event.data.roomInfo);
const room = new MGOBE.Room(event.data.roomInfo);
}
});
```

## getRoomList

**接口描述**

获取房间列表。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| getRoomListPara | MGOBE.types.GetRoomListPara | 获取房间列表参数 |
| callback | MGOBE.types.ReqCallback<MGOBE.types.GetRoomListRsp> | 响应回调函数 |

> 说明
> - 调用结果将在 callback 中异步返回。
> - 该接口为 Room 的静态方法，只能通过 Room.getRoomList 方式调用，Room 实例无法直接访问该方法。

**返回值说明**

无

**使用示例**

```
const getRoomListPara = {
pageNo: 1,
pageSize: 10,
};

// 不要使用 room.getRoomList
// 直接使用 Room 对象
MGOBE.Room.getRoomList(getRoomListPara, event => console.log(event));
```

## onJoinRoom

**接口描述**

新玩家加入房间广播回调接口。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| event | MGOBE.types.BroadcastEvent<MGOBE.types.JoinRoomBst> | 回调参数 |

> 说明
> onJoinRoom 广播表示该房间有新玩家加入。房间内全部成员都会收到该广播。

**返回值说明**

无

**使用示例**

```
room.onJoinRoom = event => console.log("新玩家加入", event.data.joinPlayerId);
```

## onLeaveRoom

**接口描述**

玩家退出房间广播回调接口。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| event | MGOBE.types.BroadcastEvent<MGOBE.types.LeaveRoomBst> | 回调参数 |

> 说明
> onLeaveRoom 广播表示该房间有玩家退出。房间内全部成员都会收到该广播。

**返回值说明**

无

**使用示例**

```
room.onLeaveRoom = event => console.log("玩家退出", event.data.leavePlayerId);
```

## onDismissRoom

**接口描述**

房间被解散广播回调接口。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| event | MGOBE.types.BroadcastEvent<MGOBE.types.DismissRoomBst> | 回调参数 |

> 说明
> onDismissRoom 广播表示房主解散了该房间。房间内全部成员都会收到该广播。

**返回值说明**

无

**使用示例**

```
room.onDismissRoom = event => console.log("房间已被房主解散");
```

## onChangeRoom

**接口描述**

房主修改房间信息广播回调接口。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| event | MGOBE.types.BroadcastEvent<MGOBE.types.ChangeRoomBst> | 回调参数 |

> 说明
> onChangeRoom 广播表示房主修改了该房间属性。房间内全部成员都会收到该广播。

**返回值说明**

无

**使用示例**

```
room.onChangeRoom = event => console.log("房间属性变更", event.data.roomInfo);
```

### onRemovePlayer

**接口描述**

房间内玩家被移除广播回调接口。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| event | MGOBE.types.BroadcastEvent<MGOBE.types.RemovePlayerBst> | 回调参数 |

> 说明
> onRemovePlayer 广播表示有玩家被房主移除。房间内全部成员都会收到该广播。

**返回值说明**

无

**使用示例**

```
room.onRemovePlayer = event => console.log("玩家被移除", event.data.removePlayerId);
```

### onChangePlayerNetworkState

**接口描述**

房间内玩家网络状态变化广播回调接口。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| event | MGOBE.types.BroadcastEvent<MGOBE.types.ChangePlayerNetworkStateBst> | 回调参数 |

> 说明
> - onChangePlayerNetworkState 广播表示 ID 为 changePlayerId 的玩家网络状态发生变化。
> - 玩家在房间中、帧同步中的网络变化都会触发该广播，因此 networkState 将有四种情况，分别表示房间中上线、房间中下线、帧同步中上线、帧同步中下线。

**返回值说明**

无

**使用示例**

```
room.onChangePlayerNetworkState = event => {
if (event.data.networkState === MGOBE.ENUM.NetworkState.COMMON_OFFLINE)
console.log("玩家下线");
};
```

### onChangeCustomPlayerStatus

**接口描述**

玩家自定义状态变化广播回调接口。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| event | MGOBE.types.BroadcastEvent<MGOBE.types.ChangeCustomPlayerStatusBst> | 回调参数 |

> 说明
> onChangeCustomPlayerStatus 广播表示房间内 ID 为 changePlayerId 的玩家状态发生变化。玩家状态由您自定义。

**返回值说明**

无

**使用示例**

```
room.onChangeCustomPlayerStatus = event => {
console.log("玩家状态变化", event.data.changePlayerId);
};
```

## onChangeRoomPlayerProfile

**接口描述**

玩家自定义属性变化广播回调接口。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| event | MGOBE.types.BroadcastEvent<MGOBE.types.ChangeRoomPlayerProfileBst> | 回调参数 |

> 说明
> onChangeRoomPlayerProfile 广播表示房间内 ID 为 changePlayerId 的玩家状态发生变化。玩家状态由您自定义。

**返回值说明**

无

**使用示例**

```
room.onChangeRoomPlayerProfile = event => {
console.log("玩家属性变化", event.data.changePlayerId);
};
```

# 匹配相关接口

最近更新时间：2022-03-29 14:17:07

> **注意：**
> 由于产品逻辑已无法满足游戏行业技术发展，游戏联机对战引擎 MGOBE 将于2022年6月1日下线，请您在2022年5月31日前完成服务迁移。

## matchPlayers

**接口描述**

玩家在线匹配。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| matchPlayersPara | MGOBE.types.MatchPlayersPara | 玩家匹配参数 |
| callback | MGOBE.types.ReqCallback<MGOBE.types.MatchPlayersRsp> | 响应回调函数 |

> **说明**
> - 调用该接口后，将发起玩家在线匹配，callback 将异步返回调用结果。返回码为0表示调用成功。
> - 调用成功后，Room.onMatch、Room.onCancelMatch 将回调匹配结果。
> - 该接口需要与匹配规则配合使用，因此，匹配超时时间由您在匹配规则中定义。您需要在控制台上创建匹配，得到匹配 Code 作为该方法的参数 matchCode。
> - matchPlayersPara.playerInfo 中的 matchAttributes 数组对应匹配规则中定义的 playerAttributes，playerAttributes 的每一种属性都要填入 matchAttributes 中，name 表示属性名，value 表示玩家该属性的值。

**返回值说明**

无

**使用示例**

```
const playerInfo = {
name: "Tom",
customPlayerStatus: 1,
customProfile: "https://xxx.com/icon.png",
matchAttributes: [{
name: "skill1",
value: 99,
}]
};

const matchPlayersPara = {
playerInfo,
matchCode: "play-xxx",
};

// 发起匹配
room.matchPlayers(matchPlayersPara, event => {
if (event.code === 0) {
console.log("请求成功");
} else {
console.log("请求失败", event.code);
}
```

```
});

// 监听匹配结果
Room.onMatch = (event) => {

if (event.data.errCode === MGOBE.ErrCode.EC_OK) {
console.log("组队匹配成功");
room.initRoom(event.data.roomInfo);
return;
}

// 匹配失败
// ...
};
```

### matchGroup

**接口描述**

组队匹配。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| matchGroupPara | MGOBE.types.MatchGroupPara | 玩家匹配参数 |
| callback | MGOBE.types.ReqCallback<MGOBE.types.MatchGroupRsp> | 响应回调函数 |

**说明**

- 调用该接口后，将以 playerInfoList 为小组一起进行匹配，callback 将异步返回调用结果。
- 调用成功后，Room.onMatch、Room.onCancelMatch 将回调匹配结果。小组成员都可以通过 cancelPlayerMatch 取消匹配。
- 匹配成功后，小组成员都会进入同一个房间，同一个队伍。
- 该接口需要与匹配规则配合使用，匹配超时时间由您在匹配规则中定义。您需要在控制台上创建匹配，得到匹配 Code 作为该方法的参数 matchCode。
- 根据匹配规则的不同，房间内的队伍可能包含多个小组。例如4V4的匹配（两个队伍），如果小组成员数为2，那么同一个队伍将由两个小组组成。

**返回值说明**

无

**使用示例**

```
const playerInfo1 = {
id: "playerId1",
name: "Tom",
customPlayerStatus: 1,
customProfile: "https://xxx.com/icon.png",
matchAttributes: [{
name: "skill1",
value: 99,
}]
};

const playerInfo2 = {
id: "playerId2",
name: "Jack",
customPlayerStatus: 1,
```

```
customProfile: "https://xxx.com/icon.png",
matchAttributes: [{
name: "skill1",
value: 99,
}]
};

const matchGroupPara = {
playerInfoList: [playerInfo1, playerInfo2],
matchCode: "match-xxx",
};

// 发起匹配
room.matchGroup(matchGroupPara, event => {
if (event.code === 0) {
console.log("发起匹配成功");
} else {
console.log("发起匹配失败");
}
});

// 监听匹配结果
Room.onMatch = (event) => {

if (event.data.errCode === MGOBE.ErrCode.EC_OK) {
console.log("组队匹配成功");
room.initRoom(event.data.roomInfo);
return;
}

// 匹配失败
// ...
};
```

### matchRoom

**接口描述**

房间匹配。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| matchRoomPara | MGOBE.types.MatchRoomPara | 房间匹配参数 |
| callback | MGOBE.types.ReqCallback<MGOBE.types.MatchRoomSimpleRsp> | 响应回调函数 |

> **说明**
> - 调用该接口后将发起房间匹配，匹配结果将在 callback 中异步返回。操作成功后，Room 对象内部 roomInfo 属性将更新。
> - 房间匹配指按照传入的参数（maxPlayers、roomType）搜索现存的房间：
>   - 如果已经存在房间属性 maxPlayers、roomType 对应相同的房间，同时满足房间允许加入（isForbidJoin 为 false）、房间为非私有（isPrivate 为 false）、房间人数未满等条件，则将调用者加入到该房间，并且房间内成员（不含调用者）会收到 onJoinRoom 广播。
>   - 如果不存在这样的房间，会直接使用 maxPlayers、roomType 这两个参数作为房间属性为调用者创建一个新房间。

**返回值说明**

**无**

**使用示例**

```
const playerInfo = {
name: "Tom",
customPlayerStatus: 1,
customProfile: "https://xxx.com/icon.png",
};

const matchRoomPara = {
playerInfo,
maxPlayers: 5,
roomType: "1",
};

room.matchRoom(matchRoomPara, event => {
if (event.code !== 0) {
console.log("匹配失败", event.code);
}
});
```

## cancelPlayerMatch

**接口描述**

取消玩家匹配。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| cancelMatchPara | MGOBE.types.CancelPlayerMatchPara | 取消匹配参数 |
| callback | MGOBE.types.ReqCallback<MGOBE.types.CancelPlayerMatchRsp> | 响应回调函数 |

> **说明**
> - 该接口作用是取消匹配请求，即 matchPlayers、matchGroup 请求。调用结果将在 callback 中异步返回。
> - cancelMatchPara.matchType 需要设置为 MGOBE.ENUM.MatchType.PLAYER_COMPLEX。

**返回值说明**

**无**

**使用示例**

```
const cancelMatchPara = {
matchType: MGOBE.ENUM.MatchType.PLAYER_COMPLEX,
};

room.cancelPlayerMatch(cancelMatchPara, event => console.log(event));
```

## onMatch

**接口描述**

匹配结束广播。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| event | MGOBE.types.BroadcastEvent<MGOBE.types.MatchGroupBst> | 回调参数 |

> 说明
> - onMatch 广播表示组队匹配结束。匹配成功或者匹配超时后，全部小组成员都会收到该广播。
> - 使用 event.data.errCode 判断是否匹配成功。
> - 注意该方法为 Room 类的静态方法。

**返回值说明**

无

**使用示例**

```
Room.onMatch = (event) =&gt; {

if (event.data.errCode === MGOBE.ErrCode.EC_OK) {
console.log("组队匹配成功");
// 需要更新 room 实例信息
room.initRoom(event.data.roomInfo);
return;
}

// 匹配超时
// ...
};
```

## onCancelMatch

**接口描述**

取消匹配广播。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| event | MGOBE.types.BroadcastEvent<MGOBE.types.CancelMatchBst> | 回调参数 |

> 说明
> - onCancelMatch 广播表示成功取消 matchPlayers、matchGroup。调用者或全部小组成员都将收到该广播。
> - 注意该方法为 Room 类的静态方法。

**返回值说明**

无

**使用示例**

```
Room.onCancelMatch = (event) => {
console.log("组队匹配已取消");
};
```

# 帧同步相关接口

最近更新时间：2022-03-29 14:17:14

> **注意：**
> 由于产品逻辑已无法满足游戏行业技术发展，游戏联机对战引擎 MGOBE 将于2022年6月1日下线，请您在2022年5月31日前完成服务迁移。

## startFrameSync

**接口描述**

开始帧同步。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| para | object | 预留参数，传{}即可 |
| callback | MGOBE.types.ReqCallback<MGOBE.types.StartFrameSyncRsp> | 响应回调函数 |

> **说明**
> - 调用结果将在 callback 中异步返回。调用成功后房间内全部成员将收到 onStartFrameSync 广播。该接口会修改房间帧同步状态为“已开始帧同步”。
> - 房间内任意一个玩家成功调用该接口将导致全部玩家开始接收帧广播。

**返回值说明**

无

**使用示例**

```
room.startFrameSync({}, event => {
if (event.code === 0) {
console.log("开始帧同步成功");
}
});
```

## stopFrameSync

**接口描述**

停止帧同步。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| para | object | 预留参数，传{}即可 |
| callback | MGOBE.types.ReqCallback<MGOBE.types.StoptFrameSyncRsp> | 响应回调函数 |

> **说明**
> - 调用结果将在 callback 中异步返回。调用成功后房间内全部成员将收到 onStopFrameSync 广播。该接口会修改房间帧同步状态为“已停止帧同步”。
> - 房间内任意一个玩家成功调用该接口将导致全部玩家停止接收帧广播。

**返回值说明**

无

**使用示例**

```
room.stopFrameSync({}, event => console.log(event));
```

## sendFrame

**接口描述**

发送帧同步数据。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| sendFramePara | MGOBE.types.SendFramePara | 发送帧同步数据参数 |
| callback | MGOBE.types.ReqCallback<MGOBE.types.SendFrameRsp> | 响应回调函数 |

> 说明
> - 帧数据内容 data 类型为普通 object，由您自定义，目前支持最大长度不超过1k。
> - 后台将集合全部玩家的帧数据，并以一定时间间隔（由房间帧率定义）通过 onRecvFrame 广播给各客户端。调用结果将在 callback 中异步返回。
> - 只有房间处于“已开始帧同步”状态才能调用该接口。

**返回值说明**

无

**使用示例**

```
const frame = {cmd: "xxxxxxxx", id: "xxxxxxxx" };
const sendFramePara = { data: frame };
room.sendFrame(sendFramePara, event => console.log(event));
```

## requestFrame

**接口描述**

请求补帧。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| requestFramePara | MGOBE.types.RequestFramePara | 请求补帧参数 |
| callback | MGOBE.types.ReqCallback<MGOBE.types.RequestFrameRsp> | 响应回调函数 |

> 说明
> - 调用结果将在 callback 中异步返回。

**返回值说明**

无

**使用示例**

```
const requestFramePara = {
beginFrameId: 100,
endFrameId: 120,
```

```
};

room.requestFrame(requestFramePara, event => console.log(event));
```

### onRecvFrame

**接口描述**

房间帧消息广播回调接口。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| event | MGOBE.types.BroadcastEvent<MGOBE.types.RecvFrameBst> | 回调参数 |

> 说明
> - onRecvFrame 广播表示收到一个帧 frame，frame 的内容由多个 MGOBE.types.FrameItem 组成，即一帧时间内房间里所有玩家向服务器发送帧消息的集合。

**返回值说明**

无

**使用示例**

```
room.onRecvFrame = event => {
console.log("帧广播", event.data.frame);
};
```

### onStartFrameSync

**接口描述**

开始帧同步广播回调接口。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| event | MGOBE.types.BroadcastEvent<MGOBE.types.StartFrameSyncBst> | 回调参数 |

> 说明
> - onStartFrameSync 广播表示房间开始帧同步。收到该广播后将持续收到 onRecvFrame 广播。

**返回值说明**

无

**使用示例**

```
room.onStartFrameSync = event => console.log("开始帧同步");
```

### onStopFrameSync

**接口描述**

停止帧同步广播回调接口。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| event | MGOBE.types.BroadcastEvent<MGOBE.types.StopFrameSyncBst> | 回调参数 |

> 说明
> - onStopFrameSync 广播表示房间停止帧同步。收到该广播后将不再收到 onRecvFrame 广播。

**返回值说明**

无

**使用示例**

```
room.onStopFrameSync = event => console.log("停止帧同步");
```

### onAutoRequestFrameError

**接口描述**

自动补帧失败回调接口。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| event | MGOBE.types.BroadcastEvent<MGOBE.types.ResponseEvent<MGOBE.types.RequestFrameRsp>> | 回调参数 |

> 说明
> - onAutoRequestFrameError 表示自动补帧失败，在初始化 Listener 时开启自动补帧后才能触发。
> - 发生补帧失败后，将不能收到帧广播，您可以使用 retryAutoRequestFrame 方法重试自动补帧。

**返回值说明**

无

**使用示例**

```
room.onAutoRequestFrameError = event => {
console.log("自动补帧失败", event.data.code);
// 重试
room.retryAutoRequestFrame();
};
```

### retryAutoRequestFrame

**接口描述**

重试自动补帧。

**参数描述**

无

> 说明
> - 当收到 onAutoRequestFrameError 回调时，表示自动补帧失败，您可以使用该方法重新触发自动补帧。

**返回值说明**

**无**

**使用示例**

```
room.onAutoRequestFrameError = event => {
console.log("自动补帧失败", event.data.code);
// 重试
room.retryAutoRequestFrame();
};
```

# 消息发送相关接口

最近更新时间：2022-03-29 14:17:23

> **注意：**
> 由于产品逻辑已无法满足游戏行业技术发展，游戏联机对战引擎 MGOBE 将于2022年6月1日下线，请您在2022年5月31日前完成服务迁移。

## sendToClient

**接口描述**

发送消息给房间内玩家。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| sendToClientPara | MGOBE.types.SendToClientPara | 发送消息参数 |
| callback | MGOBE.types.ReqCallback<MGOBE.types.SendToClientRsp> | 响应回调函数 |

> **说明**
> - 调用结果将在 callback 中异步返回。调用成功后所指定接收消息的玩家将收到 onRecvFromClient 广播。
> - 当 recvType 值为1（即 ROOM_ALL ）时，房间内全部玩家将收到消息。
> - 当 recvType 值为2（即 ROOM_OTHERS ）时，房间内除消息发送者外的其他玩家将收到消息。
> - 当 recvType 值为3（即 ROOM_SOME ）时，接收消息玩家由 recvPlayerList 决定。

**返回值说明**

无。

**使用示例**

```
const sendToClientPara = {
recvType: MGOBE.ENUM.RecvType.ROOM_SOME,
recvPlayerList: ["xxxxxxxx1", "xxxxxxxx2"],
msg: "hello",
};

room.sendToClient(sendToClientPara, event => console.log(event));
```

## onRecvFromClient

**接口描述**

收到房间内其他玩家消息广播回调接口。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| event | MGOBE.types.BroadcastEvent<MGOBE.types.RecvFromClientBst> | 回调参数 |

> **说明**
> onRecvFromClient 广播表示收到来自 ID 为 sendPlayerId 的玩家消息。

**返回值说明**

无。

**使用示例**

```
room.onRecvFromClient = event => console.log("新消息", event.data.msg);
```

### sendToGameSvr

**接口描述**

发送消息给实时服务器。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| sendToGameSvrPara | MGOBE.types.SendToGameSvrPara | 发送消息参数 |
| callback | MGOBE.types.ReqCallback<MGOBE.types.SendToGameSvrRsp> | 响应回调函数 |

> 说明
> 该接口只能在玩家进入房间后调用，调用结果将在 callback 中异步返回。

**返回值说明**

无。

**使用示例**

```
const sendToGameServerPara = {
data: {
cmd: 1,
},
};

room.sendToGameSvr(sendToGameServerPara, event => console.log(event));
```

### onRecvFromGameSvr

**接口描述**

收到实时服务器消息广播回调接口。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| event | MGOBE.types.BroadcastEvent<MGOBE.types.RecvFromGameSvrBst> | 回调参数 |

> 说明
> onRecvFromGameSvr 广播表示收到来自实时服务器的消息。

**返回值说明**

无。

**使用示例**

```
room.onRecvFromGameSvr = event => console.log("新自定义服务消息", event);
```

# Group 对象

最近更新时间：2022-03-29 14:17:34

> **注意：**
> 由于产品逻辑已无法满足游戏行业技术发展，游戏联机对战引擎 MGOBE 将于2022年6月1日下线，请您在2022年5月31日前完成服务迁移。

Group 类为 MGOBE 的子属性，用来实现玩家组成队组。

## constructor

**接口描述**

构造器。

**参数描述**

| 参数名 | 类型/值 | 描述 | 可选 |
| --- | --- | --- | --- |
| groupInfo | MGOBE.types.GroupInfo | 队组信息 | 是 |

> **说明**
> - 实例化 Group 对象时可以传入一个 MGOBE.types.GroupInfo 对象 groupInfo，后续接口调用都将基于该 groupInfo，例如修改该队组的属性、接收该队组的广播。
> - 如果不传 groupInfo 参数，您可以通过直接调用 initGroup、createGroup、joinGroup 等方法获取 groupInfo。
> - Group 对象会自动维护内部的 groupInfo 属性保持最新，您可以直接通过访问该属性获得最新的队组信息。

**返回值说明**

无

**使用示例**

```
// 示例1：不传 groupInfo 参数
// 该 Group 实例没有队组信息
const group1 = new MGOBE.Group();

// 示例2：传 groupInfo 参数
// 该 Group 实例代表 ID 为 xxx 的队组，group2.getGroupDetail 将查询 xxx 队组信息
const groupInfo = {
id: "xxx",
// 其他字段
// ...
};
const group2 = new MGOBE.Group(groupInfo);
```

## groupInfo

**对象描述**

队组信息。

**参数描述**

无

> **说明**
> groupInfo 为 Group 实例的属性，类型为 MGOBE.types.GroupInfo，调用 Group 相关的接口会导致该属性发生变化。

**使用示例**

```
// 打印 Group 实例的 groupInfo 属性
// 如需刷新该属性，可以使用 getGroupDetail 方法
console.log(group.groupInfo);
```

### initGroup

**接口描述**

初始化 Group 实例的队组信息，即更新 groupInfo 属性。

**参数描述**

| 参数名 | 类型/值 | 描述 | 可选 |
| --- | --- | --- | --- |
| groupInfo | MGOBE.types.GroupInfo 或 { id: string } | 初始化参数，id表示队组ID | 是 |

> **说明**
> - initGroup 会更新 Group 实例的 groupInfo，接受 MGOBE.types.GroupInfo 或 { id: string; } 类型的参数。
> - 如果不传参数，该方法将清空 Group 实例的 groupInfo 属性。
> - 当玩家要加入指定 ID 队组时，需要使用该接口初始化 Group 实例的 groupInfo 属性，然后才能通过调用 joinGroup 方法，加入该 Group 实例所代表的队组。

**返回值说明**

无

**使用示例**

```
const group = new Group();

// 示例1：不传 groupInfo 参数
// 该 Group 实例队组信息被清除
group.initGroup();

// 示例2：指定队组 ID
// 该 Group 实例代表 ID 为 xxx 的队组，group.getGroupDetail 将查询 xxx 队组信息
const groupInfo = { id: "xxx" };
group.initGroup(groupInfo);
```

### getGroupByGroupId

**接口描述**

通过队组 ID 获取队组信息。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| getGroupByGroupIdPara | MGOBE.types.GetGroupByGroupIdPara | 获取队组参数 |
| callback | MGOBE.types.ReqCallback<MGOBE.types.GetGroupByGroupIdRsp> | 响应回调函数 |

> **说明**
> - 调用结果将在 callback 中异步返回。
> - 该接口为 Group 的静态方法，只能通过 Group.getGroupByGroupId 方式调用，Group 实例无法直接访问该方法。

**返回值说明**

无

**使用示例**

```
// 查询指定队组ID的信息
const getGroupByGroupIdPara = {
groupId: "xxx",
};

MGOBE.Group.getGroupByGroupId(getGroupByGroupIdPara, event => console.log(event));
```

### getMyGroups

**接口描述**

获取当前玩家队组信息。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| callback | MGOBE.types.ReqCallback<MGOBE.types.GetMyGroupsRsp> | 响应回调函数 |

> 说明
> - 调用结果将在 callback 中异步返回。
> - 该接口为 Group 的静态方法，只能通过 Group.getMyGroups 方式调用，Group 实例无法直接访问该方法。

**返回值说明**

无

**使用示例**

```
MGOBE.Group.getMyGroups(event => {
if (event.code === 0 && event.data.groupInfoList.length > 0) {
console.log("玩家在队组数量=", event.data.groupInfoList.length);
// 可以使用队组信息初始化 Group 实例
const group = new MGOBE.Group(event.data.groupInfoList[0]);
}
});
```

### createGroup

**接口描述**

创建队组。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| createGroupPara | MGOBE.types.CreateGroupPara | 创建队组参数 |
| callback | MGOBE.types.ReqCallback<MGOBE.types.CreateGroupRsp> | 响应回调函数 |

> 说明

调用结果将在 callback 中异步返回。操作成功后，groupInfo 属性将更新。

**返回值说明**

无

**使用示例**

```
const playerInfo = {
name: "Tom",
customGroupPlayerStatus: 1,
customGroupPlayerProfile: "https://xxx.com/icon.png",
};

const createGroupPara = {
groupName: "队组名",
groupType: MGOBE.ENUM.GroupType.GROUP_MANY,
maxPlayers: 4,
isForbidJoin: false,
isPersistent: false,
customProperties: "自定义队组属性",
playerInfo,
};

group.createGroup(createGroupPara, event => console.log(event));
```

## getGroupDetail

**接口描述**

获取 Group 实例的队组信息。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| callback | MGOBE.types.ReqCallback<MGOBE.types.GetGroupByGroupIdRsp> | 响应回调函数 |

说明

调用结果将在 callback 中异步返回。操作成功后，groupInfo 属性将更新。

**返回值说明**

无

**使用示例**

```
group.getGroupDetail(event => {
if (event.code === 0) {
console.log("队组名", event.data.groupInfo.name);
}
});
```

## joinGroup

**接口描述**

加入队组。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| joinGroupPara | MGOBE.types.JoinGroupPara | 加入队组参数 |
| callback | MGOBE.types.ReqCallback<MGOBE.types.JoinGroupRsp> | 响应回调函数 |

**说明**

- 调用结果将在 callback 中异步返回。
- 该接口加入的队组是 Group 实例所代表的队组，如果该 Group 实例的 groupInfo 不存在队组ID，则需要使用队组ID通过 initGroup 方法初始化 Group 实例。
- 加入队组成功后，队组内全部成员（不含调用者）都会收到一条玩家加入队组广播 onJoinGroup，groupInfo 属性将更新。
- 玩家可以加入多个 GROUP_MANY 类型队组（type 为 GroupType.GROUP_MANY），同时加入的数量上限为5个。
- 玩家最多只能加入1个 GROUP_LIMITED 类型队组（type 为 GroupType.GROUP_LIMITED）。

**返回值说明**

无

**使用示例**

```
const playerInfo = {
name: "Tom",
customGroupPlayerStatus: 1,
customGroupPlayerProfile: "https://xxx.com/icon.png",
};

const joinGroupPara = {
playerInfo,
};

// 加入指定 ID 的队组
const group = new MGOBE.Group();
group.initGroup({ id: "xxx" });
group.joinGroup(joinGroupPara, event => console.log(event));
```

## leaveGroup

**接口描述**

离开队组。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| para | object | 预留参数，传{}即可 |
| callback | MGOBE.types.ReqCallback<MGOBE.types.LeaveGroupRsp> | 响应回调函数 |

**说明**

- 调用结果将在 callback 中异步返回。退出成功后，队组内剩余成员（不含调用者）都会收到一条玩家退出队组广播 onLeaveGroup，groupInfo 属性将更新，groupInfo.groupPlayerList 中将没有该玩家信息。
- 离开队组后，如果队组内还剩下其他玩家，则该 Group 实例仍然代表退房前的队组，可以继续调用 group.initGroup() 清除队组信息。

**返回值说明**

**无**

**使用示例**

```
group.leaveGroup({}, event => {
if (event.code === 0) {
console.log("退出队组成功", group.groupInfo.id);
// 可以使用 initGroup 清除 groupInfo
group.initGroup();
}
});
```

## dismissGroup

**接口描述**

解散队组。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| para | object | 预留参数，传{}即可 |
| callback | MGOBE.types.ReqCallback<MGOBE.types.DismissGroupRsp> | 响应回调函数 |

> **说明**
> - 调用结果将在 callback 中异步返回。解散成功后，队组内全部成员（不含调用者）都会收到一条解散队组广播 onDismissGroup，groupInfo 属性将更新。
> - 只有队长有权限解散队组。

**返回值说明**

**无**

**使用示例**

```
group.dismissGroup({}, event => {
if (event.code === 0) {
console.log("解散成功");
}
});
```

## changeGroup

**接口描述**

更新队组信息。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| changeGroupPara | MGOBE.types.ChangeGroupPara | 更新队组参数 |
| callback | MGOBE.types.ReqCallback<MGOBE.types.ChangeGroupRsp> | 响应回调函数 |

> **说明**
> - 调用结果将在 callback 中异步返回。调用成功后，队组内全部成员都会收到一条更新队组广播 onChangeGroup，groupInfo 属性将更新。

- 只有队长有权限修改队组。

**返回值说明**

无

**使用示例**

```
const changeGroupPara = {
groupName: "队组名",
owner: "xxxxxx",
isForbidJoin: false,
customProperties: "xxxxxx",
};

group.changeGroup(changeGroupPara, event => {
if (event.code === 0) {
console.log("更新队组成功", event.data.groupInfo);
}
});
```

### removeGroupPlayer

**接口描述**

移除队组内玩家。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| removeGroupPlayerPara | MGOBE.types.RemoveGroupPlayerPara | 移除队组内玩家参数 |
| callback | MGOBE.types.ReqCallback<MGOBE.types.RemoveGroupPlayerRsp> | 响应回调函数 |

说明

- 调用结果将在 callback 中异步返回。调用成功后，队组内全部成员都会收到一条队组内玩家被移除广播 onRemoveGroupPlayer，groupInfo 属性将更新。
- 只有队长有权限移除玩家。

**返回值说明**

无

**使用示例**

```
const removeGroupPlayerPara = {
removePlayerId: "xxxxxx",
};

group.removeGroupPlayer(removeGroupPlayerPara, event => console.log(event));
```

### changeCustomGroupPlayerStatus

**接口描述**

修改队组玩家自定义状态。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| changeCustomGroupPlayerStatusPara | MGOBE.types.ChangeCustomGroupPlayerStatusPara | 修改队组玩家自定义状态参数 |
| callback | MGOBE.types.ReqCallback<MGOBE.types.ChangeCustomGroupPlayerStatusRsp> | 响应回调函数 |

> 说明
> - 修改玩家状态是修改 GroupPlayerInfo 中的 customGroupPlayerStatus 字段，玩家的状态由您自定义。
> - 修改成功后，队组内全部成员都会收到一条修改队组玩家状态广播 onChangeCustomGroupPlayerStatus，groupInfo 属性将更新。
> - 每个玩家只能修改自己的状态，调用结果将在 callback 中异步返回。

**返回值说明**

无

**使用示例**

```
const changeCustomGroupPlayerStatusPara = {
customGroupPlayerStatus: 2,
};

group.changeCustomGroupPlayerStatus(changeCustomGroupPlayerStatusPara, event => console.log(event));
```

### changeGroupPlayerProfile

**接口描述**

修改队组玩家自定义属性。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| changeGroupPlayerProfilePara | MGOBE.types.ChangeGroupPlayerProfilePara | 修改队组玩家自定义属性参数 |
| callback | MGOBE.types.ReqCallback<MGOBE.types.ChangeGroupPlayerProfileRsp> | 响应回调函数 |

> 说明
> - 修改玩家属性是修改 GroupPlayerInfo 中的 customGroupPlayerProfile 字段，玩家的属性由您自定义。
> - 修改成功后，队组内全部成员都会收到一条修改玩家属性广播 onChangeGroupPlayerProfile，groupInfo 属性将更新。
> - 每个玩家只能修改自己的属性，调用结果将在 callback 中异步返回。

**返回值说明**

无

**使用示例**

```
const changeGroupPlayerProfilePara = {
customProfile: "{name: 'name_example'}",
};

group.changeGroupPlayerProfile(changeGroupPlayerProfilePara, event => console.log(event));
```

### sendToGroupClient

**接口描述**

发送队组内消息。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| sendToGroupClientPara | MGOBE.types.SendToGroupClientPara | 发送队组内消息参数 |
| callback | MGOBE.types.ReqCallback<MGOBE.types.SendToGroupClientRsp> | 响应回调函数 |

> 说明
> - 调用结果将在 callback 中异步返回。调用成功后所指定接收消息的玩家将收到 onRecvFromGroupClient 广播。
> - 当 recvType 值为 1（即 GROUP_ALL ）时，队组内全部玩家将收到消息。
> - 当 recvType 值为 2（即 GROUP_OTHERS ）时，队组内除消息发送者外的其他玩家将收到消息。
> - 当 recvType 值为 3（即 GROUP_SOME ）时，接收消息玩家由 recvPlayerList 决定。

**返回值说明**

无

**使用示例**

```
const sendToGroupClientPara = {
recvType: MGOBE.ENUM.GroupRecvType.GROUP_SOME,
recvPlayerList: ["xxxxxxxx1", "xxxxxxxx2"],
msg: "hello",
};

group.sendToGroupClient(sendToGroupClientPara, event => console.log(event));
```

### onUpdate

**接口描述**

队组信息更新回调接口。

**参数描述**

| 参数名 | 类型/值 | 描述 | 可选 |
| --- | --- | --- | --- |
| group | Group | 更新的 Group 实例 | 是 |

> 说明
> onUpdate 表明 Group 实例的 groupInfo 信息发生变化，这种变化原因包括各种队组操作、广播、本地网络状态变化等。

**返回值说明**

无

**使用示例**

```
group.onUpdate = (_) => {
console.log(_ === group); // true, 参数 _ 等于 group
console.log("队组信息更新", group.groupInfo);
};
```

### onRecvFromGroupClient

**接口描述**

收到队组内其他玩家消息广播回调接口。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| event | MGOBE.types.BroadcastEvent<MGOBE.types.RecvFromGroupClientBst> | 回调参数 |

> 说明
> onRecvFromGroupClient 广播表示在队组内收到来自 ID 为 sendPlayerId 的玩家消息。

**返回值说明**

无

**使用示例**

```
group.onRecvFromGroupClient = event => console.log("新消息", event.data.msg);
```

### onChangeCustomGroupPlayerStatus

**接口描述**

玩家自定义状态变化广播回调接口。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| event | MGOBE.types.BroadcastEvent<MGOBE.types.ChangeCustomGroupPlayerStatusBst> | 回调参数 |

> 说明
> onChangeCustomGroupPlayerStatus 广播表示队组内 ID 为 changePlayerId 的玩家状态发生变化。玩家状态由您自定义。

**返回值说明**

无

**使用示例**

```
group.onChangeCustomGroupPlayerStatus = event => {
console.log("玩家状态变化", event.data.changePlayerId);
};
```

### onChangeGroupPlayerProfile

**接口描述**

玩家自定义属性变化广播回调接口。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| event | MGOBE.types.BroadcastEvent<MGOBE.types.ChangeGroupPlayerProfileBst> | 回调参数 |

> 说明
> onChangeGroupPlayerProfile 广播表示队组内 ID 为 changePlayerId 的玩家属性发生变化。玩家属性由您自定义。

**返回值说明**

无

**使用示例**

```
group.onChangeGroupPlayerProfile = event => {
console.log("玩家属性变化", event.data.changePlayerId);
};
```

## onChangeGroupPlayerNetworkState

**接口描述**

队组内玩家网络状态变化广播回调接口。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| event | MGOBE.types.BroadcastEvent<MGOBE.types.ChangeGroupPlayerNetworkStateBst> | 回调参数 |

> 说明
> - onChangeGroupPlayerNetworkState 广播表示 ID 为 changePlayerId 的玩家网络状态发生变化。
> - 玩家在队组中的网络变化都会触发该广播，因此 networkState 将有两种情况，分别表示队组中上线、队组中下线。

**返回值说明**

无

**使用示例**

```
group.onChangeGroupPlayerNetworkState = event => {
if (event.data.networkState === MGOBE.ENUM.NetworkState.COMMON_OFFLINE)
console.log("玩家下线");
};
```

## onRemoveGroupPlayer

**接口描述**

队组内玩家被移除广播回调接口。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| event | MGOBE.types.BroadcastEvent<MGOBE.types.RemoveGroupPlayerBst> | 回调参数 |

> 说明

onRemoveGroupPlayer 广播表示有玩家被队长移除，队组内全部成员都会收到该广播。

**返回值说明**

无

**使用示例**

```
group.onRemoveGroupPlayer = event => console.log("玩家被移除", event.data.removePlayerId);
```

## onChangeGroup

**接口描述**

更新队组广播回调接口。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| event | MGOBE.types.BroadcastEvent<MGOBE.types.ChangeGroupBst> | 回调参数 |

说明

onChangeGroup 广播表示修改了该队组属性，队组内全部成员都会收到该广播。

**返回值说明**

无

**使用示例**

```
group.onChangeGroup = event => console.log("队组属性变更", event.data.groupInfo);
```

## onDismissGroup

**接口描述**

队组被解散广播回调接口。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| event | MGOBE.types.BroadcastEvent<MGOBE.types.DismissGroupBst> | 回调参数 |

说明

onDismissGroup 广播表示队长解散了该队组，队组内全部成员都会收到该广播。

**返回值说明**

无

**使用示例**

```
group.onDismissGroup = event => console.log("队组解散");
```

## onLeaveGroup

**接口描述**

玩家退出队组广播回调接口。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| event | MGOBE.types.BroadcastEvent<MGOBE.types.LeaveGroupBst> | 回调参数 |

> 说明
> onLeaveGroup 广播表示该队组有玩家退出，队组内全部成员都会收到该广播。

**返回值说明**

无

**使用示例**

```
group.onLeaveGroup = event => console.log("玩家退出", event.data.leavePlayerId);
```

### onJoinGroup

**接口描述**

新玩家加入队组广播回调接口。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| event | MGOBE.types.BroadcastEvent<MGOBE.types.JoinGroupBst> | 回调参数 |

> 说明
> onJoinGroup 广播表示该队组有新玩家加入，队组内全部成员都会收到该广播。

**返回值说明**

无

**使用示例**

```
group.onJoinGroup = event => console.log("新玩家加入", event.data.joinPlayerId);
```

# ErrCode 错误码对象

最近更新时间：2022-03-29 14:17:42

> **注意：**
> 由于产品逻辑已无法满足游戏行业技术发展，游戏联机对战引擎 MGOBE 将于2022年6月1日下线，请您在2022年5月31日前完成服务迁移。

该属性是一个 object，记录了 SDK 全部错误码，具体的 key 和 value 与错误码文档相对应。详情请参见 错误码。该对象结构示例如下：

```
ErrCode = {
//返回成功
EC_OK: 0,
//请求包格式错误
EC_REQ_BAD_PKG: 1,
// 其他错误码
// ...
}
```

# ENUM 枚举对象

最近更新时间：2022-03-29 14:17:47

> ⚠ 注意：
> 由于产品逻辑已无法满足游戏行业技术发展，游戏联机对战引擎 MGOBE 将于2022年6月1日下线，请您在2022年5月31日前完成服务迁移。

ENUM 对象为 MGOBE 的子属性，记录了 SDK 常用的几种枚举数据，且均由枚举名称（key）和值（value）组成的 object。ENUM 对象全部属性如下：

## ENUM

**对象描述**

操作类型枚举。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| CreateRoomType | MGOBE.types.CreateRoomType | 创建房间方式 |
| MatchType | MGOBE.types.MatchType | 匹配类型 |
| NetworkState | MGOBE.types.NetworkState | 网络状态 |
| FrameSyncState | MGOBE.types.FrameSyncState | 房间帧同步状态 |
| RecvType | MGOBE.types.RecvType | 房间内消息接收者范围 |
| GroupRecvType | MGOBE.types.GroupRecvType | 队组内消息接收者范围 |
| GroupType | MGOBE.types.GroupType | 队组类型 |

# DebuggerLog 日志打印

最近更新时间：2022-03-29 14:17:52

> **注意：**
> 由于产品逻辑已无法满足游戏行业技术发展，游戏联机对战引擎 MGOBE 将于2022年6月1日下线，请您在2022年5月31日前完成服务迁移。

### 接口描述

SDK 内部日志打印工具，记录接口关键调用步骤。

### 参数说明

| 名称 | 类型 | 描述 |
| --- | --- | --- |
| enable | boolean | 是否开启控制台日志打印 |
| callback | Function | 日志回调函数 |

> **说明**
> 设置回调函数后，每一条日志将作为该函数的参数。

### 使用示例

```
MGOBE.DebuggerLog.enable = true;
MGOBE.DebuggerLog.callback = (log) => console.log(...log); // 默认值;
```

# RandomUtil 随机数工具

最近更新时间：2022-03-29 14:17:57

> **注意：**
> 由于产品逻辑已无法满足游戏行业技术发展，游戏联机对战引擎 MGOBE 将于2022年6月1日下线，请您在2022年5月31日前完成服务迁移。

MGOBE SDK 内部实现了一个基于“线性同余算法”的随机数生成方法。RandomUtil 对象方法如下：

## init

**接口描述**

初始化随机数。

**参数描述**

| 参数名 | 类型/值 | 描述 |
| --- | --- | --- |
| seed | number | 随机数种子 |

> **说明**
> init 方法接受一个 seed 为参数，RandomUtil 在后续生成随机数的过程中将以 seed 为种子。使用相同的 seed 初始化，调用 random 方法生成的随机数序列相同。

**返回值说明**

无

**使用示例**

```
MGOBE.RandomUtil.init(12345678);
```

## random

**接口描述**

生成随机数。

**参数描述**

无

> **说明**
> 如果种子相同、初始化后调用次数相同，生成的随机数将相同。

**返回值说明**

返回值类型为 number，表示随机数，范围为[0,1)。

**使用示例**

```
const num = MGOBE.RandomUtil.random();
```

# 对象类型定义

最近更新时间：2022-03-29 14:18:02

> **注意：**
> 由于产品逻辑已无法满足游戏行业技术发展，游戏联机对战引擎 MGOBE 将于2022年6月1日下线，请您在2022年5月31日前完成服务迁移。

## PlayerInfoPara

**对象描述**

玩家信息参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| name | string | 玩家昵称 |
| customPlayerStatus | number | 自定义玩家状态 |
| customProfile | string | 自定义玩家信息 |

## MatchPlayerInfoPara

**对象描述**

玩家信息参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| name | string | 玩家昵称 |
| customPlayerStatus | number | 自定义玩家状态 |
| customProfile | string | 自定义玩家信息 |
| matchAttributes | MGOBE.types.MatchAttribute[] | 匹配属性 |

## GameInfoPara

**对象描述**

初始化参数：游戏信息。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| gameId | string | 游戏 ID |
| openId | string | 玩家 openId |
| secretKey | string | 游戏密钥 |
| createSignature | MGOBE.types.CreateSignature | 签名函数 |

> **说明**
> - 游戏密钥指控制台上的“游戏 key”。在初始化 SDK 时，secretKey、CreateSignature 两个参数传其中一个即可。如果实现了 CreateSignature 方法，则可忽略 secretKey 参数。
> - CreateSignature 用于计算签名 signature，优点在于避免客户端泄露游戏密钥。

## ConfigPara

**对象描述**

初始化参数：配置参数。

**参数描述**

| 属性名 | 类型/值 | 描述 | 可选 |
| --- | --- | --- | --- |
| reconnectMaxTimes | number | 重连接次数（默认15） | 是 |
| reconnectInterval | number | 重连接时间间隔（毫秒，默认500） | 是 |
| resendInterval | number | 消息重发时间间隔（毫秒，默认1000） | 是 |
| resendTimeout | number | 消息重发超时时间（毫秒，默认20000） | 是 |
| url | string | 服务地址 | - |
| isAutoRequestFrame | boolean | 是否自动补帧（默认 false） | 是 |
| cacertNativeUrl | string | 本地 CA 根证书路径（CocosNative 环境需要该参数） | 是 |

> 说明
> 服务地址为控制台上的“域名”。

## Signature

**对象描述**

初始化签名。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| sign | string | 签名 |
| nonce | number | 随机正整数（uint64 类型） |
| timestamp | number | 时间戳，秒（uint64 类型） |

> 说明
> 可以使用签名的方式初始化 SDK，避免客户端泄露游戏密钥。

## CreateSignature

**对象描述**

签名函数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| callback | (signature: MGOBE.types.Signature) => any | 回调函数，在该函数返回 Signature 对象 |

> 说明
> 您如果使用签名方式初始化 SDK，需要实现该方法，并在 callback 中回调 Signature 对象。

## ChangeCustomPlayerStatusPara

**对象描述**

修改玩家自定义状态参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| customPlayerStatus | number | 自定义玩家状态 |

### ChangeRoomPlayerProfilePara

**对象描述**

修改玩家自定义属性参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| customProfile | string | 玩家自定义属性 |

### CreateRoomPara

**对象描述**

创建房间参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| roomName | string | 房间名称 |
| roomType | string | 房间类型 |
| maxPlayers | number | 房间最大玩家数量 |
| isPrivate | boolean | 是否私有 |
| customProperties | string | 自定义房间属性 |
| playerInfo | MGOBE.types.PlayerInfoPara | 玩家信息 |

### CreateTeamRoomPara

**对象描述**

创建团队房间参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| roomName | string | 房间名称 |
| roomType | string | 房间类型 |
| maxPlayers | number | 房间最大玩家数量 |
| isPrivate | boolean | 是否私有 |
| customProperties | string | 自定义房间属性 |
| playerInfo | MGOBE.types.PlayerInfoPara | 玩家信息 |
| teamNumber | number | 队伍数量 |

### JoinRoomPara

**对象描述**

加入房间参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| playerInfo | MGOBE.types.PlayerInfoPara | 玩家信息 |

### JoinTeamRoomPara

**对象描述**

加入团队房间参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| playerInfo | MGOBE.types.PlayerInfoPara | 玩家信息 |
| teamId | string | 队伍 ID |

### ChangeRoomPara

**对象描述**

房间变更参数。

**参数描述**

| 属性名 | 类型/值 | 描述 | 可选 |
| --- | --- | --- | --- |
| roomName | string | 房间名称 | 是 |
| owner | string | 房主 ID | 是 |
| isPrivate | boolean | 是否私有 | 是 |
| customProperties | string | 自定义房间属性 | 是 |
| isForbidJoin | boolean | 是否禁止加入房间 | 是 |

### RemovePlayerPara

**对象描述**

移除房间内玩家参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| removePlayerId | string | 被移除玩家 ID |

### GetRoomListPara

**对象描述**

获取房间列表参数。

**参数描述**

| 属性名 | 类型/值 | 描述 | 可选 |
| --- | --- | --- | --- |
| pageNo | number | 页号，从1开始 | 否 |
| pageSize | number | 每页数量，最大为10 | 否 |
| roomType | string | 房间类型 | 是 |

| 属性名 | 类型/值 | 描述 | 可选 |
| --- | --- | --- | --- |
| isDesc | boolean | 是否按照房间创建时间倒序 | 是 |

### GetRoomByRoomIdPara

**对象描述**

获取房间参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| roomId | string | 房间 ID |

### MatchPlayersPara

**对象描述**

玩家匹配参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| matchCode | string | 匹配 Code |
| playerInfo | MGOBE.types.MatchPlayerInfoPara | 玩家信息 |

> **说明**
> 匹配 Code 需要在 控制台 创建匹配后获得。

### MatchRoomPara

**对象描述**

房间匹配参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| playerInfo | MGOBE.types.PlayerInfoPara | 玩家信息 |
| maxPlayers | number | 房间最大玩家数量 |
| roomType | string | 房间类型 |

### MatchGroupPlayerInfoPara

**对象描述**

组队匹配玩家信息参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| id | string | 玩家 ID |
| name | string | 玩家昵称 |
| customPlayerStatus | number | 自定义玩家状态 |
| customProfile | string | 自定义玩家信息 |

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| matchAttributes | MGOBE.types.MatchAttribute[] | 匹配属性 |

### MatchGroupPara

**对象描述**

组队匹配参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| matchCode | string | 匹配 Code |
| playerInfoList[] | MGOBE.types.MatchGroupPlayerInfoPara[] | 队员信息 |

> **说明**
> 匹配 code 需要在 控制台 创建匹配后获得。

### MatchBst

**对象描述**

组队匹配结束广播回调参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| roomInfo | MGOBE.types.RoomInfo | 房间信息 |
| errCode | number | 错误码 |

### CancelMatchBst

**对象描述**

组队匹配取消广播回调参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| matchCode | string | 匹配 Code |
| playerId | string | 发起取消匹配的玩家 ID |

### CancelPlayerMatchPara

**对象描述**

取消匹配参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| matchType | MGOBE.types.MatchType | 匹配类型 |

### SendFramePara

**对象描述**

发送帧数据参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| data | object | 帧数据 |

### RequestFramePara

**对象描述**

请求补帧参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| beginFrameId | number | 起始帧号 |
| endFrameId | number | 结束帧号 |

> **说明**
> 补帧范围大于等于 beginFrameId，小于等于 endFrameId。

### RecvType

**对象描述**

房间内消息接收者类型。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| ROOM_ALL | 1 | 全部玩家 |
| ROOM_OTHERS | 2 | 除自己外的其他玩家 |
| ROOM_SOME | 3 | 房间中部分玩家 |

### SendToClientPara

**对象描述**

发送房间内消息参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| recvPlayerList | string[] | 接收消息玩家 ID 列表 |
| msg | string | 消息内容 |
| recvType | MGOBE.types.RecvType | 消息接收者类型 |

### SendToGameSvrPara

**对象描述**

发送实时服务器消息参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| data | object | 消息内容 |

## RecvFromGameSvrBst

**对象描述**

实时服务器消息广播回调参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| roomId | number | 房间 ID |
| recvPlayerIdList | string[] | 接收消息玩家 ID 列表 |
| data | object | 消息内容 |

## FrameItem

**对象描述**

帧内容。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| playerId | string | 玩家 ID |
| data | object | 玩家帧内容 |
| timestamp | number | 时间戳，各玩家本地发送帧的时间 |

## Frame

**对象描述**

帧数据。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| frameId | number | 帧 ID |
| items | MGOBE.types.FrameItem[] | 帧内容 |
| ext | MGOBE.types.FrameExtInfo | 附加信息 |
| roomId | number | 房间 ID |
| time | number | 该帧到达客户端时间 |
| isReplay | boolean | 是否为补帧 |

**说明**

- 附加信息包含一个 number 类型随机种子，您可以使用帧 ID 与随机种子组合成一个值来初始化 RandomUtil 工具。
- time 为 SDK 拟合出来的时间，目的是使每一帧到达客户端的时间尽量均匀分布，并且时间间隔尽量接近帧率的倒数。
- isReplay 表示该帧是否为自动补帧产生的帧，自动补帧需要在初始化 Listener 时设置。
- items 数组表示各个客户端发送的帧消息，按照到达服务器时间先后进行排序（数组中第0个为最先到服务器）。

## RecvFrameBst

**对象描述**

帧广播回调参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| frame | MGOBE.types.Frame | 帧数据 |

### RequestFrameRsp

**对象描述**

请求补帧回调参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| frames | MGOBE.types.Frame[] | 帧数据数组 |

### JoinRoomBst

**对象描述**

玩家加入房间广播回调参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| roomInfo | MGOBE.types.RoomInfo | 房间信息 |
| joinPlayerId | string | 加房玩家 ID |

### LeaveRoomBst

**对象描述**

玩家退出房间广播回调参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| roomInfo | MGOBE.types.RoomInfo | 房间信息 |
| leavePlayerId | string | 退房玩家 ID |

### DismissRoomBst

**对象描述**

房间被解散广播回调参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| roomInfo | MGOBE.types.RoomInfo | 房间解散前的信息 |

### ChangeRoomBst

**对象描述**

房间属性变更广播回调参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| roomInfo | MGOBE.types.RoomInfo | 房间信息 |

### RemovePlayerBst

**对象描述**

房间内玩家被移除广播回调参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| roomInfo | MGOBE.types.RoomInfo | 房间信息 |
| removePlayerId | string | 被移除玩家 ID |

### RecvFromClientBst

**对象描述**

房间消息广播回调参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| roomId | number | 房间 ID |
| sendPlayerId | string | 发送者 ID |
| msg | string | 消息内容 |

### ChangePlayerNetworkStateBst

**对象描述**

房间内玩家网络状态变化广播回调参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| changePlayerId | string | 玩家 ID |
| networkState | MGOBE.types.NetworkState | 网络状态 |
| roomInfo | MGOBE.types.RoomInfo | 房间信息 |

### ChangeCustomPlayerStatusBst

**对象描述**

玩家自定义状态变化广播回调参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| changePlayerId | string | 玩家 ID |
| customPlayerStatus | number | 自定义玩家信息 |
| roomInfo | MGOBE.types.RoomInfo | 房间信息 |

### ChangeRoomPlayerProfileBst

**对象描述**

房间内玩家自定义属性变化广播回调参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| changePlayerId | string | 玩家 ID |
| customProfile | string | 玩家自定义属性 |
| roomInfo | MGOBE.types.RoomInfo | 房间信息 |

### StartFrameSyncBst

**对象描述**

开始帧同步广播回调参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| roomInfo | MGOBE.types.RoomInfo | 房间信息 |

### StopFrameSyncBst

**对象描述**

停止帧同步广播回调参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| roomInfo | MGOBE.types.RoomInfo | 房间信息 |

### RoomInfo

**对象描述**

房间属性。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| id | string | 房间 ID |
| name | string | 房间名称 |
| type | string | 房间类型 |
| createType | MGOBE.types.CreateRoomType | 创建房间方式 |
| maxPlayers | number | 房间最大玩家数量 |
| owner | string | 房主 ID |
| isPrivate | boolean | 是否私有 |
| customProperties | string | 房间自定义属性 |
| playerList | MGOBE.types.PlayerInfo[] | 玩家列表 |
| teamList | MGOBE.types.TeamInfo[] | 团队属性 |
| frameSyncState | MGOBE.types.FrameSyncState | 房间帧同步状态 |
| frameRate | number | 帧率 |

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| routeId | string | 路由 ID |
| createTime | number | 房间创建时的时间戳（单位：秒） |
| startGameTime | number | 开始帧同步时的时间戳（单位：秒） |
| isForbidJoin | boolean | 是否禁止加入房间 |

> 说明
> isPrivate 属性为 true 表示该房间为私有房间，不能被 matchRoom 接口匹配到。

### ChangeCustomPlayerStatusRsp

**对象描述**

修改玩家自定义状态回调参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| roomInfo | MGOBE.types.RoomInfo | 房间信息 |

### ChangeRoomPlayerProfileRsp

**对象描述**

修改玩家自定义属性回调参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| roomInfo | MGOBE.types.RoomInfo | 房间信息 |

### CreateRoomRsp

**对象描述**

创建房间回调参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| roomInfo | MGOBE.types.RoomInfo | 房间信息 |

### JoinRoomRsp

**对象描述**

加入房间回调参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| roomInfo | MGOBE.types.RoomInfo | 房间信息 |

### LeaveRoomRsp

**对象描述**

退出房间回调参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| roomInfo | MGOBE.types.RoomInfo | 房间信息 |

### DismissRoomRsp

**对象描述**

解散房间回调参数。

**参数描述**

无

### ChangeRoomRsp

**对象描述**

修改房间回调参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| roomInfo | MGOBE.types.RoomInfo | 房间信息 |

### RemovePlayerRsp

**对象描述**

移除房间内玩家回调参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| roomInfo | MGOBE.types.RoomInfo | 房间信息 |

### GetRoomByRoomIdRsp

**对象描述**

获取房间信息回调参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| roomInfo | MGOBE.types.RoomInfo | 房间信息 |

### GetRoomListRsp

**对象描述**

获取房间列表回调参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| roomList | MGOBE.types.RoomInfo[] | 房间列表 |
| total | number | 房间总数 |

### MatchPlayersRsp

**对象描述**

玩家匹配回调参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| matchType | MGOBE.types.MatchType | 匹配类型 |
| roomInfo | MGOBE.types.RoomInfo | 房间信息 |

### MatchRoomSimpleRsp

**对象描述**

房间匹配回调参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| roomInfo | MGOBE.types.RoomInfo | 房间信息 |

### MatchGroupRsp

**对象描述**

组队匹配回调参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| matchCode | string | 匹配 Code |

### CancelPlayerMatchRsp

**对象描述**

取消匹配回调参数。

**参数描述**

无

### StartFrameSyncRsp

**对象描述**

开始帧同步回调参数。

**参数描述**

无

### StopFrameSyncRsp

**对象描述**

停止帧同步回调参数。

**参数描述**

无

### SendFrameRsp

**对象描述**

发送帧同步数据回调参数。

**参数描述**

无

### SendToClientRsp

**对象描述**

房间内发送消息回调参数。

**参数描述**

无

### SendToGameSvrRsp

**对象描述**

发送实时服务器消息回调参数。

**参数描述**

无

### MatchAttribute

**对象描述**

匹配属性。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| name | string | 属性名称 |
| value | number | 属性值 |

### MatchType

**对象描述**

匹配类型。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| ROOM_SIMPLE | 1 | 房间匹配 |
| PLAYER_COMPLEX | 2 | 玩家匹配 |

### FrameExtInfo

**对象描述**

帧数据附加信息。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| seed | number | 随机数种子 |

### CreateRoomType

**对象描述**

创建房间方式。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| COMMON_CREATE | 0 | 普通创建 |
| MATCH_CREATE | 1 | 匹配创建 |

### NetworkState

**对象描述**

网络状态。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| COMMON_OFFLINE | 0 | 房间/队组中玩家掉线 |
| COMMON_ONLINE | 1 | 房间/队组中玩家在线 |
| RELAY_OFFLINE | 2 | 帧同步中玩家掉线 |
| RELAY_ONLINE | 3 | 帧同步中玩家在线 |

### FrameSyncState

**对象描述**

房间帧同步状态。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| STOP | 0 | 未开始帧同步 |
| START | 1 | 已开始帧同步 |

### PlayerInfo

**对象描述**

玩家信息。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| id | string | 玩家 ID |
| name | string | 玩家昵称 |
| teamId | string | 队伍 ID |
| customPlayerStatus | number | 自定义玩家状态 |
| customProfile | string | 自定义玩家信息 |
| commonNetworkState | MGOBE.types.NetworkState | 玩家在房间的网络状态 |
| relayNetworkState | MGOBE.types.NetworkState | 玩家在游戏中的网络状态 |
| isRobot | boolean | 玩家是否为机器人 |
| matchAttributes | MGOBE.types.MatchAttribute[] | 玩家匹配属性列表（isRobot 为 true 时生效） |

### TeamInfo

**对象描述**

队伍信息。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| id | string | 队伍 ID |
| name | string | 队伍名称 |
| minPlayers | number | 队伍最小人数 |
| maxPlayers | number | 队伍最大人数 |

## GroupType

**对象描述**

队组类型。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| GROUP_LIMITED | 0 | 玩家只能同时存在于1个该类型队组。该类型的队组人数上限：100。 |
| GROUP_MANY | 1 | 玩家可以同时存在于多个（上限为5）该类型队组。该类型的队组人数上限：300。 |

## GroupInfo

**对象描述**

队组信息。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| id | string | 队组 ID |
| name | string | 队组名称 |
| type | MGOBE.types.GroupType | 队组类型 |
| maxPlayers | number | 队组最大玩家数量 |
| owner | string | 队长 ID |
| customProperties | string | 自定义队组属性 |
| createTime | number | 创建队组时间 |
| isForbidJoin | boolean | 是否禁止加入队组 |
| isPersistent | boolean | 是否持久化 |
| groupPlayerList | MGOBE.types.GroupPlayerInfo[] | 队组内的玩家列表 |

**说明**

- 对于非持久化队组（isPersistent为false），当队组解散，或者队组内没有玩家，或者队组玩家全部掉线超过60分钟，或者队组存在时间超过24小时后，队组会被销毁。
- 对于持久化队组（isPersistent为true），只有当主动解散队组后，队组会被销毁。
- 玩家可以加入多个 GROUP_MANY 类型队组（type 为 GroupType.GROUP_MANY），同时加入的数量上限为5个。
- 玩家最多只能加入1个 GROUP_LIMITED 类型队组（type 为 GroupType.GROUP_LIMITED）。

## GroupPlayerInfo

**对象描述**

队组玩家信息。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| id | string | 玩家 ID |
| name | string | 玩家昵称 |
| customGroupPlayerStatus | number | 自定义玩家状态 |
| customGroupPlayerProfile | string | 自定义玩家信息 |
| commonGroupNetworkState | MGOBE.types.NetworkState | 玩家网络状态 |

### JoinGroupBst

**对象描述**

玩家加入队组广播回调参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| groupInfo | MGOBE.types.GroupInfo | 队组信息 |
| joinPlayerId | string | 加入队组玩家 ID |

### LeaveGroupBst

**对象描述**

玩家退出队组广播回调参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| groupInfo | MGOBE.types.GroupInfo | 队组信息 |
| leavePlayerId | string | 退出队组玩家 ID |

### DismissGroupBst

**对象描述**

队组解散广播回调参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| groupInfo | MGOBE.types.GroupInfo | 队组信息 |

### ChangeGroupBst

**对象描述**

队组属性变更广播回调参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| groupInfo | MGOBE.types.GroupInfo | 队组信息 |

### RemoveGroupPlayerBst

**对象描述**

队组内玩家被移除广播回调参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| groupInfo | MGOBE.types.GroupInfo | 队组信息 |
| removePlayerId | string | 被移除玩家ID |

### ChangeGroupPlayerNetworkStateBst

**对象描述**

队组内玩家网络状态变化广播回调参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| groupInfo | MGOBE.types.GroupInfo | 队组信息 |
| changePlayerId | string | 玩家 ID |
| networkState | MGOBE.types.NetworkState | 玩家网络状态 |

### ChangeCustomGroupPlayerStatusBst

**对象描述**

队组内玩家自定义状态变化广播回调参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| groupInfo | MGOBE.types.GroupInfo | 队组信息 |
| changePlayerId | string | 玩家 ID |
| customGroupPlayerStatus | number | 自定义玩家状态 |

### ChangeGroupPlayerProfileBst

**对象描述**

队组内玩家自定义属性变化广播回调参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| groupInfo | MGOBE.types.GroupInfo | 队组信息 |
| changePlayerId | string | 玩家ID |
| customProfile | number | 自定义玩家属性 |

### RecvFromGroupClientBst

**对象描述**

队组内消息广播回调参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| groupId | string | 队组 ID |
| sendPlayerId | string | 消息发送者 ID |
| msg | string | 消息内容 |

### GetGroupByGroupIdPara

**对象描述**

获取队组信息参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| groupId | string | 队组 ID |

### GetGroupByGroupIdRsp

**对象描述**

获取队组信息回调参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| groupInfo | MGOBE.types.GroupInfo | 队组信息 |

### GetMyGroupsRsp

**对象描述**

获取当前玩家队组信息回调参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| groupInfoList | MGOBE.types.GroupInfo[] | 队组信息列表 |

### GroupPlayerInfoPara

**对象描述**

队组玩家信息参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| name | string | 玩家昵称（字符串长度上限为32） |
| customGroupPlayerStatus | number | 自定义玩家状态（取值范围为[0,99999]） |
| customGroupPlayerProfile | string | 自定义玩家信息（字符串长度上限为1024） |

### CreateGroupPara

**对象描述**

创建队组参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| groupName | string | 队组名称（字符串长度上限为32） |
| groupType | MGOBE.types.GroupType | 队组类型 |
| maxPlayers | number | 队组最大玩家数量 |
| customProperties | string | 自定义队组属性（字符串长度上限为1024） |
| playerInfo | MGOBE.types.GroupPlayerInfoPara | 玩家信息 |
| isForbidJoin | boolean | 是否禁止加入队组 |
| isPersistent | boolean | 是否持久化 |

说明
- 对于持久化队组（isPersistent 为 true），maxPlayers 上限为100。
- 对于非持久化队组（isPersistent 为 false），maxPlayers 上限为300。
- 每个游戏可以创建的持久化队组数量上限为3个。

### CreateGroupRsp

**对象描述**

创建队组回调参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| groupInfo | MGOBE.types.GroupInfo | 队组信息 |

### JoinGroupPara

**对象描述**

加入队组参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| playerInfo | MGOBE.types.GroupPlayerInfoPara | 玩家信息 |

### JoinGroupRsp

**对象描述**

加入队组回调参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| groupInfo | MGOBE.types.GroupInfo | 队组信息 |

### LeaveGroupRsp

**对象描述**

离开队组回调参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| groupInfo | MGOBE.types.GroupInfo | 队组信息 |

### DismissGroupRsp

**对象描述**

解散队组回调参数。

**参数描述**

无

### ChangeGroupPara

**对象描述**

更新队组参数。

**参数描述**

| 属性名 | 类型/值 | 描述 | 可选 |
| --- | --- | --- | --- |
| groupName | string | 队组名称 | 是 |
| owner | string | 队长 ID | 是 |
| customProperties | string | 自定义队组属性 | 是 |
| isForbidJoin | boolean | 是否禁止加入队组 | 是 |

### ChangeGroupRsp

**对象描述**

更新队组回调参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| groupInfo | MGOBE.types.GroupInfo | 队组信息 |

### RemoveGroupPlayerPara

**对象描述**

移除队组玩家参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| removePlayerId | string | 被移除玩家 ID |

### RemoveGroupPlayerRsp

**对象描述**

移除队组玩家回调参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| groupInfo | MGOBE.types.GroupInfo | 队组信息 |

### ChangeCustomGroupPlayerStatusPara

**对象描述**

修改队组玩家自定义状态参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| customGroupPlayerStatus | number | 自定义玩家状态（取值范围为[0,99999]） |

### ChangeCustomGroupPlayerStatusRsp

**对象描述**

修改队组玩家自定义状态回调参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| groupInfo | MGOBE.types.GroupInfo | 队组信息 |

### ChangeGroupPlayerProfilePara

**对象描述**

修改队组玩家属性参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| customProfile | string | 自定义玩家属性 |

### ChangeGroupPlayerProfileRsp

**对象描述**

修改队组玩家属性回调参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| groupInfo | MGOBE.types.GroupInfo | 队组信息 |

### GroupRecvType

**对象描述**

消息接收者类型。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| GROUP_ALL | 1 | 全部玩家 |
| GROUP_OTHERS | 2 | 除自己外的其他玩家 |
| GROUP_SOME | 3 | 队组中部分玩家 |

### SendToGroupClientPara

**对象描述**

发送队组内消息参数。

**参数描述**

| 属性名 | 类型/值 | 描述 |
| --- | --- | --- |
| recvPlayerList | string[] | 接收消息玩家 ID 列表 |
| msg | string | 消息内容 |
| recvType | MGOBE.types.GroupRecvType | 消息接收者类型 |

### SendToGroupClientRsp

**对象描述**

发送队组内消息回调参数。

**参数描述**

无