游戏联机对战引擎

开发指南

产品文档

腾讯云

文档目录

开发指南
加入房间
房间匹配

最近更新时间：2022-03-29 14:11:53

> **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。

本文档用于指导您如何通过房间匹配的方式为玩家加入房间。
房间匹配是指按照传入的参数搜索现存的房间，如果存在，则将玩家加入该房间；如果不存在，则为玩家创建并加入一个新房间。

> **说明**
> 房间匹配不支持自定义匹配规则。请参见 匹配相关接口。

前提条件

- 已在微信/QQ/Unity 或其他平台完成 小游戏项目创建。
- 已 开通 MGOBE 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。SDK 需要对这两个参数进行校验。
- 已 导入 SDK。

操作步骤

开发示例：玩家通过房间匹配的方式加入房间。

1. 游戏配置

```
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,
};
```

2. 定义房间匹配参数

房间匹配参数包含玩家信息、房间最大玩家数、房间类型。

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

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

3. 实例化 Room 对象

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

4. 初始化 Listener，并给房间添加监听

```
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");
}
});
```

5. 调用房间匹配 API

房间匹配 API 请参见 matchRoom。

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

6. Room 接收广播回调

广播回调 API 请参见 onJoinRoom 和 onLeaveRoom。

```
// 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);
```

在线匹配

最近更新时间：2022-03-29 14:11:58

> ⚠ 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。

本文档用于指导您通过发起玩家在线匹配的方式，为玩家创建并加入房间。在线匹配支持自定义匹配规则。请参见 匹配相关接口 和 匹配规则集语法。

前提条件

- 已在微信/QQ/Unity 或其他平台完成 小游戏项目创建。
- 已 开通 MGOBE 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。SDK 需要对这两个参数进行校验。
- 已 导入 SDK。

操作步骤

开发示例：玩家通过在线匹配的方式加入房间。

1. 创建匹配

（1）开发者需要登录 MGOBE 控制台，进入 在线匹配 页面，完成新建匹配。详情请参见 匹配配置。
（2）完成后可在控制台的在线匹配页面，查看对应的**匹配 Code** 。在后续调用客户端 API 时，您需将匹配 Code 添加至玩家在线匹配的参数内。

2. 调用客户端 API

（1）游戏配置

```
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ey”
};
```

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

（2）定义玩家在线匹配参数

玩家在线匹配参数包括玩家信息、匹配编码（matchCode）。

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

const matchPlayersPara = {
playerInfo,
matchCode: "play-xxx",//在MGOBE控制台完成匹配配置后，会生成匹配 code
};
```

（3）实例化 Room 对象

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

（4）初始化 Listener，并给房间添加监听

```
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");
}
});
```

（5）调用玩家在线匹配 API

玩家在线匹配 API 请参见 matchPlayers。

```
room.matchPlayers(matchPlayersPara, event => {
if (event.code === 0) {
console.log("匹配成功", room.roomInfo);
} else {
console.log("匹配失败", event.code);
}
});
```

（6）Room 接收广播回调

广播回调 API 请参见 onJoinRoom 和 onLeaveRoom。

```
// 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);
```

组队匹配

最近更新时间：2022-03-29 14:12:04

> ⚠ 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

本文档用于指导您使用队组（Group）将多个玩家同时发起匹配，用于好友组队匹配的场景。通过将符合要求的玩家组匹配到同一个对局，为玩家组创建并加入房间。组队匹配支持自定义匹配规则。请参见 匹配相关接口 和 匹配规则集语法。

前提条件

- 已在微信/QQ/Unity 或其他平台完成 小游戏项目创建。
- 已 开通 MGOBE 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。SDK 需要对这两个参数进行校验。
- 已 导入 SDK。
- 已通过控制台完成 匹配配置 并获取 匹配 code。

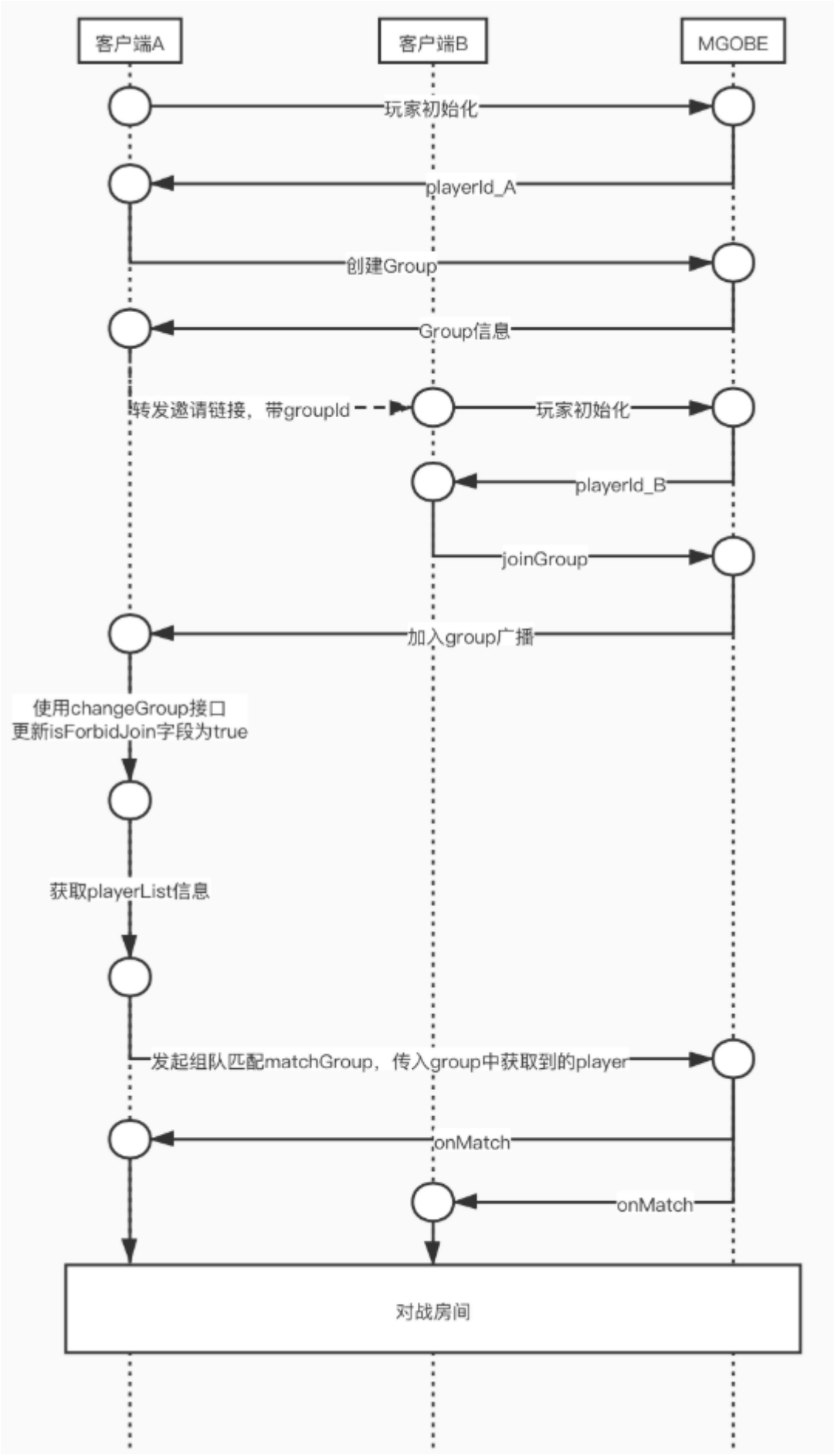
操作步骤

发起多人匹配

使用队组（Group）发起多人匹配的步骤如下：
步骤1：创建 Group。
步骤2：将需要一起匹配的玩家加入同一个队组。
步骤3：Group 的 owner 更新队组的 isForbidJoin 字段为 true（此操作用于防止其他玩家在发起匹配后进入 Group）。
步骤4：Group 的 owner 通过获取该 Group 信息，得到所有 Player 的 ID。
步骤5：Group 的 owner 将得到的 Player ID 作为入参传入 Room 对象的 matchGroup 接口，即可发起组队匹配。

> ⚠ 注意
> - 用于组队匹配场景的 Group，建议将类型定义为 GROUP_LIMITED。1个玩家只能存在于1个类型为 GROUP_LIMITED 的队组；1个玩家可以同时存在于不多于5个类型为 GROUP_MANY 的队组。

- 请使用非持久化队组用于匹配场景。
- 用于组队匹配场景的 Group，建议根据游戏业务逻辑合理销毁，否则可能影响玩家再次匹配。
- MatchGroup（）是 Room 对象的方法。Group 仅作为载体承载多个玩家的结构，与 MatchGroup（）并无耦合。


客户端A
客户端B
MGOBE
玩家初始化
playerId_A
创建Group
Group信息
转发邀请链接，带groupId
玩家初始化
playerId_B
joinGroup
加入group广播
使用changeGroup接口
更新isForbidJoin字段为true
获取playerList信息
发起组队匹配matchGroup，传入group中获取到的player
onMatch
onMatch
对战房间

客户端 API 调用

1. 初始化 Listener

在调用 MGOBE API 前，首先需要初始化 Listener。

```
const gameInfo = {
openId: 'xxxxxx',// 当前玩家的ID
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”
};

Listener.init(gameInfo, config, event => {
if (event.code === 0) {
// 初始化成功
// 继续调用其他 API
// ...
}
});
```

2. 实例化 Room 对象，添加监听

房间相关的接口都位于 Room 对象中。如果用到加房、匹配等接口，需要实例化 Room 对象进行调用。此外，为了能接收广播，需要将其加入到 Listener 中。

```
const room = new Room();
Listener.add(room);
```

3. 实例化 Group 对象，添加监听

队组相关的接口都位于 Group 对象中。如果用到创建队组、加入队组等接口，需要实例化 Group 对象进行调用。此外，为了能接收广播，需要将其加入到 Listener 中。

```
const group = new Group();
Listener.add(group);
```

4. 创建队组，获得队组 ID

使用 group 实例可以创建一个队组：

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

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

group.createGroup(createGroupPara, event => {
if (event.code === 0) {
// 创建队组成功
// 可以分享队组ID
const groupId = group.groupInfo.id;
}
});
```

5. 邀请好友，将好友加入队组

创建队组成功后，可以将队组 ID（groupId）分享给好友，好友可以调用 joinGroup 加入队组：

```
// 使用 groupId 初始化 group 实例
const groupInfo = { id: groupId };
group.initGroup(groupInfo);

// 加入队组
const playerInfo = {
name: "Tom",
```

```
customGroupPlayerStatus: 1,
customGroupPlayerProfile: "https://xxx.com/icon.png",
};

const joinGroupPara = {
playerInfo,
};

group.joinGroup(joinGroupPara, event => {
if (event.code === 0) {
// 加入成功
}
});
```

每个客户端的 group 实例，可以监听玩家加入队组的广播：

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

6. 获得队组内的玩家列表

队组内的玩家列表 groupPlayerList 为队组信息的属性：

```
// 玩家列表
const groupPlayerList = group.groupInfo.groupPlayerList;
// 如果需要刷新当前队组信息，可以使用 getGroupDetail 接口：
group.getGroupDetail(event => {
if (event.code === 0) {
// 调用成功
// 玩家列表，或者使用 event.groupInfo.groupPlayerList
const groupPlayerList = group.groupInfo.groupPlayerList;
}
});
```

7. 修改队组信息

发起匹配前，可以修改队组信息，禁止其他玩家加入队组：

```
const changeGroupPara = {
isForbidJoin: true,
};

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

// 监听广播
group.onChangeGroup = event => {
console.log("队组属性变更", event.data.groupInfo);
};
```

8. 调用队组匹配 API

队组匹配为 room 实例的方法，需要指定玩家列表 playerInfoList。调用成功后可以通过 Room.onMatch、Room.onCancelMatch 监听匹配结果（注意 onMatch、onCancelMatch 为 Room 类的静态方法）。

```
// 队组玩家列表
const groupPlayerList = group.groupInfo.groupPlayerList;

// 匹配玩家信息列表
const playerInfoList = groupPlayerList.map(player => {
return {
id: player.id,
name: player.name,
customPlayerStatus: player.customGroupPlayerStatus,
customProfile: player.customGroupPlayerProfile,
// 可以指定每个玩家的 matchAttributes
matchAttributes: [],
};
});

// 匹配参数
const matchGroupPara = {
```

```
playerInfoList,
// 从控制台获取 matchCode
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（队组内每个客户端都能监听）
Room.onMatch = (event) => {

if (event.data.errCode === 0) {
console.log("组队匹配成功");
// 使用 roomInfo 初始化 room
room.initRoom(event.data.roomInfo);
return;
}

// 匹配失败
// ...
};

// 监听匹配取消（队组内每个客户端都能监听）
Room.onCancelMatch = (event) => {
console.log("组队匹配已取消");
};
```

邀请加入房间

最近更新时间：2022-03-29 14:12:09

> **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。

本文档用于指导您调用游戏联机对战引擎 MGOBE 客户端 API，实现通过邀请的方式将玩家加入指定 ID 房间的场景。

前提条件

- 已在微信/QQ/Unity 或其他平台完成 小游戏项目创建。
- 已 开通 MGOBE 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。SDK 需要对这两个参数进行校验。
- 已 导入 SDK。

操作步骤

开发示例：玩家通过邀请方式加入指定 ID 的房间。

1. 游戏配置

```
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,
};
```

2. 实例化和初始化

```
//定义创建房间参数
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,
};

//实例化Room对象
const room = new Room();

//初始化Listener，并给房间添加监听
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");
}
});
```

3. 创建房间

创建房间 API 请参见 createRoom。

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

4. 通过 room 实例处理广播回调

广播回调 API 请参见 onJoinRoom 和 onLeaveRoom。

```
// 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);
```

5. 调用获取房间信息接口，获取当前玩家房间 ID

```
//2 ）调用获取房间信息接口，获取当前玩家房间ID
const getRoomByRoomIdPara2 = {
roomId: "",
};
MGOBE.Room.getRoomByRoomId(getRoomByRoomIdPara2, event => console.log(event));
```

6. 邀请

调用微信/QQ 小游戏原生接口实现“生成邀请链接”和“分享”。

7. 玩家加入指定 ID 的房间

其他玩家通过邀请链接获得房间 ID 后，通过下面的步骤加入房间：

（1）初始化 SDK

如果玩家进入房间时，尚未初始化 SDK，则需要按上述指引进行实例化和初始化。

（2）加入指定 ID 的房间

加入房间 API 请参见 joinRoom。

```
room.initRoom({ id: "xxx" });
room.joinRoom(joinRoomPara, event => console.log(event));
```

消息通信
发送客户端消息

最近更新时间：2022-03-29 14:12:16

> **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

本文档用于指导您调用游戏联机对战引擎 MGOBE 客户端 API，实现直接发送消息到客户端的通信方式。

前提条件

已通过任意方式 创建并加入房间。

操作步骤

开发示例：发送消息到客户端。

1. 定义参数

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

2. 调用发送到客户端接口

发送到客户端 API 请参见 sendToClient。

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

3. 客户端广播回调

广播回调 API 请参见 onRecvFromClient。

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

帧同步

最近更新时间：2022-03-29 14:12:23

> ⚠ 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。

操作场景

本文档用于指导您调用游戏联机对战引擎 MGOBE 客户端 API，实现帧同步。

前提条件

已通过任意方式 创建并加入房间。

操作步骤

开发示例：帧同步。

1. 开始帧同步

开始帧同步 API 请参见 startFrameSync。

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

2. 开始帧同步广播回调

开始帧同步广播回调 API 请参见 onStartFrameSync。

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

3. 发送帧消息

发送帧消息 API 请参见 sendFrame。

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

4. 房间帧消息广播回调

房间帧消息广播回调 API 请参见 onRecvFrame。

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

5. 停止帧同步

停止帧同步 API 请参见 stopFrameSync。

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

6. 停止帧同步广播回调

停止帧同步广播回调 API 请参见 onStopFrameSync。

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

状态同步

最近更新时间：2022-03-29 14:12:29

> ⚠ 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

本文档用于指导您使用游戏联机对战引擎 MGOBE 实时服务器和客户端 API，实现状态同步。

前提条件

已通过任意方式 创建并加入房间。

操作步骤

开发示例：状态同步。

步骤1：上传服务端逻辑

您需要提供一个 index.js 脚本，在 MGOBE 控制台上传服务端逻辑。上传操作请参见 实时服务器。

index.js 中需要导出一个 mgobexsCode 对象，该对象拥有一个 gameServer 属性。示例代码如下：

```
exports.mgobexsCode = {
gameServer
};

const gameServer = {
// 消息模式
mode: 'sync',
// 初始化游戏数据
onInitGameData: function () {
return {};
},
// 监听客户端数据
onRecvFromClient: function onClientData(args) {
```

```
args.SDK.logger.debug("onRecvFromClient");
args.SDK.exitAction();
}
};
```

步骤2：调用客户端 API

1. 定义实时服务器参数

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

2. 发送到实时服务器

发送到实时服务器 API 请参见 sendToGameSvr。

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

3. 自定义服务消息广播回调

自定义服务消息广播回调 API 请参见 onRecvFromGameSvr。

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

匹配使用说明
规则脚本设计

最近更新时间：2022-03-29 14:12:38

> **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。

您可通过设置匹配规则集，从而实现 1V1、多V多游戏对战。

游戏联机对战引擎控制台 默认提供以下规则集内容：1V1 、2V2、3V3、5V5，添加分段规则，添加误差规则。

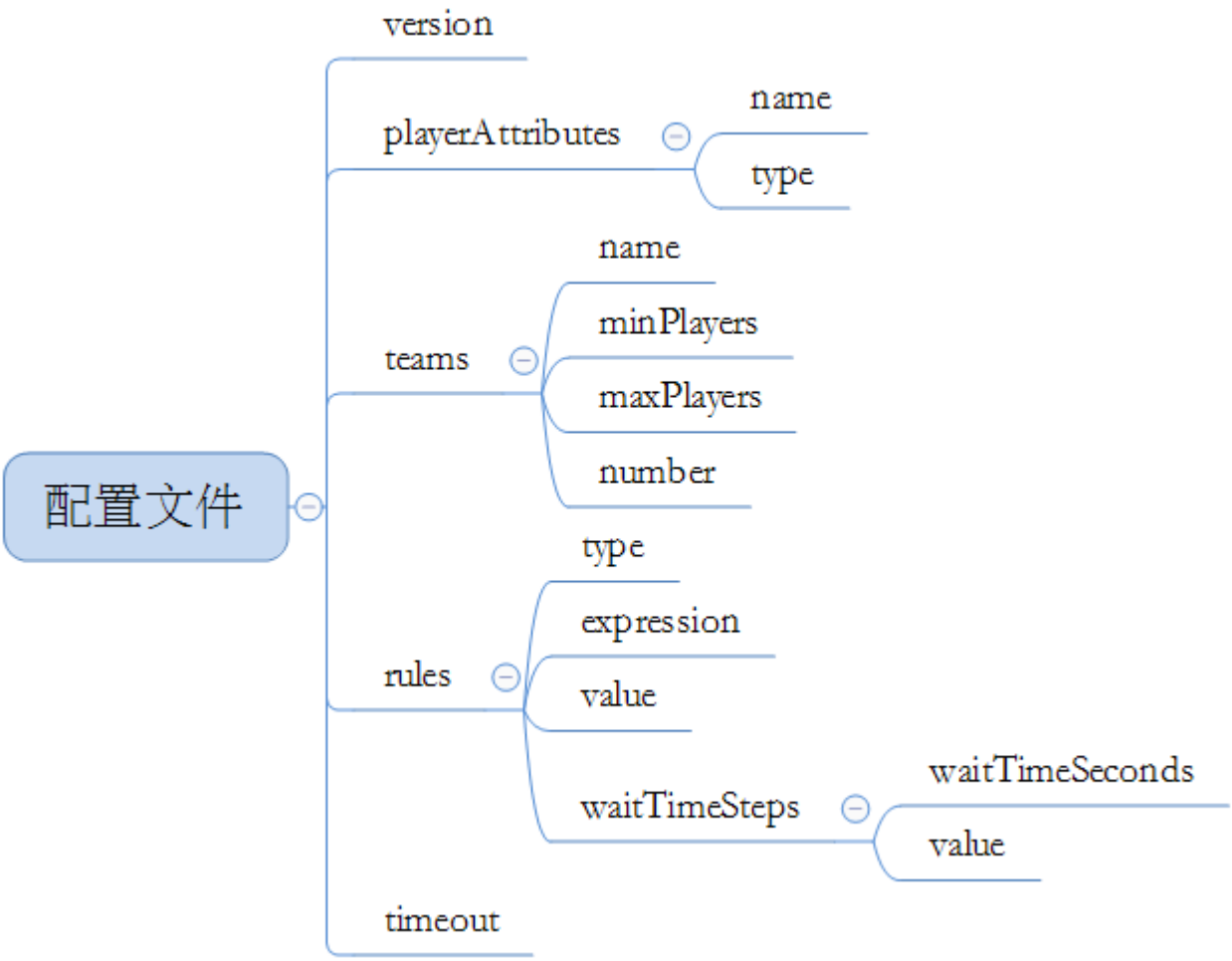
适用场景

- 参与匹配的房间人数不超过100人。
- 用于配置匹配规则的玩家属性为数值型。
- 房间内有1支或多支队伍，支持对称匹配和非对称匹配。

> **说明**
> - 对称匹配：房间中每支队伍人数一致。
> - 非对称匹配：房间中每支队伍人数不一致。

JSON 结构

配置 JSON 结构如下图所示：



示例介绍

规则说明

1. 3V3 比赛。
2. 组队条件：年龄在同一个阶段的人组成一队；1 – 12岁一组，13 – 18岁一组，19 – 30岁一组，31 – 100岁一组。
3. 队与队的匹配条件：每个团队平均技能相差在2以内的匹配在一起。

```
{
"version": "V1.0",
"teams": [{
"name": "3v3", //3v3的比赛
"maxPlayers": 3, //每支队伍最大人数3
"minPlayers": 3, //每支队伍最小人数3
"number": 2 //一共两个队
}],
```

```
"playerAttributes": [{
"name": "age", //年龄，匹配时传入的参数
"type": "number"
},
{
"name": "skill", //技能，匹配时传入的参数
"type": "number"
}
],
"rules": [{
"type": "segment",
"expression": "teams[i].players.age", //根据年龄进行组队
"value": [
[1, 12],
[13, 18],
[19, 30],
[31, 100]
],
"waitTimeSteps": [{
"waitTimeSeconds": 10, //等待10s若匹配不上，放宽规则
"value": [
[1, 30],
[31, 100]
]
}]
}, {
"type": "deviation",
"expression": "avg(teams[*].players.skill)", //每个团队的平均技能
"value": 2, //技能相差不大于2
"waitTimeSteps": [{
"waitTimeSeconds": 10, //等待10s若匹配不上，放宽规则
"value": 5
}]
}],
"timeout": 40
}
```

配置详解

version

版本说明，初始版本只能填写"V1.0"。

playerAttributes

属性数组，包含0 – 5组属性值，可不填属性，属性值可不被用于匹配规则。

参数	类型	意义
name	string	属性字段，接口调用时需要传入的参数。数组内 name 不能重复，只能由英文、数字、下划线组成，且最大长度不超过20个字符
type	string	属性类型，暂时只支持 number 属性，可不填，默认 number

示例代码

```
{
"playerAttributes": [
{
"name": "deviationAttri1",//该参数代表的值，一般由数字度量的属性，如score、skill等
"type": "number"
},
{
"name": "segmentAttri1",//该参数代表的值，一般由数字度量的属性，如score、skill等
"type": "number"
}
]
}
```

teams

创建队伍，一个房间内的玩家最大数量不超过100。

参数	类型	意义
name	string	队伍类型名称， 只能由英文、数字、下划线组成，且最大长度不超过20个字符；i、* 不可作为 team name 使用
maxPlayers	number	队伍最大人数，队伍个数 x 队伍最大人数 <= 100 ，即 maxPlayers x number <= 100

参数	类型	意义
minPlayers	number	队伍最小人数
number	number	该类型队伍个数，队伍个数 x 队伍最大人数 <= 100 ，即 maxPlayers x number <= 100

说明

- 当定义的 team 数组长度 =1 时，number 取值 >= 1。
- 当定义的 team 数组长度 >1 时，number 取值须 = 1。
- 当定义的 team 数组长度 >1 时，team name 不可重复。
- 当前仅支持 minPlayers = maxPlayers。

单一队伍类型示例代码

当规则集脚本中只定义了一种 Team 时，属于单一队伍类型匹配。常见的游戏玩法包括：对称的1v1、3v3、5v5 MOBA 游戏。单一队伍类型匹配，队伍 number 取值范围1-100。规则配置如下：

```
{
"version": "V1.0",
"teams": [{
"name": "5v5",
"maxPlayers": 5, //每个队伍的最大人数为5
"minPlayers": 5, //每个队伍的最小人数为5
"number": 2 //队伍个数为2
}]
}
```

多样队伍类型示例代码

当规则集脚本中定义了多于一种 Team 时，属于多样队伍类型匹配。常见的游戏玩法包括：狼人杀等队伍人数和角色不对称的游戏玩法类型。多样队伍类型匹配，每种队伍 number 取值为1。规则配置如下：

```
{
"version": "V1.0",
"teams":
[{
"name": "killer", // 杀手队
```

```
"maxPlayers": 1, // 每支杀手队伍最大人数1
"minPlayers": 1, // 每支杀手队伍最小人数1
"number": 1 // 每局1个杀手队
},
{
"name": "survivor", // 逃生者队
"maxPlayers": 4, // 每支逃生者队伍最大人数4
"minPlayers": 4, // 每支逃生者队伍最小人数4
"number": 1 // 每局1个逃生者队
}]
}
```

rules

设置匹配规则，总共最多支持5组规则，也可不填 rules。匹配规则分为误差规则、分段规则。

> 说明
> 组队**误差规则**最多支持一组，**分段规则**最多支持5组。

参数说明

参数	类型	意义
type	string	规则类型， "deviation" 或 "segment"
expression	string	玩家某个属性的表达式，表达式可转化成数组
value	number	当 type 为 deviation 误差规则时，该值表示最大误差值； 当 type 为 segment 分段规则时，该值表示分段数组
waitTimeSteps	数组	等待时间超时后重新指定 value 值，仅一组，非必填

规则配置语法

- 用 team[name] 代指具体队伍、用 team[i] 代指每一支队伍，可用于组队。即，将玩家匹配至同一个队伍。
- 用 team[*] 代指队伍与队伍之间，可用于队伍匹配。即，将队伍匹配至同一个房间。

type 规则类型说明

type 包含误差规则 deviation 和分段规则 segment。

- 误差规则 deviation
  误差规则表示一个数组里最大值和最小值相差不超过某个值。误差规则 deviation 的有效 expression 是一个可以转化为数组的字符串，value 是一个值。例如，一人1个队伍，3个人的房间，要求所有玩家技能相差不超过2。例如：
  玩家1技能：1
  玩家2技能：3
  玩家3技能：5
  玩家4技能：2
  玩家1和玩家2的技能相差为2，可以匹配上，玩家3和玩家1技能相差为4，所以匹配不上，玩家4与玩家1和玩家2技能相差为1，可以匹配上。

- 误差规则限制
  - team 内匹配支持不多于1条 value 取值非0的误差规则。
  - team 间匹配支持不多于1条 value 取值非0的误差规则。

  示例代码：

```
{
"version": "V1.0",
"teams": [{
"name": "battle",
"maxPlayers": 3,
"minPlayers": 3,
"number": 2
}],
"playerAttributes": [{
"name": "skill",
"type": "number"
}],
"rules": [{
"type": "deviation", //误差规则
"expression": "teams[battle].players.skill",
"value": 2
}]
}
```

- 分段规则 segment
  表示将一个段的匹配在一起，分段规则 segment 的有效 expression 是一个可以转化为数组的字符串，

value 是一个二维数组。如技能1 – 3的匹配在一起，4 – 6的匹配在一起，7 – 10的匹配在一起。

示例代码：

```
{
"version": "V1.0",
"teams": [{
"name": "da_guai",
"maxPlayers": 3,
"minPlayers": 3,
"number": 1
}],
"playerAttributes": [{
"name": "skill",
"type": "number"
}],
"rules": [{
"type": "segment", //分段规则
"expression": "teams[i].players.skill", //表示每个队伍，这里只有一个队伍
"value": [
[1, 3],
[4, 6],
[7, 10]
] //表示技能1-3的会匹配在一起，技能4-6的匹配在一起，技能7-10的匹配在一起。
}]
}
```

expression 表达式说明

目前仅支持以下表达式，如有更多需求，请 联系我们。

表达式	类型	意义	适用规则
teams[name].players.attribute1	List<Number>	房间里某个指定团队所有成员的属性1	分段规则 segment，误差规则 deviation

表达式	类型	意义	适用规则
teams[i].players.attribute1	List<Number>	房间里任意一个团队所有成员的属性1，如果没有团队概念（1个 team），即房间所有成员的属性1	分段规则 segment，误差规则 deviation
teams[*].players.attribute1	List< List<Number> >	房间里每个团队的属性1，这是一个二维数组	–
avg(teams[*].players.attribute1)	List<number>	房间内每个团队的属性1平均值，通过 avg，二维数组变成一维数组，多个团队时，通常用于团队与团队的匹配条件	分段规则 segment，误差规则 deviation

先通过技能进行分组，技能1 – 3的组队，4 – 6的组队，7 – 10的组队，再通过每个队伍的平均技能，相差为2以内的队伍匹配在一个房间。

示例代码：

```
{
"version": "V1.0",
"teams": [{
"name": "da_guai",
"maxPlayers": 3,
"minPlayers": 3,
"number": 2
}],
"playerAttributes": [{
"name": "skill",
"type": "number"
}],
"rules": [{
"type": "segment", //误差规则，也可用分段规则
"expression": "teams[i].players.skill",//表示组队条件：teams[i]表示队伍i的每个成员，是一个数组。
"value": [
[1, 3],
[4, 6],
[7, 10]
```

```
] //表示技能1-3的会组队，技能4-6的会组队，技能7-10的会组队
}, {
"type": "deviation", //分段规则，也可使用误差规则
"expression": "avg(teams[*].players.skill)", //表示队和队的匹配条件：表示每个队伍之间的平均技能
"value": 2 //技能误差为2
}]
}
```

waitTimeSteps 超时设置

等待时间超时后重新指定 value 值，仅一组，非必填。 这里的超时设置，不能大于 timeout 的时间。

参数	类型	意义
waitTimeSeconds	number	设置等待匹配超时时间，规则集内全部 waitTimeSeconds 必须相同，取值范围：[2, 300]
value	number	与 rules.value 规则一致。当 rules.type 为 deviation 误差规则时，该值表示最大误差值； 当 rules.type 为 segment 分段规则时，该值表示分段数组

示例代码：

```
{
"version": "V1.0",
"teams": [{
"name": "5v5",
"maxPlayers": 5,
"minPlayers": 5,
"number": 2
}],
"playerAttributes": [{
"name": "skill",
"type": "number"
}, {
"name": "score",
"type": "number"
}],
"rules": [{
"type": "deviation", //误差规则
```

```
"expression": "teams[i].players.skill",
"value": 2,
"waitTimeSteps": [{
"waitTimeSeconds": 10,
"value": 5 //误差规则，则waitTimeSteps设定的value值为数字
}]
},
{
"type": "segment", //分段规则
"expression": "teams[i].players.score",
"value": [
[1, 3],
[4, 6],
[7, 10]
],
"waitTimeSteps": [{
"waitTimeSeconds": 10,
"value": [
[0, 5],
[6, 10]
] //分段规则，则waitTimeSteps设定的value范围比前面的value范围更宽松；
}]
}
]
}
```

timeout

参数说明

参数	类型	意义
timeout	number	全局匹配超时，取值范围：[2, 300]s，必须大于 waitTimeSeconds，默认是40s

示例代码

房间内每个团队 skill 数值平均值相差不超过3的匹配在一起。如果超过1分钟未匹配上，将 skill 平均值相差调整到5，整个匹配的超时时间是2分钟。

```
{
"version": "V1.0",
"teams": [{
"name": "5v5",
"maxPlayers": 5,
"minPlayers": 5,
"number": 2
}],
"playerAttributes": [{
"name": "skill",
"type": "number"
}],
"rules": [{
"type": "deviation",
"expression": "avg(teams[*].players.skill)",
"value": 3,
"waitTimeSteps": [{
"waitTimeSeconds": 60,
"value": 5
}]
}],
"timeout": 120//匹配超时60s后，skill平均值相差为5，整个匹配超时时间是120S。
}
```

相关文档

- 规则集配置
- 匹配配置
- 规则脚本示例
- 匹配机制说明

规则脚本示例

最近更新时间：2022-03-29 14:12:46

> **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。

本文档主要介绍5种游戏场景的规则集脚本示例代码。

斗地主 1V1V1

- 匹配条件：按玩家所拥有的积分值区间进行组队，在同一个积分区间内的玩家可被匹配在同一个斗地主房间。
- 积分区间分别为：0 – 100积分、101 – 1000积分、1001 – 5000积分、5001 – 10000积分。

示例代码如下：

```
{
"version": "V1.0",
"teams": [{
"name": "1v1v1",
"maxPlayers": 3,
"minPlayers": 3,
"number": 1
}],
"playerAttributes": [{
"name": "score",
"type": "number"
}],
"rules": [{
"type": "segment",
"expression": "teams[i].players.score",
"value": [
[0, 100],
[101, 1000],
[1001, 5000],
[5001, 10000]
```

```
]
}],
"timeout": 20
}
```

3V3 比赛

- 组队条件：年龄在同一个阶段的玩家组成一队；1 – 12岁一组，13 – 18岁一组，19 – 30岁一组，31 – 100岁一组。
- 队与队的匹配条件：每个团队平均技能相差在2以内的匹配在一个房间。

示例代码如下：

```
{
"version": "V1.0",
"teams": [{
"name": "3v3", //3v3的比赛
"maxPlayers": 3, //每支队伍最大人数3
"minPlayers": 3, //每支队伍最小人数3
"number": 2 //一共两个队
}],
"playerAttributes": [{
"name": "age", //年龄，匹配时传入的参数
"type": "number"
},
{
"name": "skill", //技能，匹配时传入的参数
"type": "number"
}
],
"rules": [{
"type": "segment",
"expression": "teams[i].players.age", //根据年龄进行组队
"value": [
[1, 12],
[13, 18],
[19, 30],
```

```
[31, 100]
],
"waitTimeSteps": [{
"waitTimeSeconds": 10, //等待10s若匹配不上，放宽规则
"value": [
[1, 30],
[31, 100]
]
}]
}, {
"type": "deviation",
"expression": "avg(teams[*].players.skill)", //每个团队的平均技能
"value": 2, //技能相差不大于2
"waitTimeSteps": [{
"waitTimeSeconds": 10, //等待10s若匹配不上，放宽规则
"value": 5
}]
}],
"timeout": 40
}
```

实时对战 5V5

队伍之间的平均分相差3以内，如果超过40秒未匹配上，则将会把平均分相差调整到10以内，整个匹配的超时时间是2分钟。

示例代码如下：

```
{
"version": "V1.0",
"teams": [{
"name": "5v5",
"maxPlayers": 5,
"minPlayers": 5,
"number": 2
}],
"playerAttributes": [{
"name": "score",
```

```
"type": "number"
}],
"rules": [{
"type": "deviation",
"expression": "avg(teams[*].players.score)",
"value": 3,
"waitTimeSteps": [{
"waitTimeSeconds": 40,
"value": 10
}]
}],
"timeout": 120
}
```

答题游戏 3V3

按地域和技能进行匹配，相同地域的玩家被分到同一个房间；同一个房间内两队的技能平均值相差不超过3。

示例代码如下：

```
{
"version": "V1.0",
"teams": [{
"name": "3v3",
"maxPlayers": 3,
"minPlayers": 3,
"number": 2
}],
"playerAttributes": [{
"name": "area",
"type": "number"
},
{
"name": "skill",
"type": "number"
}
],
"rules": [{
```

```
"type": "segment",
"expression": "teams[i].players.area", //相同的地域进行组队
"value": [
[1, 1],
[2, 2],
[3, 3],
[4, 4]
]
}, {
"type": "segment",
"expression": "avg(teams[*].players.area)", //相同的地域进行匹配
"value": [
[1, 1],
[2, 2],
[3, 3],
[4, 4]
]
}, {
"type": "deviation",
"expression": "avg(teams[*].players.skill)", //两队的技能平均值相差不超过3
"value": 3
}],
"timeout": 120
}
```

非对称匹配 1V5

房间中有1个国王，5个士兵；在同一个队的士兵，战斗力（power）在同一个区间内；国王队和士兵队的技能（skill）差值为2。

示例代码如下：

```
{
"version": "V1.0",
"teams": [
{
"name": "king",
"maxPlayers": 1,
```

```
"minPlayers": 1,
"number": 1
},
{
"name": "soldiers",
"maxPlayers": 5,
"minPlayers": 5,
"number": 1
}
],
"playerAttributes": [
{
"name": "wantToBeKing",
"type": "number"
},
{
"name": "power",
"type": "number"
},
{
"name": "skill",
"type": "number"
}
],
"rules": [
{
"type": "segment",
"expression": "teams[king].players.wantToBeKing",
"value": [
[1,1]
]
},
{
"type": "segment",
"expression": "teams[soldiers].players.wantToBeKing",
"value": [
[0, 0]
]
```

```
},
{
"type": "segment",
"expression": "teams[soldiers].players.power",
"value": [
[1,3],
[4,6],
[7,10]
]
},
{
"type": "deviation",
"expression": "avg(teams[*].players.skill)",
"value": 2
}
]
}
```

相关文档

- 规则集配置
- 匹配配置
- 规则脚本设计
- 匹配机制说明

匹配机制说明

最近更新时间：2022-03-29 14:12:51

> **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。

本文档主要介绍 MGOBE 的匹配实现机制，帮助您理解匹配完成结果和匹配超时的现象。

概念解释

玩家组

发起同一个匹配请求中传入的玩家称为一个玩家组。例如通过 MatchPlayers 接口传入的单个玩家，或通过 MatchGroup 接口传入的单个或多个玩家。

队伍

一个队伍表示游戏对局中的一个对战方。队伍在匹配规则集中进行定义。

房间

用于承载一个游戏对局的对象。MGOBE 匹配会自动为成功匹配的玩家创建并将其加入房间。

机器人填充

已开启机器人开关的匹配，玩家匹配超时将被按要求填充机器人，并将玩家加入房间。

搜索合适的玩家组成游戏对局

匹配过程

发起匹配的玩家组，会进入同一个匹配池。匹配池中的玩家组将经历**两个阶段**：

- 等待被匹配成队伍。
- 等待被匹配进房间。

MGOBE 会根据您配置的规则，在现有的匹配池中尽可能为玩家搜索符合条件的匹配对象组成队伍，再将符合条件的队伍组成房间，搜索过程中的两个阶段对您是不可见。由于不断有玩家进入匹配池，也不断有玩家超时，匹配池中的玩家组和队伍都是动态的。所以为了匹配效率，匹配结果并不总是全局最优解。

匹配超时

玩家匹配超时可能发生在上述匹配过程中的任意一个阶段。

为匹配填充机器人

机器人开关

在 游戏联机对战引擎控制台 的**在线匹配**中，可以选择是否开启机器人开关。详见 匹配配置 。

- 关闭机器人开关时，MGOBE 不对超时玩家填充机器人。匹配超时后，玩家无法进入房间开始对战。
- 开启机器人开关，匹配超时的玩家将被独立创建房间，并在空余席位填充机器人；或与其他符合匹配规则的超时玩家一起匹配进入同一房间，并在空余席位填充机器人。

机器人逻辑实现

MGOBE 只会在匹配中指定机器人的席位和属性，并不关注机器人的逻辑实现。您需要在实时服务器或客户端自行实现机器人行为逻辑。机器人的属性与同队伍的真人玩家属性平均值相等；当机器人所在队伍没有真人玩家时，机器人的属性与同房间的真人玩家属性平均值相等。

> ⚠ 注意
> MGOBE 不能保证在所有规则情况下，机器人的属性都符合其所在队伍的属性要求。

超时填充机器人的机制

超时填充机器人的机制可以在 游戏联机对战引擎控制台 的**在线匹配**中进行配置，目前支持两种机器人超时填充机制：

- **优先机器人填充（默认）**
  - 将任意阶段匹配超时的玩家组独立加入房间，在房间内的其余空缺席位填充机器人。如5v5的游戏中，单人发起匹配的玩家超时后，将同队伍的其余4个空缺、不同队伍的5个空缺席位都填充机器人。
  - 这种机制一般用于不希望对战双方的真人玩家数量不对等的玩法，避免单人与组队开黑的多人对战。

- **优先玩家填充**
  - 将有超时玩家组的队伍、超时的玩家组按规则进行匹配并加入房间，在房间内其余空缺席位填充机器人。

> ⚠ 注意
> MGOBE 不会为超时的玩家组匹配未超时的玩家组。

- 在超时优先玩家填充的机制下，可能在同一个房间中存在以下几种队形：包含单个真人玩家组的队伍、包含多个真人玩家组的队伍、真人玩家组与机器人混合的队伍、完全是机器人的队伍。MGOBE 不保证房间内的真人玩家

在每个队伍中平均分布。

- 该机制一般用于大乱斗、大逃杀类型的游戏，使人数较多的房间内，尽可能有较多的真人玩家。

相关文档

- 规则集配置
- 匹配配置
- 规则脚本设计
- 规则脚本示例

队组使用说明

最近更新时间：2022-03-29 14:13:04

> ⚠ 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。

使用说明

1. Group 类为 MGOBE 的子属性，用于实现玩家组成队组，并对队组进行管理。Group 类与 MGOBE 的其他子属性相互独立。例如，Room 中的玩家可以同时存在于 Group 中。
2. 根据类型可将 Group 分为 GROUP_LIMITED 和 GROUP_MANY，类型用于限制一个玩家可同时进入的 Group 数量。
3. 使用持久化队组时，注意记录队长 ID。持久化队组无法自动销毁，需要通过队长主动解散销毁。
4. 使用队组进行 组队匹配。

使用约束

- 玩家可**同时在一个队组或多个队组**，取决于队组的“队组类型”参数：
  - 1个玩家只能存在于1个类型为 GROUP_LIMITED 的队组。
  - 1个玩家可以同时存在于不多于5个类型为 GROUP_MANY 的队组。
- 队组的**持久化能力**取决于队组的“是否持久化”参数：
  - 一个游戏的免费持久化队组数量上限为**3个**。
  - 非持久化的队组，符合以下4个条件之一，即自动销毁。
    - 队组被队长解散。
    - 队组内所有玩家都退出或被移除。
    - 队组内所有玩家均掉线超过60min。
    - 队组存在超过24h。
- 持久化队组解散情景：队长主动调用 dismissGroup() 解散队组。
- 队组内**玩家数量上限**取决于队组的“队组类型”参数：
  - 一个类型为 GROUP_LIMITED 的队组，玩家数量上限为100。
  - 一个 GROUP_MANY 类型的队组，玩家数量上限为300。
- **消息发送频率限制**：一个队组内的免费消息发送频率上限为1条/玩家/秒。即在一个 Group 内，每个玩家调用 SentToClient() 的频率为1条/秒

队组结构

详情请参见 GroupInfo。

接口概览

1. 队组管理相关接口

接口	描述
initGroup()	初始化队组
onUpdate()	队组信息更新回调接口
createGroup()	创建队组
joinGroup()	加入队组
leaveGroup()	离开队组
dismissGroup()	解散队组
changeGroup()	更新队组信息
removeGroupPlayer()	移除队组内玩家
getGroupByGroupId()	通过 ID 获取队组信息
getMyGroups()	获取当前玩家队组信息
changeCustomGroupPlayerStatus()	修改队组玩家自定义状态
onJoinGroup()	玩家加入队组广播回调接口
onLeaveGroup()	玩家离开队组广播回调接口
onDismissGroup()	队组解散广播回调接口
onChangeGroup()	队组信息更新广播回调接口
onRemoveGroupPlayer()	队组内玩家被移除广播回调接口
onChangeGroupPlayerNetworkState()	队组内玩家网络状态变化广播回调接口
onChangeCustomGroupPlayerStatus()	玩家自定义状态变化广播回调接口

2. 消息发送相关接口

接口	描述

接口	描述
sendToClient()	发送消息给队组内玩家

实时服务器调用云 API

最近更新时间：2022-03-29 14:13:08

> **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

该文档指导您通过实时服务器直接调用云 API。下载示例代码 >>

操作步骤

步骤1：安装云 API Node.js SDK

1. 在实时服务器上使用云 API，您可以直接使用 云 API Node.js SDK。安装命令如下：

```
npm install tencentcloud-sdk-nodejs --save
```

2. 安装成功后，导入指定版本产品的 Client 对象（代码示例为 MgobeClient 对象）：

```
import * as tencentcloud from "tencentcloud-sdk-nodejs";

// 导入 client
// v20201014 为 api 版本号，可以从具体 api 文档上获得该参数信息
const MgobeClient = tencentcloud.mgobe.v20201014.Client;
```

步骤2：使用云 API

1. 登录访问管理控制台，在**API密钥管理**中，获取腾讯云 API 密钥。

> **说明**
> 关于腾讯云 API 密钥的创建和管理请参考 访问密钥 文档。

2. 实例化 Client 对象。

```
const MgobeClient = tencentcloud.mgobe.v20201014.Client;

const clientConfig = {
credential: {
secretId: "请填写腾讯云API密钥ID",
secretKey: "请填写腾讯云API密钥KEY",
},
region: "ap-shanghai", // 默认为上海地域
profile: {
httpProfile: {
endpoint: "mgobe.internal.tencentcloudapi.com", // 设置内部接入域名
}
}
};

// 实例化 client 对象
const client = new MgobeClient(clientConfig)
```

> **说明**
> 本地调试时无需实例化 Client 对象，否则会出现超时异常。建议与本地调试分开使用。

3. 调用云 API。以 查询房间信息 为例：

```
const describeRoom = (para: { GameId: string, RoomId: string }, callback: (err: any, res: a
ny) => any) => {
client.DescribeRoom(para)
.then(res => callback(null, res))
.catch(err => callback(err, null));
};
```

在 onRecvFromClient 广播 中可以直接调用并返回结果到客户端：

```
onRecvFromClient: function onRecvFromClient({ actionData, gameData, SDK, room, expor
ts }) {
// 查询房间
describeRoom({ GameId: "填写游戏ID", RoomId: "填写房间ID" }, (err, res) => {
if (err) {
```

```
try {
err = JSON.stringify(err);
} catch(e) {}
SDK.sendData({ playerIdList: [], data: { res, err: err + "", msg: "fail" } });
return;
}
// 查询成功
SDK.sendData({ playerIdList: [], data: { res, msg: "success" } });
return;
});
},
```

步骤3：打包发布代码

由于 tencentcloud-sdk-nodejs 安装在 node_modules 目录中，打包项目时需要注意将 node_modules 也上传到实时服务器。

在实时服务器使用云开发

最近更新时间：2022-03-29 14:13:13

> **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。

实时服务器集成游戏云开发，开发者可以在实时服务器上直接调用 腾讯云云开发 或 微信云开发 中的云函数、云数据库、云存储等服务。

腾讯云云开发

下面以云函数为例介绍如何在实时服务器中使用云开发，云开发中其它服务调用过程类似。

创建云函数

1. 登录 游戏联机对战引擎控制台。
2. 进入环境首页页面，单击**新建环境**。
3. 创建云开发环境后，并进入该环境总览页面，即可得到该环境 ID。

4. 在左侧菜单中单击**云函数**，创建一个名为 “mgobe_test” 的 Hello World 云函数。

云函数示例代码如下：

```
'use strict';
exports.main = (event, context, callback) => {
console.log("Hello World")
console.log(event)
console.log(event["non-exist"])
console.log(context)
callback(null, event);
};
```

获取 API 固定密钥对

1. 登录腾讯云 访问管理控制台，选择左侧菜单**访问密钥>API 密钥管理**。
2. 进入 API 密钥管理页面，单击**新建密钥**。

3. 密钥创建完成后，获得 SecretId 和 SecretKey。

在实时服务器调用云函数

1. 在实时服务器入口文件 index.ts 中实现 onInitGameServer 方法，填写环境 ID、SecretId、SecretKey。
2. 下面的示例代码实现了初始化云开发 TCB 对象，每次收到客户端消息后就调用云函数，并将结果返回。
3. 游戏 ID、后端密钥可以开通实时服务器后，在 MGOBE 控制台获取。

```
import { mgobexsInterface } from './mgobexsInterface';

const gameServer: mgobexsInterface.GameServer.IGameServer = {
mode: 'sync',
onInitGameData: function (): mgobexsInterface.GameData {
return {};
},
onRecvFromClient: function onRecvFromClient({ actionData, gameData, SDK, room, exports })
{
// 收到客户端消息就调用云函数
tcbApp.callFunction({
name: "mgobe_test",
data: { a: 1 },
}).then((res: any) => {
// 返回成功结果
SDK.logger.debug({ res });
SDK.sendData({ playerIdList: [], data: { res } });
SDK.exitAction();
}).catch((err: any) => {
// 返回失败结果
SDK.logger.debug({ err });
SDK.sendData({ playerIdList: [], data: { err } });
```

```
SDK.exitAction();
});
}
};

let tcbApp: any;

// 服务器初始化时调用
function onInitGameServer(tcb: any) {
// 初始化 TCB
tcbApp = tcb.init({
secretId: "请填写腾讯云API密钥 SecretId",
secretKey: "请填写腾讯云API密钥 SecretKey",
env: "请填写云开发环境",
serviceUrl: 'http://tcb-admin.tencentyun.com/admin',
timeout: 5000,
});
}

export const mgobexsCode: mgobexsInterface.mgobexsCode = {
logLevel: 'debug+',
logLevelSDK: 'debug+',
gameInfo: {
gameId: "请填写游戏 ID",
serverKey: "请填写后端密钥",
},
onInitGameServer,
gameServer
}
```

4. 将 index.ts 编译为 js，并且打包发布后，使用客户端 sendToGameSvr 接口发送消息，即可得到实时服务器返回的 TCB 云函数调用结果。

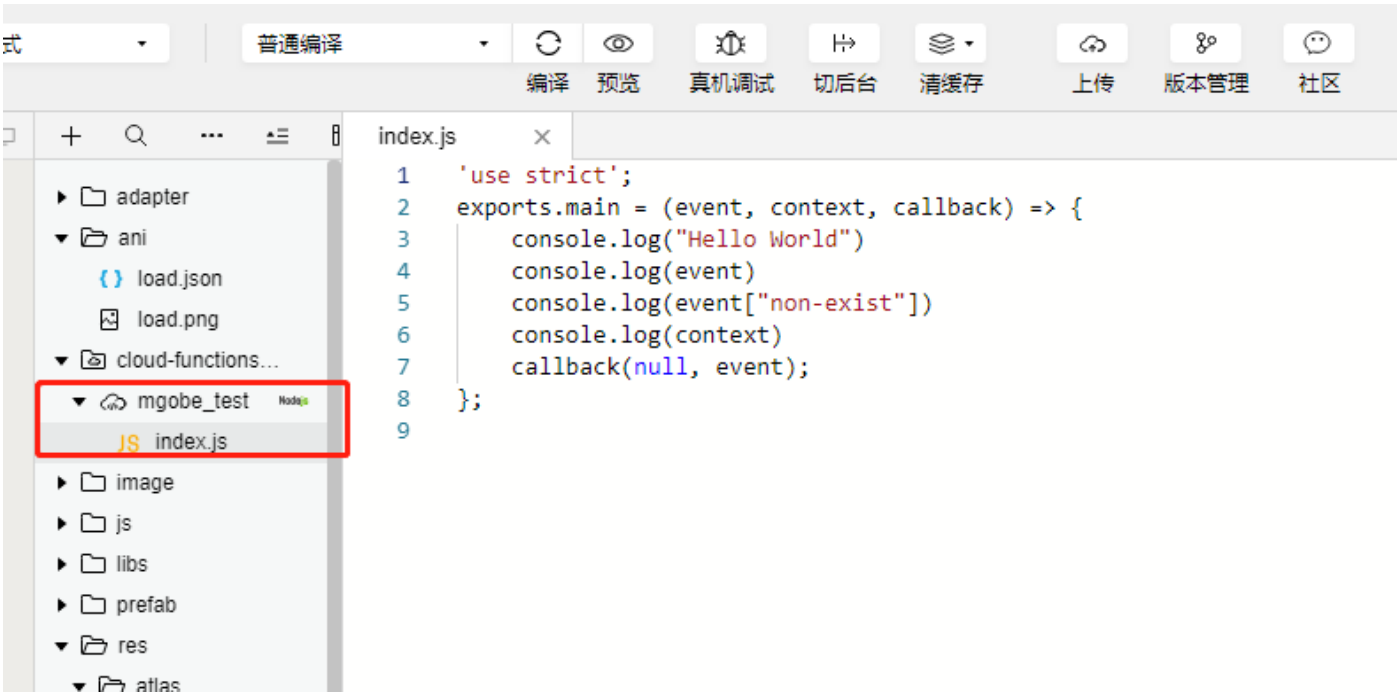
微信云开发

下面以云函数为例介绍如何在实时服务器中使用微信云开发，其它服务调用过程类似。

创建云函数

1. 在微信开发者工具中开通云开发，进入云开发控制台，获取环境 ID。

2. 单击【云函数】，创建一个名为 “mgobe_test” 的 Hello World 云函数。在微信编辑器中拉取云函数，可以进行编辑、上传。



云函数示例代码如下：

```
'use strict';
exports.main = (event, context, callback) => {
console.log("Hello World")
console.log(event)
console.log(event["non-exist"])
console.log(context)
callback(null, event);
};
```

获取 API 固定密钥对

1. 单击该 链接，使用微信小游戏的管理员账号登录，并授权当前小游戏账号进行登录。

2. 成功后，前往 API 密钥管理页面，创建 API 密钥，获取 SecretId、SecretKey。

在实时服务器调用云函数

该步骤与上文腾讯云云开发章节中的 在实时服务器调用云函数 一致，在实时服务器中初始化 TCB 对象后即可调用云开发中的各种服务。API 可参见腾讯云 云开发 SDK 文档。

使用签名初始化 SDK

最近更新时间：2022-03-29 14:13:17

> ⚠ 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。

在初始化 MGOBE SDK 时，为了避免在客户端泄露游戏项目的游戏密钥，可以使用签名的方式初始化 SDK。在开发者服务器通过游戏 ID、游戏密钥、玩家 openId 等信息计算出游戏签名，然后下发给客户端。客户端在初始化、掉线重连、前后台切换等场景中均会验证玩家签名信息。

签名计算方式

签名涉及到的字段如下：

字段名	类型	含义
game_id	string	游戏 ID
nonce	uint64	随机数
open_id	string	玩家 openId
timestamp	uint64	时间戳（单位：秒）
secretKey	string	游戏密钥

1. 拼接字符串 str：

```
// 注意字段的顺序为 game_id、nonce、open_id、timestamp
// 字段和值之间使用 = 连接
// 不同字段之间使用 & 连接
const str = "game_id=您的游戏ID&nonce=1655790837&open_id=玩家openId&timestamp=1571902273"
```

2. 计算字符串 str、secretKey 的 HMACSHA1 值，然后使用 BASE64 编码即可得到签名 sign：

```
// 使用 secretKey 作为密钥计算 str 的 HmacSHA1 值
// 然后使用 BASE64 编码
```

```
const sign = Base64.encode(HmacSHA1(str, secretKey));
```

3. 将 sign、nonce、timestamp 传给客户端 SDK 进行验证。

使用示例

服务端

服务端可使用一个 HTTPS 服务计算签名，然后将签名 sign、随机数 nonce、时间戳 timestamp 返回给客户端。计算签名的 TypeScript 示例代码如下：

```
const Base64 = require("crypto-js/enc-base64");
const HmacSHA1 = require("crypto-js/hmac-sha1");

/**
* 生成SDK初始化签名
* @param secretKey {string} 游戏密钥
* @param gameId {string} 游戏 ID
* @param openId {string} 玩家 ID
*/
function getSignature(secretKey: string, gameId: string, openId: string): { sign: string, nonce: n
umber, timestamp: number } {

// 随机正整数
const nonce = Math.floor(Math.random() * (Math.pow(2, 32) - 1));
// 时间戳，秒
const timestamp = Math.floor(Date.now() / 1000);

const fields = [
["game_id", gameId],
["nonce", nonce],
["open_id", openId],
["timestamp", timestamp],
].sort();

const str = fields.map((field) => field.join("=")).join("&");
const hmac = HmacSHA1(str, secretKey);
const sign = Base64.stringify(hmac);
```

```
return { sign, nonce, timestamp };
}

export default getSignature;
```

单击 这里，下载签名示例代码。

客户端

客户端在初始化 SDK 时，需要实现一个 createSignature 签名函数，从服务端获取签名信息然后回调给 SDK。示例代码如下：

```
const gameInfo = {
gameId: "xxxxx",
openId: "xxxxxx",
// 实现签名函数，初始化、掉线重连时会被调用
createSignature: callback => {
fetch("https://example.com/sign").then(rsp => rsp.json()).then(json => {
const sign = json.sign;
const nonce = json.nonce;
const timestamp = json.timestamp;

return callback({ sign, nonce, timestamp });
});
},
};

const config = {
url: "xxxxxxx.com",
};

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");
}
});
```

> **说明**
> 示例代码帮助您理解如何使用签名的方式初始化 SDK，具体业务里的服务端部署、请求方式、请求协议格式，由您自行实现。