游戏联机对战引擎

服务端 SDK

产品文档

腾讯云

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100。

文档目录

服务端 SDK
实时服务器
框架下载
使用简介
API
mgobexsCode 对象
GameServer.IGameServer 对象
ActionArgs 类型

服务端 SDK
实时服务器
框架下载

最近更新时间：2020-11-26 17:42:49

为方便您创建实时服务器，在下载 示例代码之前，请查阅 更新历史。

平台	更新时间	版本	示例代码	文档
Nodejs	2019/9/24	V1.1.0	下载	操作指南 开发指南

使用简介

最近更新时间：2020-11-26 17:43:05

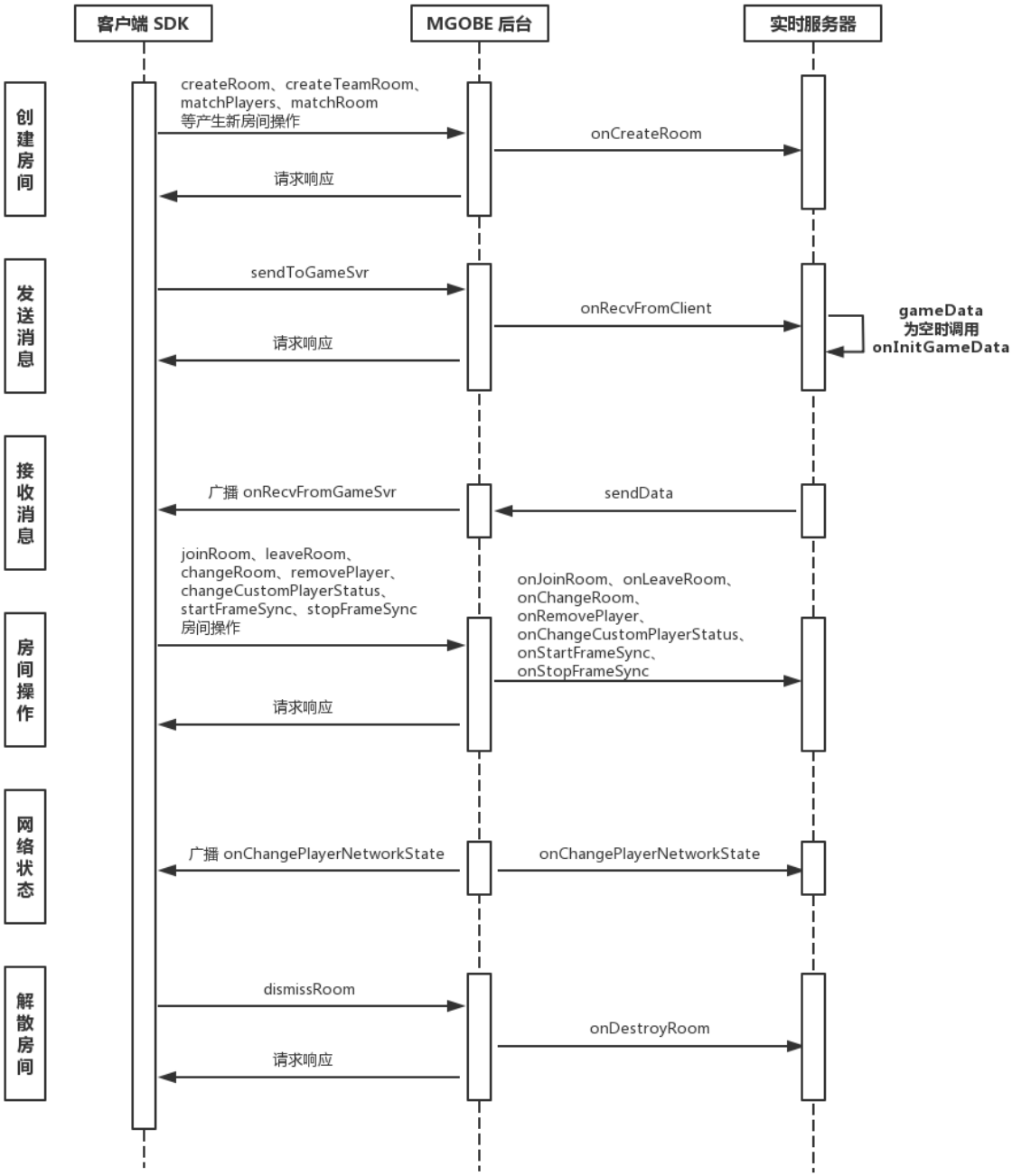
自定义服务逻辑的 实时服务器 部分包含两部分内容：

1. 房间的系列操作会自动触发到实时服务器逻辑。
2. 前端请求到实时服务器。当玩家加房成功后，可以使用客户端 SDK 中的 sendToGameSvr 方法直接与游戏服务器通信，实现游戏服务端拓展逻辑，如保存玩家数据，游戏状态同步等。

实时服务器时序图

> ⚠ 注意：
> onInitGameData 方法是在收到任意广播时检查 gameData，如果 gameData 为空，先执行 onInitGameData 再执行广播回调函数。

开通实时服务器后，客户端发起的房间相关操作，均被 MGOBE 后台广播到实时服务器。客户端、 MGOBE 后台、实时服务器三者交互时序图如下：



上传脚本

您只需要提供一个 index.js 脚本，然后按照以下文件夹结构压缩成 zip 文件，在 MGOBE 控制台上传即可。

```
├── mgobexs
│   └── index.js
```

> **注意：**
> zip 文件的根目录内只能有一个文件夹，建议命名为 mgobexs。mgobexs 目录下的 index.js 文件是实时服务器入口。

示例代码

您在接入实时服务器时，需要在 index.js 中导出一个 mgobexsCode 对象，该对象拥有一个 gameServer 属性。Node.js 示例代码如下：

```
exports.mgobexsCode = {
gameServer
};
```

在 gameServer 中需要实现 onInitGameData、onRecvFromClient、onCreateRoom、onJoinRoom 等接口。NodeJS 示例代码如下：

```
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 (args) {
args.SDK.logger.debug("onRecvFromClient");
args.SDK.exitAction();
},
// 监听加房广播
onJoinRoom: function (args) {
args.SDK.logger.debug("onJoinRoom");
},
// 监听创建房间广播
onCreateRoom: function (args) {
args.SDK.logger.debug("onCreateRoom");
},
// 监听退房广播
onLeaveRoom: function (args) {
```

```
args.SDK.logger.debug("onLeaveRoom");
},
// 监听玩家被移除广播
onRemovePlayer: function (args) {
args.SDK.logger.debug("onRemovePlayer");
},
// 监听房间销毁广播
onDestroyRoom: function (args) {
args.SDK.logger.debug("onDestroyRoom");
},
// 监听修改房间属性广播
onChangeRoom: function (args) {
args.SDK.logger.debug("onChangeRoom");
},
// 监听修改玩家自定义状态广播
onChangeCustomPlayerStatus: function (args) {
args.SDK.logger.debug("onChangeCustomPlayerStatus");
},
// 监听玩家网络状态变化广播
onChangePlayerNetworkState: function (args) {
args.SDK.logger.debug("onChangePlayerNetworkState");
},
// 监听开始帧同步广播
onStartFrameSync: function (args) {
args.SDK.logger.debug("onStartFrameSync");
},
// 监听停止帧同步广播
onStopFrameSync: function (args) {
args.SDK.logger.debug("onStopFrameSync");
}
};
```

查看日志

您使用 SDK.logger 输出的日志，您可前往控制台选择【实时服务器】，单击【查看日志】链接进行查看。

与客户端交互

实时服务器与客户端交互主要是以“发消息-监听消息”的方式进行。

客户端发送消息给实时服务器：

```
// 客户端代码，需要先进入房间
const sendToGameServerPara = {
data: {
cmd: 1,
},
};

room.sendToGameSvr(sendToGameServerPara, event => {
if (event.code === 0) {
console.log("发送成功")
}
});
```

客户端监听实时服务器返回的消息：

```
// 客户端代码
room.onRecvFromGameSvr = event => console.log("新消息", event);
```

实时服务器监听客户端发送的消息：

```
// 服务端代码
gameServer.onRecvFromClient = args => {

// 收到的数据
const actionData = args.actionData;
// 发送消息的玩家ID为
const sender = args.sender;

// 可以调用 args.SDK.sendData 方法发消息给客户端
args.SDK.sendData({ playerIdList: [], data: { msg: "hello" } });
};
```

实时服务器返回消息给客户端：

```
// 服务端代码
// 在各个广播监听回调函数里面使用 SDK 对象调用 sendData 方法
args.SDK.sendData({ playerIdList: [], data: { msg: "hello" } });
```

API
mgobexsCode 对象

最近更新时间：2020-11-26 17:43:32

mgobexsCode 对象是实时服务器的入口，您需要在代码中导出该对象。

gameServer 属性

描述

gameServer 是 mgobexsCode 对象的一个属性，类型为 GameServer.IGameServer 。您需要实现一个 GameServer.IGameServer 对象，并赋值给 gameServer。

使用示例

```
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 onRecvFromClient(args) {
args.SDK.logger.debug("onRecvFromClient");
// 发送消息给客户端
args.SDK.sendData({ playerIdList: [], data: { msg: "hello" } });
args.SDK.exitAction();
}
};

exports.mgobexsCode = {
gameServer
};
```

logLevelSDK 属性

描述

logLevelSDK 是 mgobexsCode 对象的一个属性，类型为字符串，表示实时服务器内部日志的打印级别。只能填写以下值：

值	含义
debug+	打印 debug、info、error
info+	打印 info、error
error+	打印 error（默认值）

使用示例

```
exports.mgobexsCode = {
logLevelSDK: "debug+",
gameServer
};
```

logLevel 属性

描述

logLevel 是 mgobexsCode 对象的一个属性，类型为字符串，表示您代码内使用 ActionArgs.SDK.logger 时的日志打印级别。只能填写以下值：

值	含义
debug+	打印 debug、info、error
info+	打印 info、error
error+	打印 error（默认值）

使用示例

```
exports.mgobexsCode = {
logLevel: "debug+",
gameServer
};
```

onInitGameServer 属性

描述

onInitGameServer 是 mgobexsCode 对象的一个属性，类型为 function。该函数在实时服务器初始化后会被调用，您可以在该函数内初始化 TCB 实例。

参数说明

参数名	类型	描述
tcb	object	腾讯云云开发模块

返回值说明

无。

使用示例

```
exports.mgobexsCode = {
onInitGameServer: (tcb) => {
// 可以在此初始化 TCB
const tcbApp = tcb.init({
secretId: "请填写腾讯云API密钥ID",
secretKey: "请填写腾讯云API密钥KEY",
env: "请填写云开发环境ID",
serviceUrl: 'http://tcb-admin.tencentyun.com/admin',
timeout: 5000,
});
},
gameServer
};
```

gameInfo 属性

描述

gameInfo 是 mgobexsCode 对象的一个属性，类型为 object。您如果需要在实时服务器调用 getRoomByRoomId、changeRoom、changeCustomPlayerStatus、removePlayer 方法需要实现该对象。该对象属性如下：

属性名	类型	描述
gameId	string	游戏 ID，从控制台获取
serverKey	string	后端密钥，从控制台获取

使用示例

```
exports.mgobexsCode = {
gameInfo: {
gameId: "请填写游戏ID，从控制台获取",
serverKey: "请填写后端密钥，从控制台获取",
},
gameServer
};
```

GameServer.IGameServer 对象

最近更新时间：2020-11-26 17:43:44

GameServer.IGameServer 对象即实时服务器接口。提供了接收客户端消息、监听房间广播相关接口。

mode 属性

描述

mode 是实时服务器处理客户端消息的模式。可以取值为 "sync" 或 "async"。

- 当 mode 为 "sync" 时，实时服务器将使用同步模式处理客户端消息。您在 onRecvFromClient 回调中必须显式调用 SDK.exitAction 方法，实时服务器才能处理下一条 onRecvFromClient 广播。
- 当 mode 为 "async" 时，实时服务器将使用异步模式处理客户端消息。每次监听到 onRecvFromClient 广播时都将执行回调函数。

使用示例

```
const gameServer = {};
gameServer.mode = "async";
```

onInitGameData 接口

描述

初始化游戏数据时的回调接口。

参数说明

参数名	类型	描述
args	{ room: IRoomInfo; }	回调参数

IRoomInfo 定义如下：

字段名	类型	描述
id	number	房间 ID
name	string	房间名称
type	string	房间类型
createType	CreateType	创建房间方式，参考 枚举类型 一节

字段名	类型	描述
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	IPlayerInfo[]	玩家列表
teamList	ITeamInfo[]	团队属性
frameSyncState	FrameSyncState	房间状态，参考 枚举类型 一节
frameRate	number	帧率
routeId	string	路由ID
createTime	number	房间创建时的时间戳，秒
startGameTime	number	开始帧同步时的时间戳，秒

ITeamInfo 定义如下：

字段名	类型	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

IPlayerInfo 定义如下：

字段名	类型	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

字段名	类型	描述
commonNetworkState	NetworkState	玩家在房间的网络状态，参考 枚举类型 一节，只取房间中的两种状态
relayNetworkState	NetworkState	玩家在游戏中的网络状态，参考 枚举类型 一节，只取游戏中的两种状态

返回值说明

在该接口需要返回一个 GameData 类型数据，该数据会作为游戏初始化数据，在整个房间被销毁之前都能使用。

GameData 默认为 object 类型，您可以根据需要进行自定义。

> 说明：
> onInitGameData 方法是在收到任意广播时检查 gameData，如果 gameData 为空，先执行 onInitGameData 再执行广播回调函数。

使用示例

```
const gameServer = {};
gameServer.onInitGameData = args => {
return { room: args.room };
};
```

onRecvFromClient 接口

描述

接收玩家消息回调接口。

参数说明

参数名	类型	描述
args	ActionArgs<UserDefinedData>	回调参数

ActionArgs 定义请参考 ActionArgs 对象。

UserDefinedData 即玩家的消息类型，类型为 object。您可以根据需要进行自定义。

返回值说明

无。

```
// 可以调用 args.SDK.sendData 方法发消息给客户端
args.SDK.sendData({ playerIdList: [], data: { msg: "hello" } });
args.SDK.exitAction();
};
```

onCreateRoom 接口

描述

创建房间广播回调接口。

参数说明

参数名	类型	描述
args	ActionArgs<ICreateRoomBst>	回调参数

ICreateRoomBst 定义如下：

字段名	类型	描述

字段名	类型	描述
roomInfo	IRoomInfo	房间信息

返回值说明

无。

使用示例

```
const gameServer = {};
gameServer.onCreateRoom = args => {

// 当前房间信息
const room = args.room;
// 游戏数据
const gameData = args.gameData;

// 收到的广播内容
// args.actionData 类型为 ICreateRoomBst
const bst = args.actionData;

args.SDK.logger.debug("onCreateRoom", bst.roomInfo);
};
```

onJoinRoom 接口

描述

玩家加房广播回调接口。

参数说明

参数名	类型	描述
args	ActionArgs<IJoinRoomBst>	回调参数

IJoinRoomBst 定义如下：

字段名	类型	描述
roomInfo	IRoomInfo	房间信息

字段名	类型	描述
joinPlayerId	string	加房玩家 ID

返回值说明

无。

使用示例

```
gameServer.onJoinRoom = args => {

// 当前房间信息
const room = args.room;
// 游戏数据
const gameData = args.gameData;

// 收到的广播内容
// args.actionData 类型为 IJoinRoomBst
const bst = args.actionData;

args.SDK.logger.debug("onJoinRoom", bst.joinPlayerId);
};
```

onLeaveRoom 接口

描述

玩家退房广播回调接口。

参数说明

参数名	类型	描述
args	ActionArgs<ILeaveRoomBst>	回调参数

ILeaveRoomBst 定义如下：

字段名	类型	描述
roomInfo	IRoomInfo	房间信息
leavePlayerId	string	退房玩家 ID

返回值说明

无。

使用示例

```
const gameServer = {};
gameServer.onLeaveRoom = args => {

// 当前房间信息
const room = args.room;
// 游戏数据
const gameData = args.gameData;

// 收到的广播内容
// args.actionData 类型为 ILeaveRoomBst
const bst = args.actionData;

args.SDK.logger.debug("onLeaveRoom", bst.leavePlayerId);
};
```

onRemovePlayer 接口

描述

移除玩家广播回调接口。

参数说明

参数名	类型	描述
args	ActionArgs<IRemovePlayerBst>	回调参数

IRemovePlayerBst 定义如下：

字段名	类型	描述
roomInfo	IRoomInfo	房间信息
removePlayerId	string	被移除玩家 ID

返回值说明

无。

使用示例

```
const gameServer = {};
gameServer.onRemovePlayer = args => {

// 当前房间信息
const room = args.room;
// 游戏数据
const gameData = args.gameData;

// 收到的广播内容
// args.actionData 类型为 IRemovePlayerBst
const bst = args.actionData;

args.SDK.logger.debug("onRemovePlayer", bst.removePlayerId);
};
```

onChangeRoom 接口

描述

修改房间属性广播回调接口。

参数说明

参数名	类型	描述
args	ActionArgs<IChangeRoomBst>	回调参数

IChangeRoomBst 定义如下：

字段名	类型	描述
roomInfo	IRoomInfo	房间信息

返回值说明

无。

使用示例

```
const gameServer = {};
gameServer.onChangeRoom = args => {
```

```
// 当前房间信息
const room = args.room;
// 游戏数据
const gameData = args.gameData;

// 收到的广播内容
// args.actionData 类型为 IChangeRoomBst
const bst = args.actionData;

args.SDK.logger.debug("onChangeRoom", bst.roomInfo);
};
```

onChangeCustomPlayerStatus 接口

描述

修改玩家自定义状态广播回调接口。

参数说明

参数名	类型	描述
args	ActionArgs<IChangeCustomPlayerStatusBst>	回调参数

IChangeCustomPlayerStatusBst 定义如下：

字段名	类型	描述
changePlayerId	string	玩家 ID
customPlayerStatus	number	玩家状态
roomInfo	IRoomInfo	房间信息

返回值说明

无。

使用示例

```
const gameServer = {};
gameServer.onChangeCustomPlayerStatus = args => {
```

```
// 当前房间信息
const room = args.room;
// 游戏数据
const gameData = args.gameData;

// 收到的广播内容
// args.actionData 类型为 IChangeCustomPlayerStatusBst
const bst = args.actionData;

args.SDK.logger.debug("onChangeCustomPlayerStatus", bst.changePlayerId);
};
```

onChangePlayerNetworkState 接口

描述

玩家网络状态变化广播回调接口。

参数说明

参数名	类型	描述
args	ActionArgs<IChangePlayerNetworkStateBst>	回调参数

IChangePlayerNetworkStateBst 定义如下：

字段名	类型	描述
changePlayerId	string	玩家 ID
networkState	NetworkState	网络状态，有四种情况，参考 枚举类型 一节
roomInfo	IRoomInfo	房间信息

返回值说明

无。

使用示例

```
const gameServer = {};
gameServer.onChangePlayerNetworkState = args => {

// 当前房间信息
```

```
const room = args.room;
// 游戏数据
const gameData = args.gameData;

// 收到的广播内容
// args.actionData 类型为 IChangePlayerNetworkStateBst
const bst = args.actionData;

args.SDK.logger.debug("onChangePlayerNetworkState", bst.changePlayerId);
};
```

onDestroyRoom 接口

描述

销毁房间广播回调接口。

参数说明

参数名	类型	描述
args	ActionArgs<IDestroyRoomBst>	回调参数

IDestroyRoomBst 定义如下：

字段名	类型	描述
roomInfo	IRoomInfo	房间信息

返回值说明

无。

使用示例

```
const gameServer = {};
gameServer.onDestroyRoom = args => {

// 当前房间信息
const room = args.room;
// 游戏数据
const gameData = args.gameData;
```

```
// 收到的广播内容
// args.actionData 类型为 IDestroyRoomBst
const bst = args.actionData;

args.SDK.logger.debug("onDestroyRoom", bst.roomInfo);
};
```

onStartFrameSync 接口

描述

开始帧同步广播回调接口。

参数说明

参数名	类型	描述
args	ActionArgs<IStartFrameSyncBst>	回调参数

IStartFrameSyncBst 定义如下：

字段名	类型	描述
roomInfo	IRoomInfo	房间信息

返回值说明

无。

使用示例

```
const gameServer = {};
gameServer.onStartFrameSync = args => {

// 当前房间信息
const room = args.room;
// 游戏数据
const gameData = args.gameData;

// 收到的广播内容
// args.actionData 类型为 IStartFrameSyncBst
const bst = args.actionData;
```

```
args.SDK.logger.debug("onStartFrameSync", bst.roomInfo);
};
```

onStopFrameSync 接口

描述

停止帧同步广播回调接口。

参数说明

参数名	类型	描述
args	ActionArgs<IStopFrameSyncBst>	回调参数

IStopFrameSyncBst 定义如下：

字段名	类型	描述
roomInfo	IRoomInfo	房间信息

返回值说明

无。

使用示例

```
const gameServer = {};
gameServer.onStopFrameSync = args => {

// 当前房间信息
const room = args.room;
// 游戏数据
const gameData = args.gameData;

// 收到的广播内容
// args.actionData 类型为 IStopFrameSyncBst
const bst = args.actionData;

args.SDK.logger.debug("onStopFrameSync", bst.roomInfo);
};
```

ActionArgs 类型

最近更新时间：2020-11-26 17:43:58

ActionArgs 是一个模板类型，其 TypeScript 定义如下：

```
export interface ActionArgs<T> {
sender: string;
actionData: T;
gameData: GameData;
room: IRoomInfo;
exports: { data: GameData; };
SDK: {
sendData: (data: { playerIdList: string[]; data: UserDefinedData; }, resendConf?: { tim
eout: number; maxTry: number; }) => void;
dispatchAction: (actionData: UserDefinedData) => void;
clearAction: () => void;
exitAction: () => void;

getRoomByRoomId: (getRoomByRoomIdPara: IGetRoomByRoomIdPara, callback?: ReqCallback<IGe
tRoomByRoomIdRsp>) => void;
changeRoom: (changeRoomPara: IChangeRoomPara, callback?: ReqCallback<IChangeRoomRsp>) =
> void;
changeCustomPlayerStatus: (changeCustomPlayerStatusPara: IChangeCustomPlayerStatusPara,
callback?: ReqCallback<IChangeCustomPlayerStatusRsp>) => void;
removePlayer: (removePlayerPara: IRemovePlayerPara, callback?: ReqCallback<IRemovePlaye
rRsp>) => void;

logger: {
debug: (...args: any[]) => void;
info: (...args: any[]) => void;
error: (...args: any[]) => void;
};
};
}
```

因此，模板类型指定了 actionData 的类型。例如，在 gameServer.onRecvFromClient 接口中，入参是 ActionArgs<UserDefinedData> 类型，表明 actionData 类型为 UserDefinedData。

sender 属性

描述

该属性在 gameServer.onRecvFromClient 中有效，其类型为 string，表示消息发送者的玩家 ID。

actionData 属性

描述

该属性在 gameServer 不同回调中的类型不同，表示该回调的响应数据。例如，在 gameServer.onRecvFromClient 中表示玩家发送给实时服务器的数据；在 onJoinRoom 表示加房广播数据；在 onLeaveRoom 中表示玩家退房广播数据。

gameData 属性

描述

该属性类型为 GameData，表示游戏数据，您可以用来实现游戏状态同步等功能。在第一次执行 gameServer.onCreateRoom 时会被初始化，在执行 gameServer.onDestroyRoom 时会被销毁。

room 属性

描述

该属性类型为 IRoomInfo，表示当前房间信息。

exports 属性

描述

该属性类型为 object，包含了一个类型为 GameData 的子属性 data，用于更新游戏数据 gameData。

如果您需要重新给 gameData 赋值，可以参考以下代码：

```
const newData = {};
exports.data = newData;
// ...
// 执行完回调函数之后，gameData 指向 newData
```

SDK 属性

描述

该属性类型为 object，包含了一系列实时服务器提供的方法。

SDK.sendData 方法

描述

实时服务器向客户端推送消息。

参数说明

参数名	类型	描述
data	{ playerIdList: string[]; data: UserDefinedData; }	消息内容

> 说明：
>
> - data.playerIdList 表示接收消息的玩家 ID 列表。数组为空表示发给房间内全部玩家。
> - data.data 为具体消息，类型为 UserDefinedData，即 object。

返回值说明

无。

使用示例

```
// 例子1：发给房间列表中第一个玩家
const id = room.playerList[0].id;
let data1 = { playerIdList: [id], data: { msg: "hello" } };
SDK.sendData(data1);

// 例子2：发给房间全部玩家
let data2 = { playerIdList: [], data: { msg: "hello" } };
SDK.sendData(data2);
```

SDK.dispatchAction 方法

描述

模拟客户端给实时服务器发送数据。

参数说明

参数名	类型	描述
actionData	UserDefinedData	消息内容

返回值说明

无。

> 说明：
>
> 使用该方法后，下次 gameServer.onRecvFromClient 接口回调将处理该方法发送的消息。

使用示例

```
let actionData = { data: "hello" };
SDK.dispatchAction(actionData);
```

SDK.clearAction 方法

描述

清空 onRecvFromClient 队列。

参数说明

无。

返回值说明

无。

> 说明：
> 当 gameServer.mode 为 "sync" 时，gameServer.onRecvFromClient 广播会保存在一个队列里面，在 gameServer.onRecvFromClient 回调函数中通过调用 SDK.exitAction 才能处理下一条 gameServer.onRecvFromClient 广播。SDK.clearAction 作用就是清空 gameServer.onRecvFromClient 队列，可用于游戏结束后实时服务器忽略客户端消息的场景。

使用示例

```
SDK.clearAction();
```

SDK.exitAction 方法

描述

结束 gameServer.onRecvFromClient 方法。

参数说明

无。

返回值说明

无。

> 说明：

> 当 gameServer.mode 为 "sync" 时，需要在 gameServer.onRecvFromClient 回调里面显式调用 SDK.exitAction 方法才能继续处理下一条 gameServer.onRecvFromClient 广播消息。

使用示例

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

SDK.logger 属性

描述

logger 是 SDK 提供的日志记录能力，可以使用 logger.debug、logger.info、logger.error 三种日志级别进行记录。记录的日志可以在 MGOBE 控制台的实时服务器页面查看。

SDK.getRoomByRoomId 方法

描述

根据房间 ID 查询房间信息。

参数说明

参数名	类型	描述
getRoomByRoomIdPara	IGetRoomByRoomIdPara	请求参数
callback	ReqCallback<IGetRoomByRoomIdRsp>	回调函数

IGetRoomByRoomIdPara 定义如下：

属性名	类型/值	描述
roomId	string	房间 ID

IGetRoomByRoomIdRsp 定义如下：

属性名	类型/值	描述
roomInfo	IRoomInfo	房间信息

返回值说明

无。

> 说明：

> 调用该接口需要在 mgobexsCode 配置正确的游戏 ID 和后端密钥。

使用示例

```
const getRoomByRoomIdPara = { roomId: "xxx", };
SDK.getRoomByRoomId(getRoomByRoomIdPara, event => {
console.log(event.code, event.data);

if (event.code === 0) {
// 操作成功
const roomInfo = event.data.roomInfo;
}
});
```

SDK.changeRoom 方法

描述

修改指定房间的房间信息。

参数说明

参数名	类型	描述
changeRoomPara	IChangeRoomPara	请求参数
callback	ReqCallback<IChangeRoomRsp>	回调函数

IChangeRoomPara 定义如下：

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

IChangeRoomRsp 定义如下：

属性名	类型/值	描述
roomInfo	IRoomInfo	房间信息

返回值说明

无。

> 说明：
> 调用该接口需要在 mgobexsCode 配置正确的游戏 ID 和后端密钥。

使用示例

```
const changeRoomPara = { roomId: "xxx", roomName: "xxx" };
SDK.changeRoom(changeRoomPara, event => {
console.log(event.code, event.data);

if (event.code === 0) {
// 操作成功
const roomInfo = event.data.roomInfo;
}
});
```

SDK.changeCustomPlayerStatus 方法

描述

修改指定房间的玩家自定义状态。

参数说明

参数名	类型	描述
changeCustomPlayerStatusPara	IChangeCustomPlayerStatusPara	请求参数
callback	ReqCallback<IChangeCustomPlayerStatusRsp>	回调函数

IChangeCustomPlayerStatusPara 定义如下：

属性名	类型/值	描述

属性名	类型/值	描述
roomId	string	房间 ID
playerId	string	玩家 ID
customPlayerStatus	number	玩家自定义状态

IChangeCustomPlayerStatusRsp 定义如下：

属性名	类型/值	描述
roomInfo	IRoomInfo	房间信息

返回值说明

无。

> 说明：
> 调用该接口需要在 mgobexsCode 配置正确的游戏 ID 和后端密钥。

使用示例

```
const changeCustomPlayerStatusPara = { roomId: "xxx", playerId: "xxx", customPlayerStat
us: 1 };
SDK.changeCustomPlayerStatus(changeCustomPlayerStatusPara, event => {
console.log(event.code, event.data);

if (event.code === 0) {
// 操作成功
const roomInfo = event.data.roomInfo;
}
});
```

SDK.removePlayer 方法

描述

在指定房间踢除玩家。

参数说明

参数名	类型	描述

参数名	类型	描述
removePlayerPara	IRemovePlayerPara	请求参数
callback	ReqCallback<IRemovePlayerRsp>	回调函数

IRemovePlayerPara 定义如下：

属性名	类型/值	描述
roomId	string	房间 ID
removePlayerId	string	玩家 ID

IRemovePlayerRsp 定义如下：

属性名	类型/值	描述
roomInfo	IRoomInfo	房间信息

返回值说明

无。

> 说明：
> 调用该接口需要在 mgobexsCode 配置正确的游戏 ID 和后端密钥。

使用示例

```
const removePlayerPara = { roomId: "xxx", removePlayerId: "xxx" };
SDK.removePlayer(removePlayerPara, event => {
console.log(event.code, event.data);

if (event.code === 0) {
// 操作成功
const roomInfo = event.data.roomInfo;
}
});
```