游戏联机对战引擎

最佳实践

产品文档

腾讯云

文档目录

最佳实践
Hello World
概述

最近更新时间：2022-03-29 14:10:36

> **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。

本文档介绍如何使用游戏联机对战引擎 SDK 在微信小游戏平台实现游戏帧同步服务。
重点介绍游戏联机对战引擎接口有房间匹配、退房、开始帧同步、停止帧同步、帧消息广播等。

主要步骤

1. 创建项目与导入 SDK
2. 添加工具类方法
3. 创建组件
4. 创建页面
5. 初始化 SDK
6. 实现接口功能

单击 这里，下载完整代码。如有更多问题请 联系我们。

创建项目与导入 SDK

最近更新时间：2022-03-29 14:10:41

> ⚠ 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

本文档主要介绍如何创建一个 Hello World 的小游戏项目和如何导入 SDK 的操作步骤。

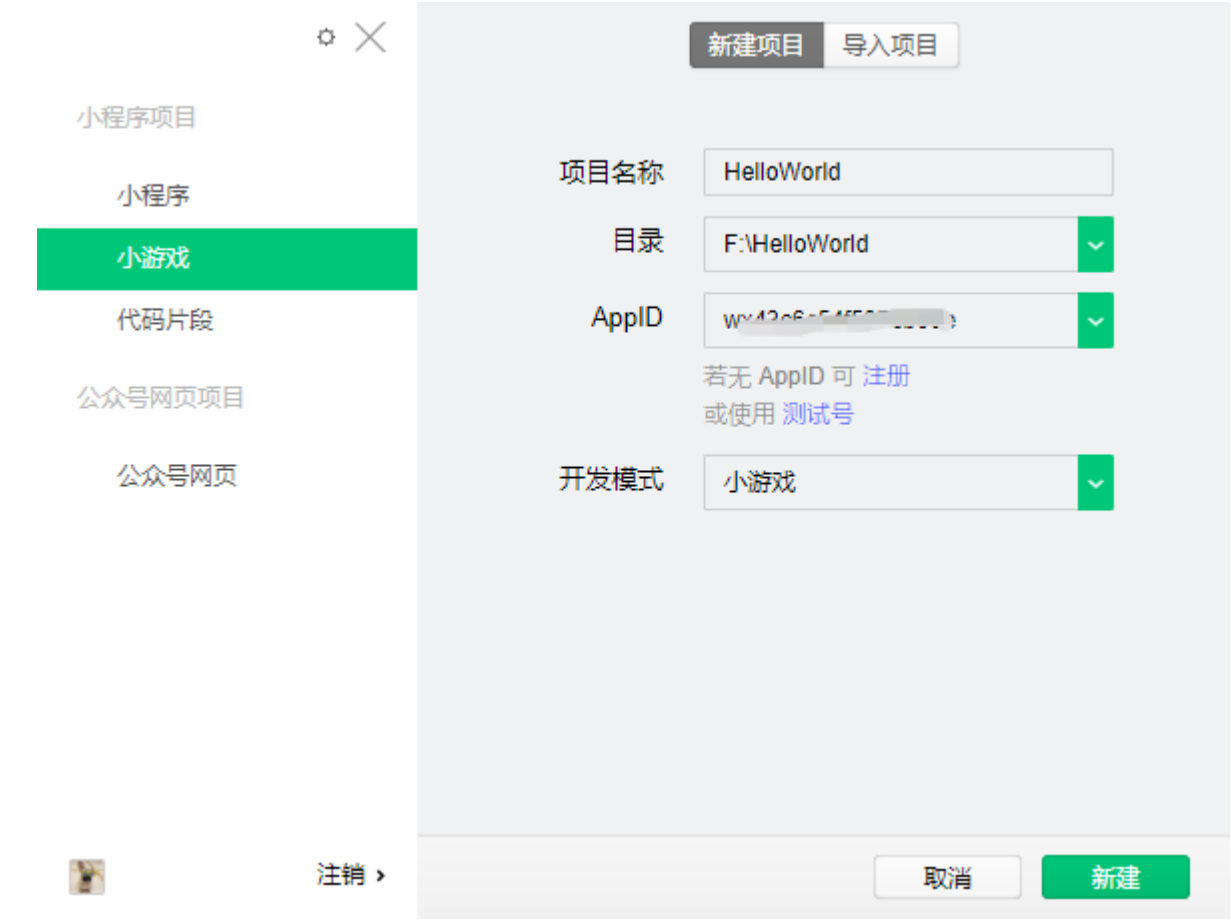
前提条件

- 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，详情可参考 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。SDK 需要对这两个参数进行校验。

操作步骤

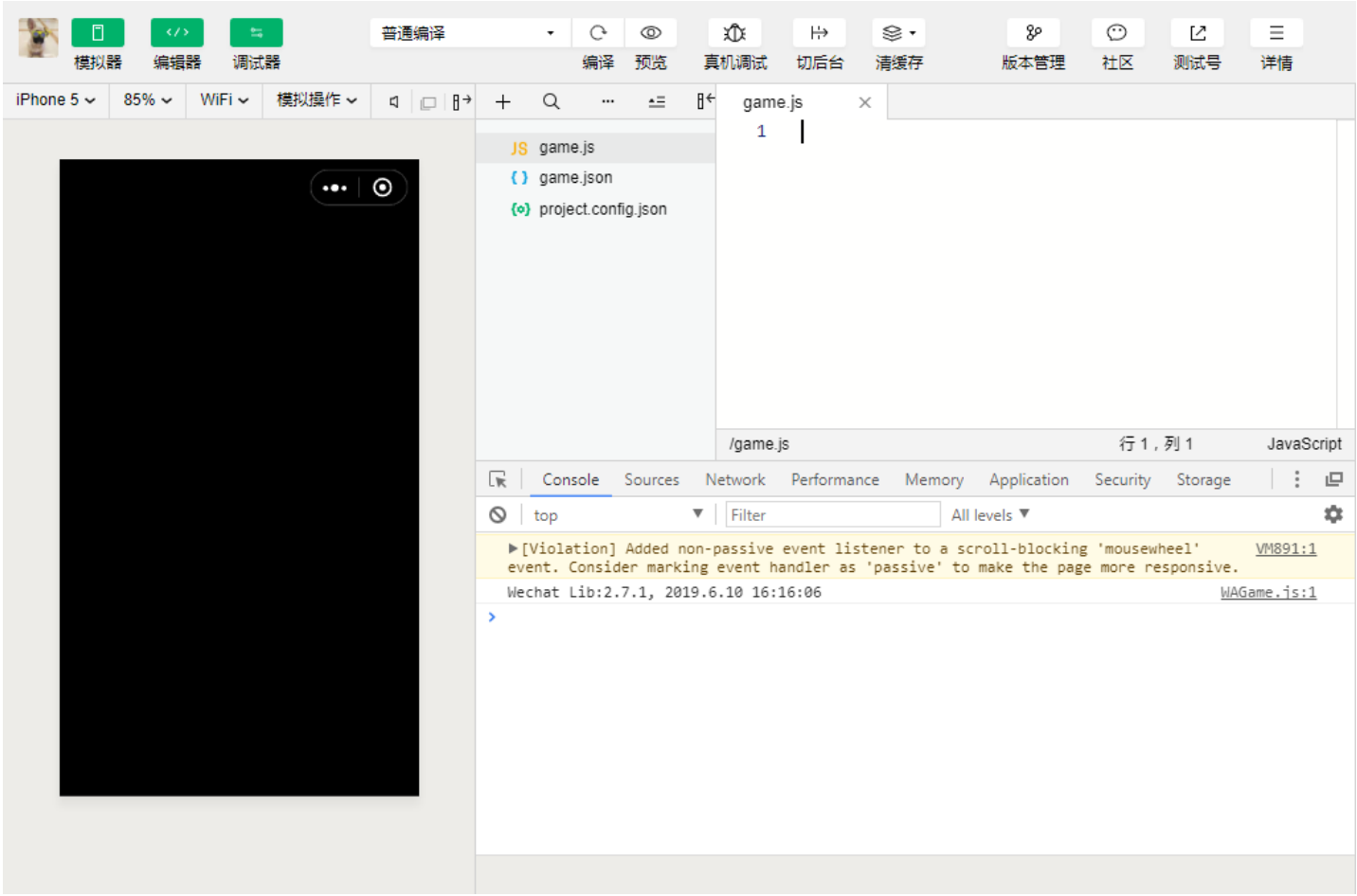
创建一个微信小游戏项目

1. 打开微信开发者工具，创建一个名为 HelloWorld 的小游戏项目。



2. 进入编辑界面后，删除多余的文件，只保留 game.js、game.json、project.config.json 三个文件，并清空 game.js 文件内容，如下图所示：

> **注意**
> 在开发过程中可以先跳过微信的域名检查。微信开发者工具上通过右侧**详情**设置，手机预览小程序/小游戏时打开调试功能跳过检查。



导入 MGOBE SDK

1. 将 MGOBE.js 添加到项目根目录。
2. 在 game.js 中添加如下代码，完成导入 MGOBE SDK。

```
// 导入 MGOBE.js
import "./MGOBE.js";
// 获取 Room、Listener 对象
const { Room, Listener, ErrCode, ENUM } = MGOBE;
```

添加工具类方法

最近更新时间：2022-03-29 14:10:54

> **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

本文档主要介绍如何添加工具类方法。

前提条件

已在根目录创建一个 Util.js 文件。

操作步骤

1. 在 Util.js 文件中，添加如下代码，设置一个对象，用于保存一些常用属性和方法。

```
// 模拟全局对象
export const Global = {
MainView: null,
RoomView: null,
GameView: null,
room: null,
ErrCode: null,
ENUM: null,
gameInfo: null,
name: "",
};
```

2. 在 Util.js 文件中，添加如下代码，用于生成 openId，在实际项目中，需要使用微信平台的接口获取 openId。

```
// 生成测试 openId
export const mockOpenId = () => "openId_" + Math.ceil(Math.random() * 100) + (new Date
()).getMilliseconds();
```

3. 在 Util.js 文件中，添加如下代码，用于生成 userName，在实际项目中，需要使用微信平台的接口获取玩家昵称。

```
// 生成测试 userName
export const mockName = () => "user_" + Math.ceil(Math.random() * 100);
```

4. 在 Util.js 文件中，添加如下代码，获得 canvas 对象上下文，以及画布宽度、高度信息。

```
// 获得 canvas 上下文
export const canvas = wx.createCanvas();
export const ctx = canvas.getContext('2d');
export const width = canvas.width;
export const height = canvas.height;
```

5. 在 Util.js 文件中，添加如下代码，准备基本的点击事件管理方法，例如，使用一个数组 clickHandlers 保存全部点击区域和点击回调，当监听到点击事件发生时，遍历该数组并执行点击回调。

```
// 点击事件回调函数数组
// Handler 结构：  [id, [x1, y1, x2, y2], callback]
let clickHandlers = [];
// 开启点击事件监听
wx.onTouchStart(function (e) {
const {
clientX,
clientY
} = e.touches[0];
clickHandlers.forEach((handler) => {
const x1 = handler[1][0];
const y1 = handler[1][1];
const x2 = handler[1][2];
const y2 = handler[1][3];
if (clientX > x1 && clientX < x2 && clientY > y1 && clientY < y2) {
handler[2] && handler[2]();
}
});
});
// 添加监听
export const onClick = (id, area, callback) => clickHandlers.push([id, area, callback]);
```

```
// 移除监听
export const offClick = (id) => clickHandlers = clickHandlers.filter(handler => handler[0] !==
= id);
```

创建组件

最近更新时间：2022-03-29 14:10:59

> **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。

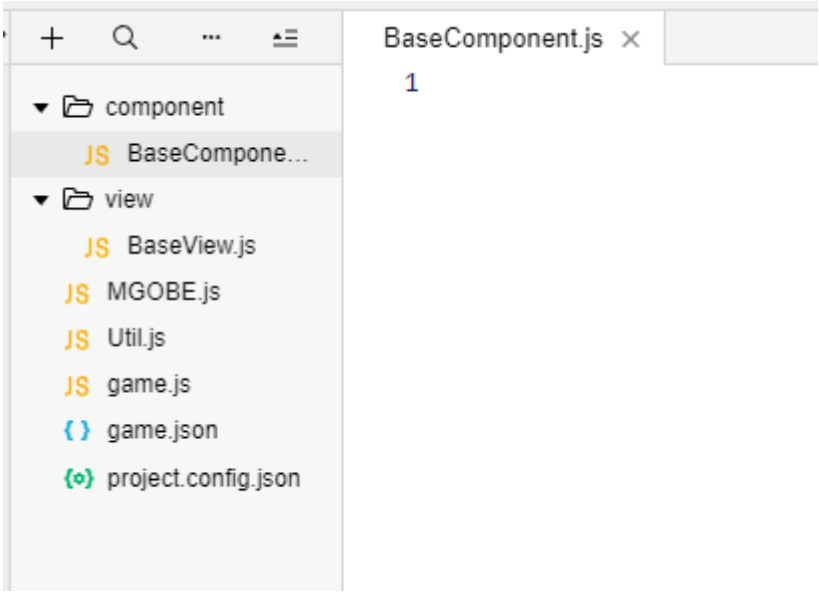
操作场景

组件是指页面上的按钮、文本框、输入框、下拉框等，每个组件需要维护组件 ID、点击事件、渲染、销毁等功能。本文档指导您如何实现按钮 Button 组件和文本框 MsgBox 组件。

操作步骤

创建组件基类

1. 在根目录新建两个文件夹，分别命名为 component 和 view，并在 component 中创建一个 BaseComponent.js 文件，在 view 中创建一个 BaseView.js 文件。如下图所示：



   - BaseComponent.js 文件：作为 Button 和 MsgBox 的基类。
   - BaseView.js 文件：作为页面的基类。

2. 在 BaseComponent.js 中添加以下代码：

```
// 组件基类
import * as Util from "../Util.js";
```

```
export default class BaseComponent {
id;
// 组件区域
area = [0, 0, 0, 0];
constructor() {
// 为组件生成 ID
this.id = Math.ceil(Math.random() * 10000) + "_" + Date.now();
}
// 子类实现具体绘制方法
render() { }
// 记录组件区域
setArea(x, y, w, h) {
this.area[0] = x;
this.area[1] = y;
this.area[2] = x + w;
this.area[3] = y + h;
}
// 注册点击事件
onClick(callback) {
this.offClick();
Util.onClick(this.id, this.area, callback);
}
// 取消点击事件
offClick() {
Util.offClick(this.id);
}
// 销毁
onDestroy() {
this.offClick();
}
}
```

实现按钮 Button

在 component 文件下，新建一个 Button.js 文件，用于实现一个基于 BaseComponent 的 Button 类。其 Button 类包含 render、setText 两个方法。

示例代码如下所示：

```
import * as Util from "../Util.js";
import BaseComponent from "./BaseComponent.js";

export default class Button extends BaseComponent {
x;
y;
text;

constructor(x, y, text) {
super();

this.x = x;
this.y = y;
this.text = text;
}

// 绘制
render() {
// 清除原图像
Util.ctx.lineWidth = 2;
Util.ctx.fillStyle = "white";

const x1 = this.area[0] - Util.ctx.lineWidth / 2;
const y1 = this.area[1] - Util.ctx.lineWidth / 2;
const x2 = this.area[2] + this.area[0] + Util.ctx.lineWidth / 2;
const y2 = this.area[3] + this.area[1] + Util.ctx.lineWidth / 2;
Util.ctx.fillRect(x1, y1, x2, y2);

// 绘制图形
Util.ctx.strokeStyle = "black";
Util.ctx.fillStyle = "black";
Util.ctx.font = "20px Arial";

const padding = 5;
const width = Util.ctx.measureText(this.text).width + padding * 2;
const height = 20 + padding * 2;
```

```
Util.ctx.strokeRect(this.x, this.y, width, height);
Util.ctx.fillText(this.text, this.x + padding, this.y + height - padding - 3);

// 记录按钮区域
this.setArea(this.x, this.y, width, height);
}

// 设置文本内容
setText(text) {
this.text = text;
this.render();
}
}
```

实现文本框 MsgBox

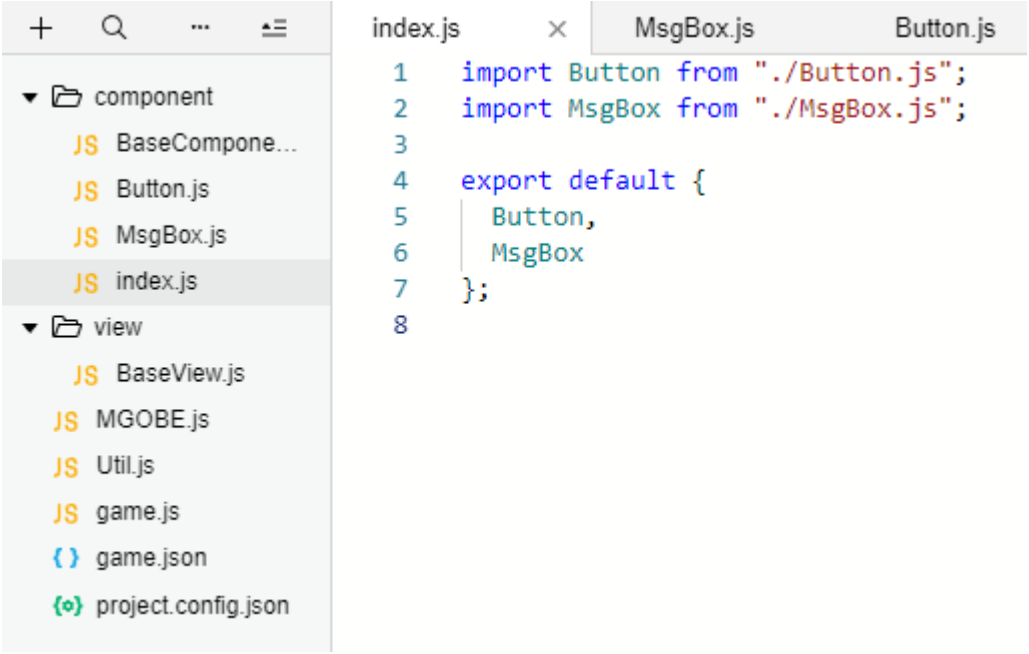
1. 在 component 文件夹下，新建一个 MsgBox.js 文件，用于实现一个基于 BaseComponent 的 MsgBox 类。MsgBox 类包含 render、setText 两个方法。
   示例代码如下所示：

```
import * as Util from "../Util.js";
import BaseComponent from "./BaseComponent.js";
export default class MsgBox extends BaseComponent {
x;
y;
text;
constructor(x, y, text) {
super();
this.x = x;
this.y = y;
this.text = text;
}
// 绘制
render() {
const width = Util.width - 2 * this.x;
const height = 300;
const padding = 5;
// 清除原图形
```

```
Util.ctx.strokeStyle = "black";
Util.ctx.fillStyle = "white";
Util.ctx.fillRect(this.x, this.y, width, height);
Util.ctx.strokeRect(this.x, this.y, width, height);
// 绘制图形
Util.ctx.strokeStyle = "black";
Util.ctx.fillStyle = "black";
Util.ctx.font = "15px Arial";
const texts = this.text.split("\n");
texts.forEach((t, i) => Util.ctx.fillText(t, this.x + padding, this.y + padding + (i + 1) * (15 + p
adding), width - padding * 2));
// 记录文本框区域
this.setArea(this.x, this.y, width, height);
}
// 设置文本内容
setText(text) {
this.text = text;
this.render();
}
}
```

2. 在 component 文件夹中，创建一个 index.js 文件，用于导出 Button.js、MsgBox.js 两个文件。
示例代码如下：

```
import Button from "./Button.js";
import MsgBox from "./MsgBox.js";
export default {
Button,
MsgBox
};
```


component
BaseCompone...
Button.js
MsgBox.js
index.js
view
BaseView.js
MGOBE.js
Util.js
game.js
game.json
project.config.json
index.js
MsgBox.js
Button.js
1 import Button from "./Button.js";
2 import MsgBox from "./MsgBox.js";
3
4 export default {
5 Button,
6 MsgBox
7 };
8

创建页面

最近更新时间：2022-03-29 14:11:05

> **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。

操作场景

一般帧同步游戏都包含至少三个页面：主页、房间、游戏页。
本文档指导您在 Hello World 项目中，分别创建实现 MainView 主页、RoomView 房间页和 GameView 游戏页三个页面。

前提条件

已在 view 文件夹下创建 BaseView.js 文件。

操作步骤

1. 在 BaseView.js 文件中，实现页面 onInit、onUpdate、onDestory 三个生命周期方法。并调用 requestAnimationFrame，用于驱动画面更新。示例代码如下所示：

```
// 页面基类
import * as Util from "../Util.js";
const Global = Util.Global;
export default class BaseView {
// 页面内组件数组，如 Button、MsgBox
components = [];
constructor() {
// 使用每个页面自己的 onUpdate 函数进行渲染
AnimateUtil.callback = this.onUpdate.bind(this);
// 初始化页面背景
Util.ctx.fillStyle = 'white';
Util.ctx.fillRect(0, 0, Util.width, Util.height);
wx.hideLoading();
this.onInit();
}
```

```
// 页面的生命周期
// onInit: 进入页面前调用
// onUpdate: 渲染层页面更新时调用
// onDestroy: 离开页面后调用
// 子类可以实现这三个方法
onInit() { }
onUpdate() { }
onDestroy() { }
// 跳转到其他页面
open(ViewClass) {
this.onDestroy();
// 清除页面组件
this.components.forEach(component => component.onDestroy());
this.components = [];
// 跳转
setTimeout(() => new ViewClass());
}
// 添加组件
addComponent(component) {
this.components.push(component);
component.render();
}
// 显示 toast
toast(title) {
wx.showToast({
title,
icon: 'none',
});
}
// 显示 loading
loading(title) {
wx.showLoading({
title,
});
}
// 显示 Dialog
dialog(content, confirm) {
wx.showModal({
```

```
title: '提示',
content,
success(res) {
if (res.confirm) {
confirm && confirm();
}
}
})
}
}
// 页面定时渲染
const AnimateUtil = {
callback: () => { },
run: () => {
AnimateUtil.callback && AnimateUtil.callback();
requestAnimationFrame(AnimateUtil.run);
}
}
AnimateUtil.run();
```

MainView 主页

1. 在 view 文件夹下，创建 MainView.js 文件，继承 BaseView 实现一个包含有一个按钮和文本框的页面。示例代码如下所示：

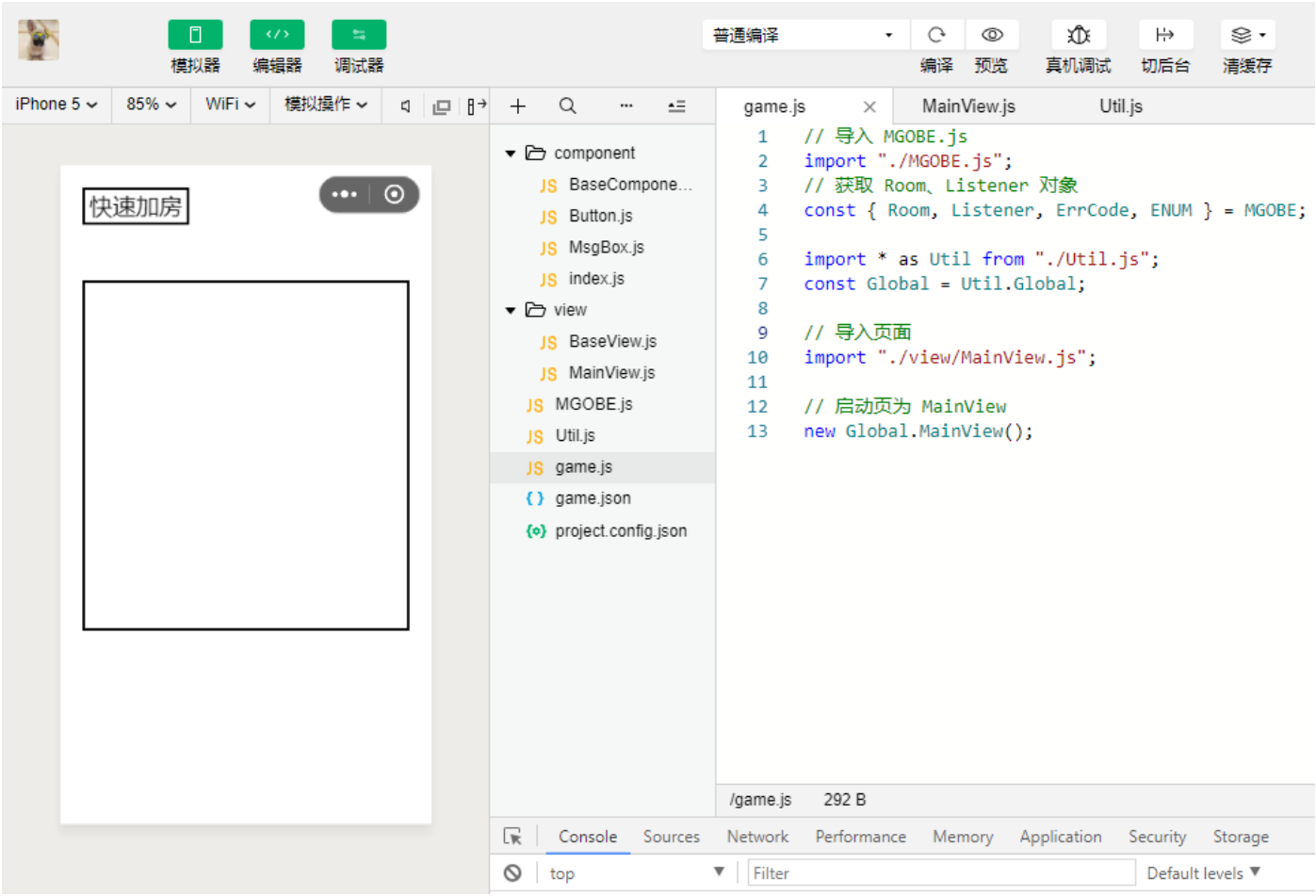
```
// 游戏主页
import * as Util from "../Util.js";
import BaseView from "./BaseView.js";
import Component from "../component/index.js";
const Global = Util.Global;
export default class MainView extends BaseView {
constructor() {
super();
}
onInit() {
const button = new Component.Button(20, 20, "快速加房");
const msgBox = new Component.MsgBox(20, 100, "");
this.addComponent(button);
```

```
this.addComponent(msgBox);
}
onUpdate() { }
onDestroy() { }
}
Global.MainView = MainView;
```

2. 在根目录的 game.js 中导入 Util.js 和 MainView.js，并实例化 MainView。示例代码如下所示：

```
// 导入 MGOBE.js
import "./MGOBE.js";
// 获取 Room、Listener 对象
const { Room, Listener, ErrCode, ENUM } = MGOBE;
import * as Util from "./Util.js";
const Global = Util.Global;
// 导入页面
import "./view/MainView.js";
// 启动页为 MainView
new Global.MainView();
```

3. 编译后即可在左侧模拟器中看到效果。如下图所示：



RoomView 房间页

1. 在 view 文件夹下，创建 RoomView.js 文件，用于实现一个包含有两个按钮和一个文本框的页面。示例代码如下所示：

```
// 房间页
import * as Util from "../Util.js";
import BaseView from "./BaseView.js";
import Component from "../component/index.js";
const Global = Util.Global;

export default class RoomView extends BaseView {

button;
msgBox;

constructor() {
```

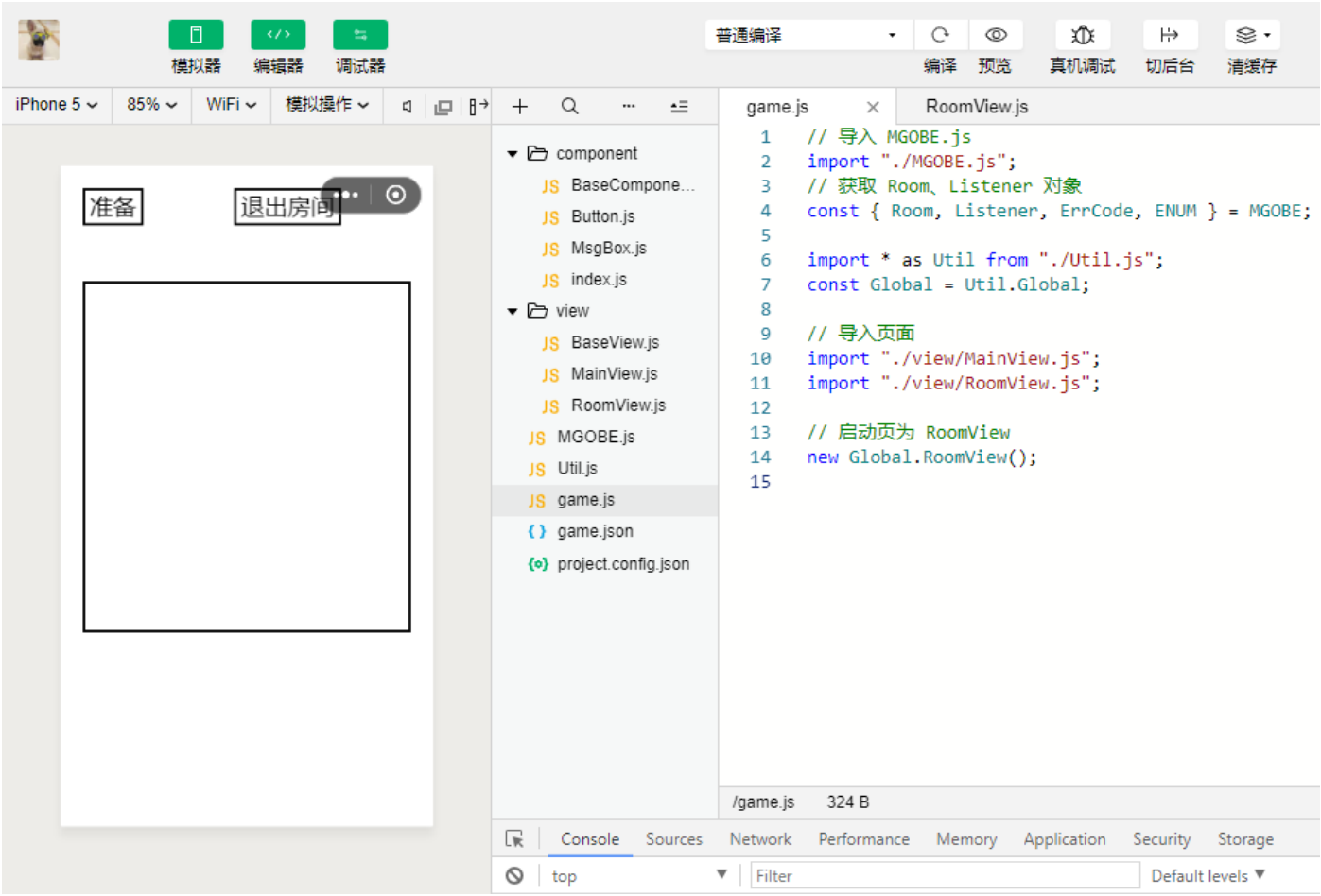
```
super();
}

timer;

onInit() {

const button1 = new Component.Button(20, 20, "准备");
this.button = button1;

const button2 = new Component.Button(150, 20, "退出房间");

const msgBox = new Component.MsgBox(20, 100, "");
this.msgBox = msgBox;

this.addComponent(button1);
this.addComponent(button2);
this.addComponent(msgBox);
}

onUpdate() { }

onDestroy() { }
}

Global.RoomView = RoomView;
```

2. 将 game.js 替换为以下代码，导入 RoomView.js，并将启动页换成 RoomView（也就是实例化 RoomView）。

```
// 导入 MGOBE.js
import "./MGOBE.js";
// 获取 Room、Listener 对象
const { Room, Listener, ErrCode, ENUM } = MGOBE;

import * as Util from "./Util.js";
const Global = Util.Global;
```

```
// 导入页面
import "./view/MainView.js";
import "./view/RoomView.js";

// 启动页为 RoomView
new Global.RoomView();
```

3. 编译后就能看到效果。如下图所示：



GameView 游戏页

1. 在 view 文件夹下,创建 GameView.js 文件，用于实现一个包含有一个按钮和一个文本框的页面。示例代码如下所示：

```
// 帧同步游戏页面
import * as Util from "../Util.js";
import BaseView from "./BaseView.js";
import Component from "../component/index.js";
const Global = Util.Global;
```

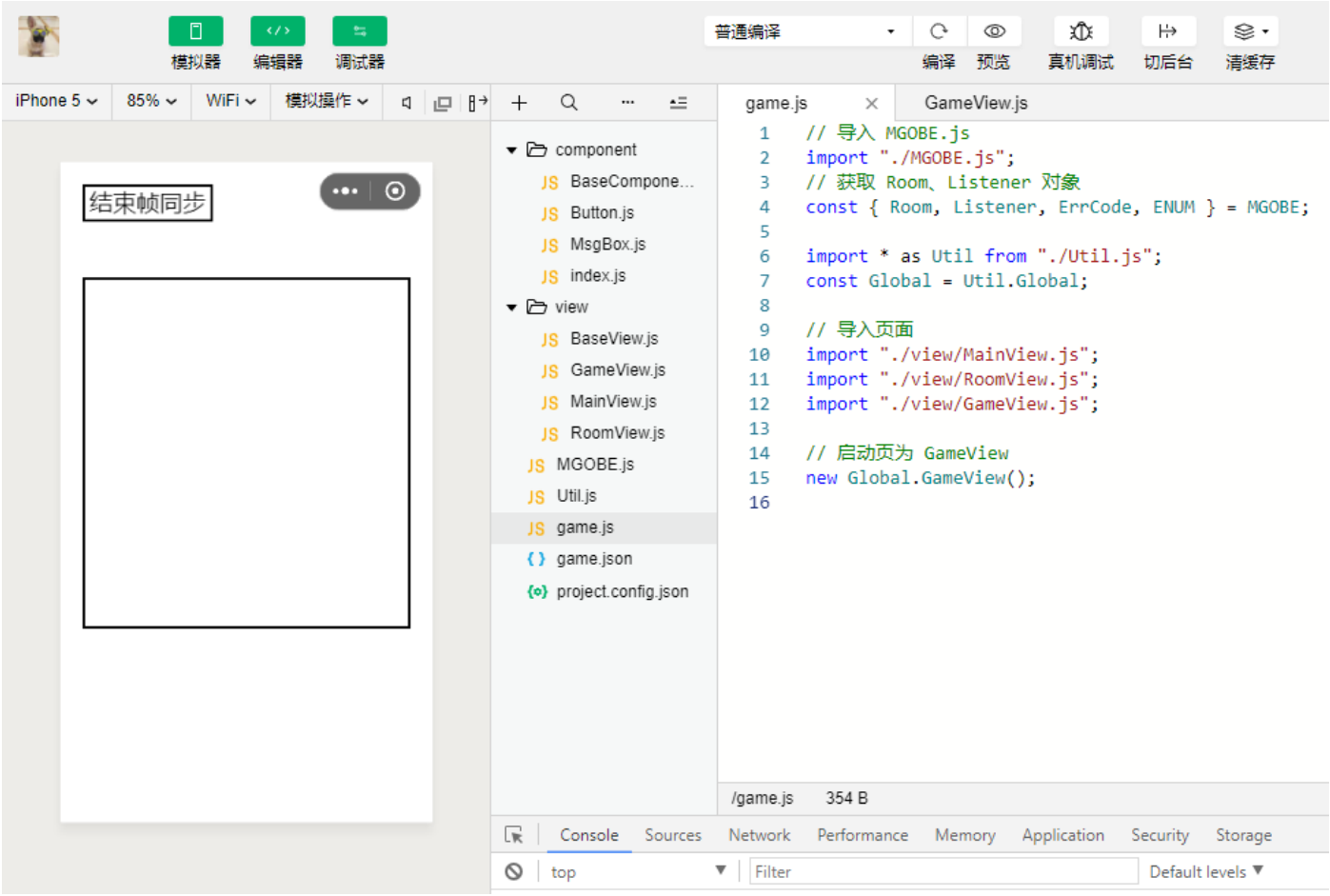
```
export default class GameView extends BaseView {

msgBox;

constructor() {
super();
}

onInit() {
const button = new Component.Button(20, 20, "结束帧同步");

const msgBox = new Component.MsgBox(20, 100, "");
this.msgBox = msgBox;

this.addComponent(button);
this.addComponent(msgBox);
}

onUpdate() { }

onDestroy() { }
}

Global.GameView = GameView;
```

2. 将 game.js 替换为以下代码，导入 GameView.js，并将启动页换成 GameView。

```
// 导入 MGOBE.js
import "./MGOBE.js";
// 获取 Room、Listener 对象
const { Room, Listener, ErrCode, ENUM } = MGOBE;

import * as Util from "./Util.js";
const Global = Util.Global;

// 导入页面
import "./view/MainView.js";
```

```
import "./view/RoomView.js";
import "./view/GameView.js";

// 启动页为 GameView
new Global.GameView();
```

3. 编译后就能看到效果。如下图所示：



初始化 SDK

最近更新时间：2022-03-29 14:11:11

> **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。

操作场景

本文档指导您如何初始化 SDK。

操作步骤

1. 在 game.js 文件中，将启动页改为 MainView。
2. 完成 SDK 监听器初始化、实例化 Room 对象。玩家 playerId 通过使用 Util.js 中的 mockPlayerId 方法生成。

game.js 最终代码如下所示：

```
// 导入 MGOBE.js
import "./MGOBE.js";
// 获取 Room、Listener 对象
const { Room, Listener, ErrCode, ENUM } = MGOBE;

import * as Util from "./Util.js";
const Global = Util.Global;

// 导入页面
import "./view/MainView.js";
import "./view/RoomView.js";
import "./view/GameView.js";

const gameInfo = {
// 随机生成 玩家 ID
openId: Util.mockOpenId(),
```

```
// 替换 为控制台上的“游戏 ID”
gameId: "xxxxxx",
// 替换 为控制台上的“游戏 Key”
secretKey: 'xxxxxxxxxxxxxxxxxxx',
};

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

// 初始化 Listener
Listener.init(gameInfo, config, event => {
if (event.code === ErrCode.EC_OK) {

console.log("初始化成功");

// 接收广播
Listener.add(room);

// 启动页为 MainView
new Global.MainView();
} else {
console.error("初始化失败", event);
}
});

// 实例化 Room 对象
const room = new Room();

// 保存常用对象的引用
Global.room = room;
Global.ErrCode = ErrCode;
```

```
Global.ENUM = ENUM;
Global.gameInfo = gameInfo;
```

实现接口功能

最近更新时间：2022-03-29 14:11:19

> **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

本文档指导您如何实现，从匹配加房到开始帧同步、结束帧同步的流程。

前提条件

已创建 MainView.js、RoomView.js 、GameView.js 文件。

操作步骤

实现快速加房

快速加房操作在 MainView 主页完成。交互逻辑是玩家单击**快速加房**后，SDK 发起房间匹配请求，请求成功后玩家将加入房间，此时需跳转至 RoomView 房间页。

1. 在 MainView.js 的 onInit 方法中为 button 绑定点击事件。
2. 为 MainView 类添加 matchRoom 方法。
3. 编译代码后，单击**快速加房**，模拟器页面将跳转到 RoomView 房间页。

MainView 主页示例代码如下：

```
// 游戏主页
import * as Util from "../Util.js";
import BaseView from "./BaseView.js";
import Component from "../component/index.js";
const Global = Util.Global;

export default class MainView extends BaseView {
constructor() {
super();
}
```

```
onInit() {
const button = new Component.Button(20, 20, "快速加房");
// 绑定点击事件
button.onClick(() => this.matchRoom());

const msgBox = new Component.MsgBox(20, 100, "");

this.addComponent(button);
this.addComponent(msgBox);
}

// 房间匹配方法
matchRoom() {

this.loading('匹配中...');

const playerInfo = {
name: Util.mockName(),
};

Global.name = playerInfo.name;

const matchRoomPara = {
playerInfo,
maxPlayers: 2,
roomType: "1",
};

// 调用房间匹配接口实现快速加房
Global.room.matchRoom(matchRoomPara, event => {
wx.hideLoading();
if (event.code === Global.ErrCode.EC_OK) {
// 接口调用成功，跳转到 RoomView
return this.open(Global.RoomView);
}

return this.toast("匹配失败" + event.code + " " + event.msg);
```

```
});
}

onUpdate() {}

onDestroy() {}
}

Global.MainView = MainView;
```

实现修改玩家状态

RoomView 房间页有两个按钮，分别是**准备**、**退出房间**。交互逻辑是单击**准备**将修改玩家自定义状态，当房间内全部玩家都准备好后，可以开始帧同步进行游戏。单击**退出房间**，玩家将发起退房请求，成功后进入 MainView 主页。

1. 为**准备**绑定点击事件。在 RoomView.js 的 onInit 中为 button1 添加事件。示例代码如下：

```
const button1 = new Component.Button(20, 20, "准备");
// 点击后切换状态
button1.onClick(() => this.changeCustomPlayerStatus(this.customPlayerStatus === 1 ? 0 :
1));

this.button = button1;
```

2. 0和1表示玩家“未准备”、“已准备”两种状态。customPlayerStatus 为 RoomView 类的属性，用来保存玩家状态。示例代码如下：

```
export default class RoomView extends BaseView {

button;
msgBox;

customPlayerStatus;

// ...
```

3. 为 RoomView 类添加 changeCustomPlayerStatus 方法。示例代码如下：

```
// 修改用户自定义状态
changeCustomPlayerStatus(customPlayerStatus) {
const changeCustomPlayerStatusPara = {
customPlayerStatus
};

Global.room.changeCustomPlayerStatus(changeCustomPlayerStatusPara, event => {
if (event.code !== Global.ErrCode.EC_OK) {
return this.toast("操作失败" + event.code);
}

return this.customPlayerStatus = customPlayerStatus;
});
}
```

检查房间属性

MGOBE SDK 中的 Room 对象会管理所有房间变化，任何玩家进行加房、退房、修改玩家状态等会改变房间信息的操作后，Room 对象会自动更新房间信息。
因此，玩家在 RoomView 房间页修改状态后，只需要访问 Room 实例的 roomInfo 属性就能获得最新房间信息，页面也可以同步更新。可以直接使用 Room 实例的 onUpdate 方法检查房间属性并更新页面。

1. 为 RoomView 类添加 onRoomUpdate 方法检查房间最新属性，更新按钮和文本框，并当全部玩家处于**已准备**状态时跳转到 GameView 游戏页。示例代码如下：

```
onRoomUpdate() {
// 更新 按钮
const playerId = MGOBE.Player.id;
const player = Global.room.roomInfo.playerList.find(player => player.id === playerId);

if (player) {
this.customPlayerStatus = player.customPlayerStatus;

// 更新 按钮 文字
if (player.customPlayerStatus === 0) {
this.button.setText("准备");
} else {
this.button.setText("取消准备");
```

```
}
}

// 更新 MsgBox
let msg = "房间ID:\n" + Global.room.roomInfo.id + "\n玩家列表:\n";
Global.room.roomInfo.playerList.forEach((player, i) => {
msg += i + ":" + player.id + (player.customPlayerStatus === 1 ? " 已准备" : " 未准备") + "\n"
;
});

this.msgBox.setText(msg);

if (Global.room.isInRoom() && !Global.room.roomInfo.playerList.find(player => player.custo
mPlayerStatus !== 1)) {
// 全部玩家准备好就跳转
setTimeout(() => this.open(Global.GameView), 1000);
}
}
```

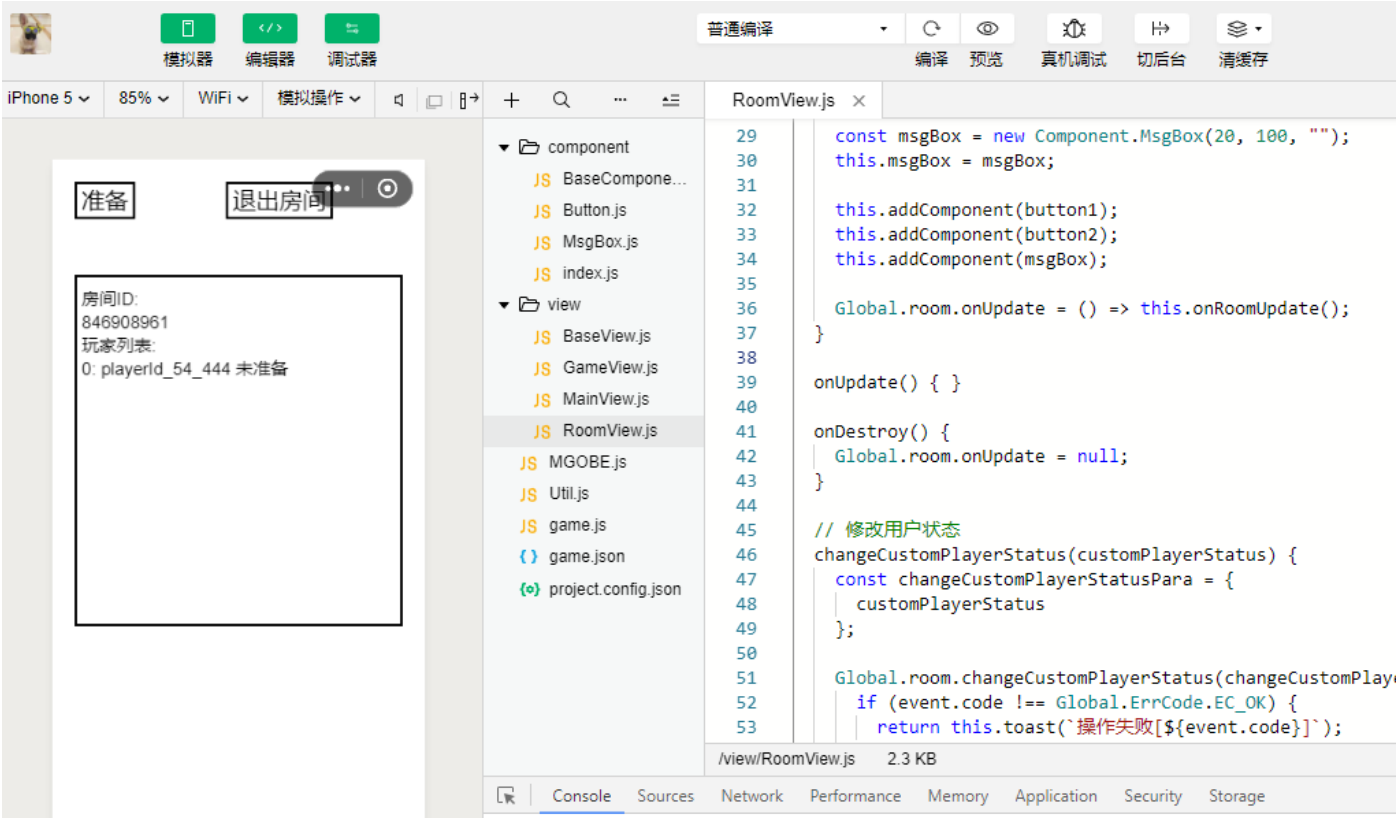
2. 在 onInit 追加代码，绑定 onRoomUpdate。示例代码如下：

```
onInit() {
// ...
Global.room.onUpdate = this.onRoomUpdate.bind(this);
// 刷新canvas
this.onRoomUpdate();
}
```

3. 在 onDestroy 中解除绑定 onRoomUpdate。示例代码如下：

```
onDestroy() {
Global.room.onUpdate = null;
}
```

4. 编译代码，玩家单击**快速加房**进入 RoomView 房间页后，单击**准备**，按钮和文本框都将更新。如果房间内的玩家都是**已准备**状态，页面将跳转到 GameView 游戏页。



处理其他玩家加房

1. 当其他玩家进入房间时，SDK 会触发 onJoinRoom 回调。因此可以在进入页面时为房间绑定 onJoinRoom 广播回调函数。修改 RoomView.js 中的 onInit 方法。示例代码如下所示：

```
onInit() {
// ...

// 设置广播回调
Global.room.onJoinRoom = this.onJoinRoom.bind(this);
}
```

2. 添加 onJoinRoom 方法，这里做一个 toast 提醒玩家即可。示例代码如下所示：

```
// 加房广播
onJoinRoom(event) {
this.toast("新玩家加入");
}
```

> ⚠ 注意

> Global.room.onJoinRoom 指定了一个回调函数，当页面跳转到其他页面时，这个回调函数引用还在。如果该函数依赖于页面脚本的上下文，有可能在触发广播时报错。

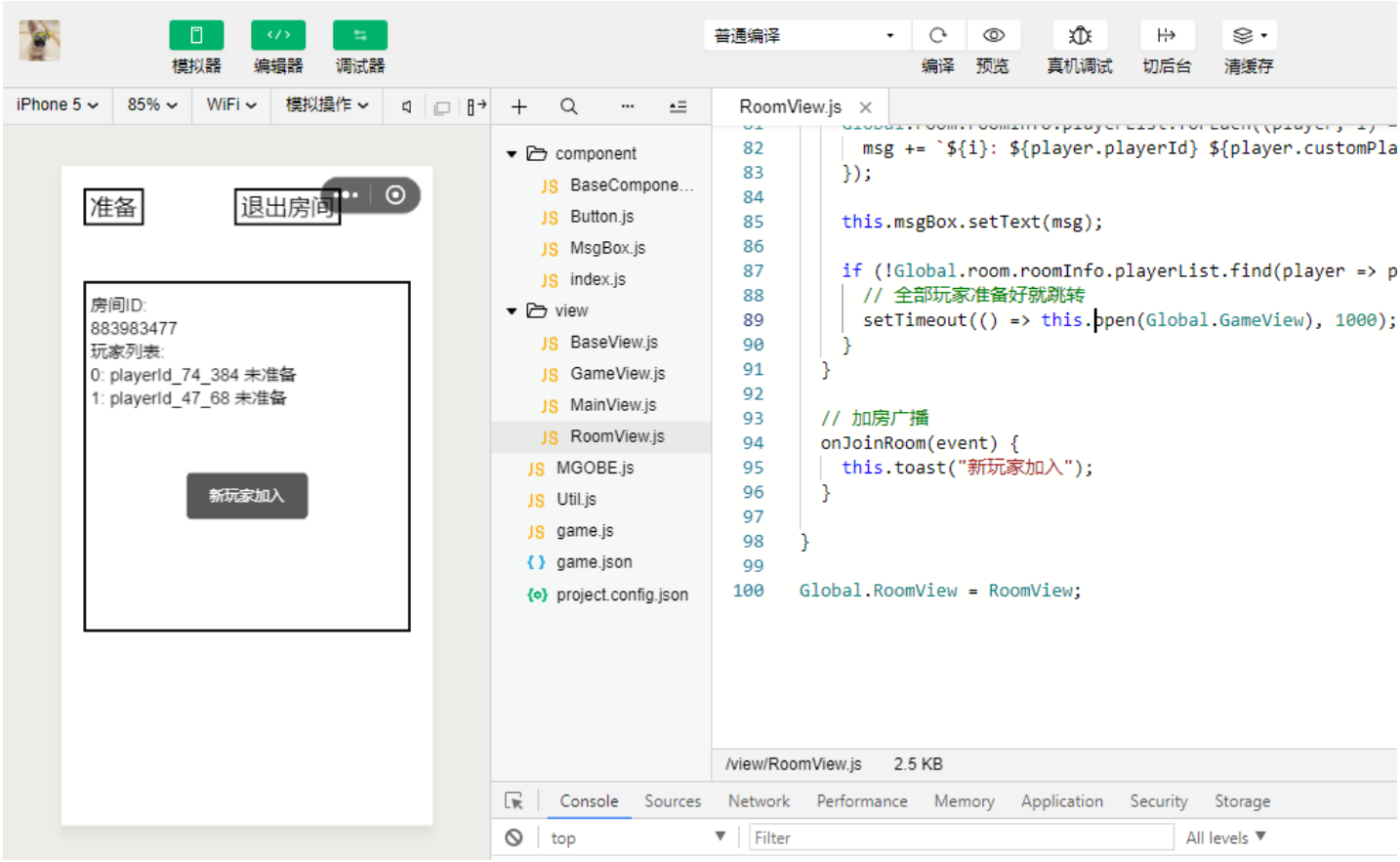
3. 离开页面时置空 Global.room.onJoinRoom。示例代码如下所示：

```
onDestroy() {
Global.room.onUpdate = null;
Global.room.onJoinRoom = null;
}
```

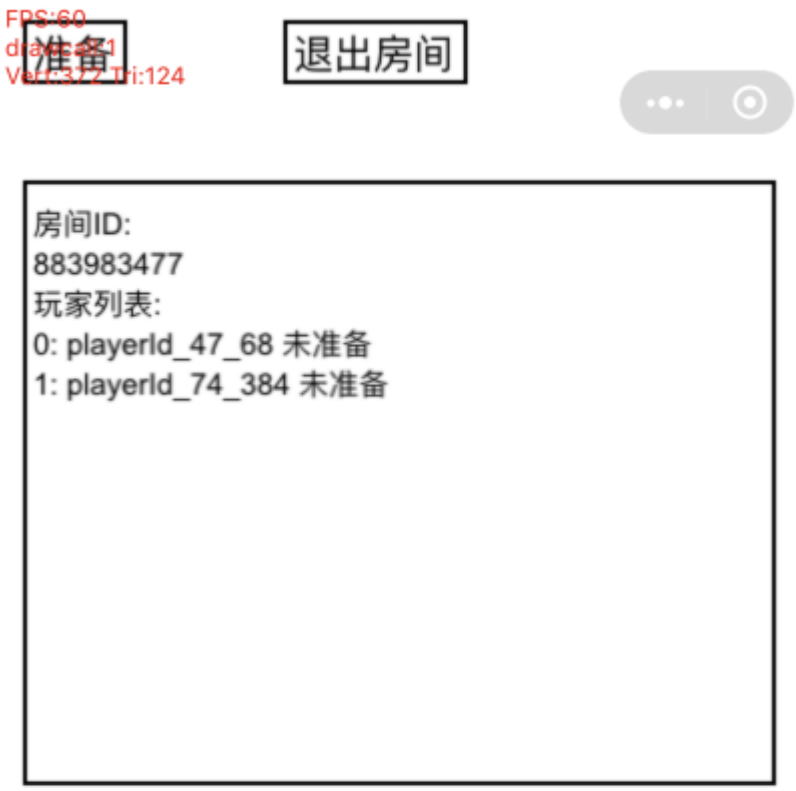
4. 使用两个设备进行“快速加房”操作（如模拟器+手机预览），两个玩家有可能匹配在一起，这时页面上将弹出“新玩家加入”提示。

> **注意**
> 手机预览时打开调试，跳过域名检查。

模拟器界面效果图如下所示：



手机预览效果图如下所示：



实现退房功能

1. 在 RoomView 房间页中为 button2 绑定点击事件。示例代码如下：

```
onInit() {

// ...

const button2 = new Component.Button(150, 20, "退出房间");
button2.onClick(() => this.leaveRoom());

// ...
}
```

2. 在 RoomView 类中实现 leaveRoom 方法。示例代码如下：

```
// 退出房间
leaveRoom() {
Global.room.leaveRoom({}, event => {
if (event.code !== Global.ErrCode.EC_OK && event.code !== Global.ErrCode.EC_ROOM_PLA
YER_NOT_IN_ROOM) {
return this.toast("操作失败" + event.code);
}
return this.open(Global.MainView);
});
}
```

RoomView 房间页最终示例代码如下：

```
// 房间页
import * as Util from "../Util.js";
import BaseView from "./BaseView.js";
import Component from "../component/index.js";
const Global = Util.Global;

export default class RoomView extends BaseView {

button;
msgBox;

// 自定义玩家状态：“0和1”表示玩家“未准备”、“已准备”
customPlayerStatus;

constructor() {
super();
}

onInit() {
const button1 = new Component.Button(20, 20, "准备");
// 点击后切换状态
button1.onClick(() => this.changeCustomPlayerStatus(this.customPlayerStatus === 1 ? 0 : 1
));
```

```
this.button = button1;

const button2 = new Component.Button(150, 20, "退出房间");
button2.onClick(() => this.leaveRoom());

const msgBox = new Component.MsgBox(20, 100, "");
this.msgBox = msgBox;

this.addComponent(button1);
this.addComponent(button2);
this.addComponent(msgBox);

Global.room.onUpdate = this.onRoomUpdate.bind(this);
// 刷新canvas
this.onRoomUpdate();
// 设置广播回调
Global.room.onJoinRoom = this.onJoinRoom.bind(this);
}

// 修改用户自定义状态
changeCustomPlayerStatus(customPlayerStatus) {
const changeCustomPlayerStatusPara = {
customPlayerStatus
};

Global.room.changeCustomPlayerStatus(changeCustomPlayerStatusPara, event => {
if (event.code !== Global.ErrCode.EC_OK) {
return this.toast("操作失败" + event.code);
}

return this.customPlayerStatus = customPlayerStatus;
});
}

onRoomUpdate() {
// 更新 按钮
const playerId = MGOBE.Player.id;
```

```
const player = Global.room.roomInfo.playerList.find(player => player.id === playerId);

if (player) {
this.customPlayerStatus = player.customPlayerStatus;

// 更新 按钮 文字
if (player.customPlayerStatus === 0) {
this.button.setText("准备");
} else {
this.button.setText("取消准备");
}
}

// 更新 MsgBox
let msg = "房间ID:\n" + Global.room.roomInfo.id + "\n玩家列表:\n";
Global.room.roomInfo.playerList.forEach((player, i) => {
msg += i + ":" + player.id + (player.customPlayerStatus === 1 ? " 已准备" : " 未准备") + "\n";
});

this.msgBox.setText(msg);

if (Global.room.isInRoom() && !Global.room.roomInfo.playerList.find(player => player.customPl
ayerStatus !== 1)) {
// 全部玩家准备好就跳转
setTimeout(() => this.open(Global.GameView), 1000);
}
}

// 加房广播
onJoinRoom(event) {
this.toast("新玩家加入");
}

// 退出房间
leaveRoom() {
Global.room.leaveRoom({}, event => {
if (event.code !== Global.ErrCode.EC_OK && event.code !== Global.ErrCode.EC_ROOM_PLAYE
R_NOT_IN_ROOM) {
```

```
return this.toast("操作失败" + event.code);
}
return this.open(Global.MainView);
});
}

onUpdate() { }

onDestroy() {
Global.room.onUpdate = null;
Global.room.onJoinRoom = null;
}
}

Global.RoomView = RoomView;
```

3. 编译代码后，在模拟器中进行加房操作进入 RoomView 房间页后，单击**退出房间**，即可跳转到 MainView 主页。

实现开始帧同步

在 RoomView 房间页中，全部玩家单击**准备**后，页面自动跳转到 GameView 游戏页。此时需要在 GameView 游戏页中检查房间帧同步状态，如果房间未开启帧同步，则调用开始帧同步接口。之后继续将玩家状态改为**未准备**，避免出现回到 RoomView 房间页中又自动跳转 GameView 游戏页的情况。

1. 修改 GameView.js 的 onInit 方法。示例代码如下：

```
onInit() {
const button = new Component.Button(20, 20, "结束帧同步");

const msgBox = new Component.MsgBox(20, 100, "");
this.msgBox = msgBox;

this.addComponent(button);
this.addComponent(msgBox);

if (Global.room.roomInfo.frameSyncState !== Global.ENUM.FrameSyncState.START) {
// 调用 startFrameSync 接口
this.startFrameSync();
```

```
}

// 修改玩家状态
this.changeCustomPlayerStatus(0);
}
```

2. 在 GameView 类中添加 startFrameSync 方法。示例代码如下：

```
// 开始帧同步
startFrameSync() {
const func = () => Global.room.startFrameSync({}, event => {
if (event.code !== Global.ErrCode.EC_OK) {
return this.dialog("操作失败，是否重试？", () => func());
}
});

func();
}
```

3. 在 GameView 类中添加 changeCustomPlayerStatus 方法。示例代码如下：

```
// 修改用户状态
changeCustomPlayerStatus(customPlayerStatus) {
const changeCustomPlayerStatusPara = { customPlayerStatus };

const func = () => Global.room.changeCustomPlayerStatus(changeCustomPlayerStatusPara,
event => {
if (event.code !== Global.ErrCode.EC_OK) {
this.dialog("操作失败，是否重试？", () => func());
}
});

func();
}
```

实现发送帧消息

玩家在游戏过程中发送的帧消息都是由一些指令组成，例如“跳跃”、“发射子弹”等操作。这里实现一个简单的指令，每个玩家发送一个随机数出去。在 GameView 类中添加一个 sendFrame 方法。示例代码如下：

```
// 玩家发送帧消息
sendFrame() {
const data = {
name: Global.name,
action: "random",
number: Math.ceil(Math.random() * 100),
}

Global.room.sendFrame({ data });
}
```

实现接收帧广播

开始帧同步后，SDK 会收到服务器推送的帧广播消息。您需要绑定 onFrame 回调函数，接收并处理每一帧消息。

1. 在 GameView 类的 onInit 中为 room 对象绑定帧消息广播回调，并实现 onFrame 方法。示例代码如下：

```
onInit() {
// ...
// 设置广播回调
Global.room.onRecvFrame = this.onRecvFrame.bind(this);
}
// ...
onRecvFrame(event) {
// 在这里处理帧广播
}
```

2. 在 onFrame 中添加代码，实现记录帧广播消息，并定时发送帧消息。示例代码如下：

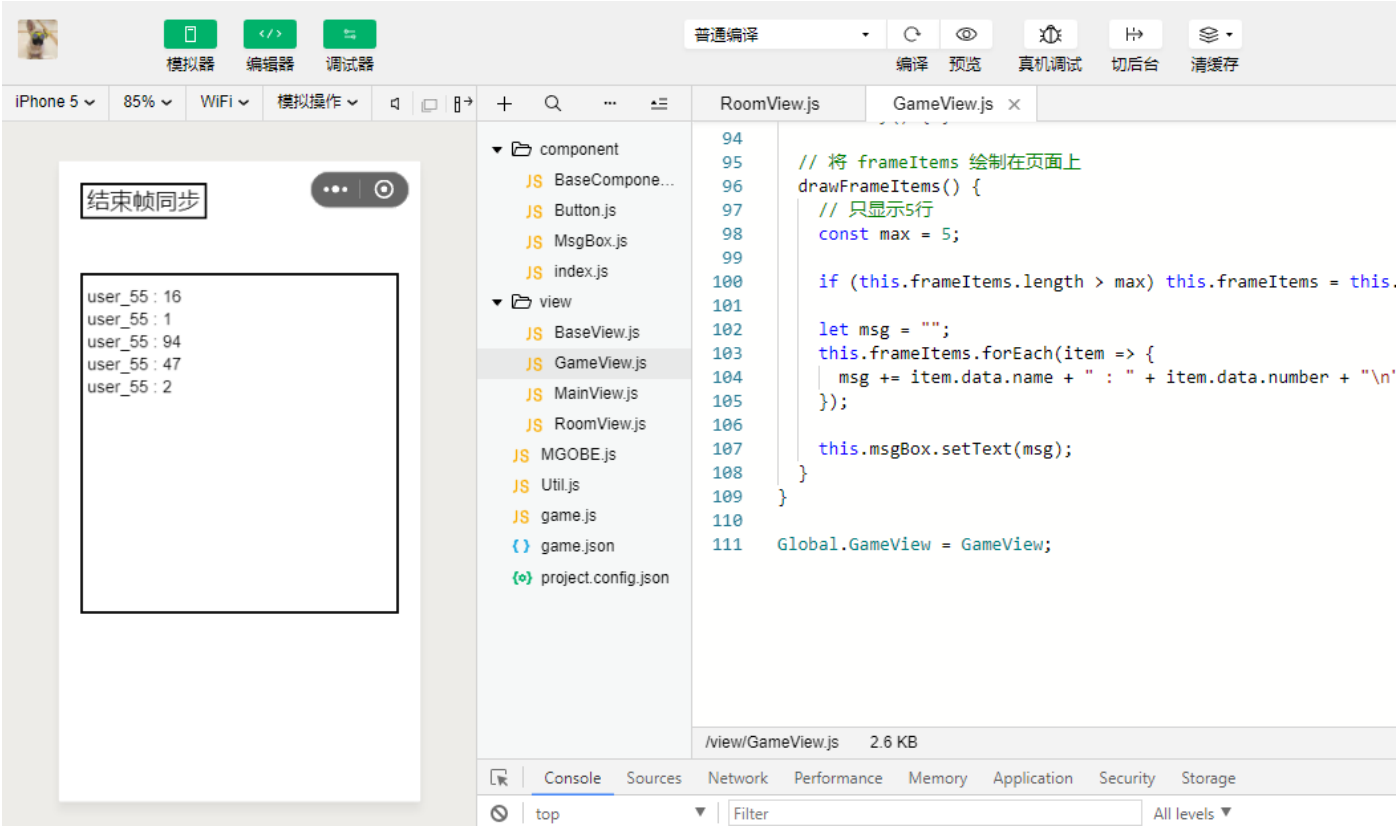
```
frameId = 0;
frameItems = [];
onRecvFrame(event) {
// 在这里处理帧广播
const frameId = event.data.frame.id;
// 每隔 15 帧发送一次帧消息
```

```
if (frameId > this.frameId + 15) {
this.frameId = frameId;
this.sendFrame();
}
// 记录帧广播消息
if (event.data.frame.items) {
this.frameItems = this.frameItems.concat(event.data.frame.items);
}
}
```

3. 在 onUpdate 中将帧消息渲染到页面上。示例代码如下：

```
onUpdate() {
// 渲染层不断更新页面
this.drawFrameItems();
}
// ...
drawFrameItems() {
// 只显示5行
const max = 5;
if (this.frameItems.length > max) this.frameItems = this.frameItems.slice(this.frameItems.le
ngth - max);
let msg = "";
this.frameItems.forEach(item => {
msg += item.data.name + " : " + item.data.number + "\n";
});
this.msgBox.setText(msg);
}
```

4. 编译项目，在模拟器上依次单击**快速加房**、**准备**后，进入 GameView 游戏页，页面将自动更新 MsgBox，并显示玩家消息。如下图所示：



实现停止帧同步

为 button 绑定点击事件监听，实现停止帧同步。

1. 在 onInit 中添加代码。示例代码如下：

```
onInit() {
const button = new Component.Button(20, 20, "结束帧同步");
button.onClick(() => this.stopFrameSync());

// ...
}
```

2. 使用 stopFrameSync 方法实现。示例代码如下：

```
// 停止帧同步
stopFrameSync() {

if (Global.room.frameSyncState === Global.ENUM.FrameSyncState.STOP) {
return;
}
```

```
const func = () => Global.room.stopFrameSync({}, event => {
if (event.code !== Global.ErrCode.EC_OK) {
this.dialog("操作失败，是否重试？", () => func());
}
});

func();
}
```

3. 帧同步停止后，需要跳转到 RoomView 房间页。检查房间帧同步状态有两种方法：
   - 监听 onStartFrameSync、onStopFrameSync 广播。
   - 在 room.onUpdate 中检查 room.roomInfo.frameStatus 的值。

4. 利用 room.onUpdate 回调同时检查帧同步状态和房间成员状态，如果房间停止帧同步并且房间成员状态全部为 0，就跳转到 RoomView 房间页。示例代码如下：

```
onRoomUpdate() {
if (Global.room.roomInfo.frameSyncState === Global.ENUM.FrameSyncState.STOP &&
!Global.room.roomInfo.playerList.find(player => player.customPlayerStatus === 1)) {
return this.open(Global.RoomView);
}
}
```

5. 在 onInit 方法、startFrameSync 的回调中绑定 onRoomUpdate。示例代码如下：

```
onInit() {
// ...
if (Global.room.roomInfo.frameSyncState !== Global.ENUM.FrameSyncState.START) {
// 调用 startFrameSync 接口
this.startFrameSync();
} else {
Global.room.onUpdate = this.onRoomUpdate.bind(this);
}
// ...
}
// ...
```

```
// 开始帧同步
startFrameSync() {
const func = () => Global.room.startFrameSync({}, event => {
if (event.code !== Global.ErrCode.EC_OK) {
return this.dialog("操作失败，是否重试？", () => func());
}
Global.room.onUpdate = this.onRoomUpdate.bind(this);
});
func();
}
```

6. 在 onDestroy 中清除 onUpdate、onFrame 回调。示例代码如下：

```
onDestroy() {
Global.room.onUpdate = null;
Global.room.onFrame = null;
}
```

7. 编译代码，在模拟器中进入 GameView 游戏页开始帧同步，然后单击**结束帧同步**，界面将跳转到 RoomView 房间页，并且玩家状态为**未准备**。

   GameView 游戏页最终示例代码如下：

```
// 帧同步游戏页面
import * as Util from "../Util.js";
import BaseView from "./BaseView.js";
import Component from "../component/index.js";
const Global = Util.Global;
export default class GameView extends BaseView {
msgBox;
constructor() {
super();
}
onInit() {
const button = new Component.Button(20, 20, "结束帧同步");
button.onClick(() => this.stopFrameSync());
const msgBox = new Component.MsgBox(20, 100, "");
this.msgBox = msgBox;
```

```
this.addComponent(button);
this.addComponent(msgBox);
if (Global.room.roomInfo.frameSyncState !== Global.ENUM.FrameSyncState.START) {
// 调用 startFrameSync 接口
this.startFrameSync();
} else {
Global.room.onUpdate = this.onRoomUpdate.bind(this);
}
// 修改玩家状态
this.changeCustomPlayerStatus(0);
// 设置广播回调
Global.room.onRecvFrame = this.onRecvFrame.bind(this);
}
// 开始帧同步
startFrameSync() {
const func = () => Global.room.startFrameSync({}, event => {
if (event.code !== Global.ErrCode.EC_OK) {
return this.dialog("操作失败，是否重试？", () => func());
}
Global.room.onUpdate = this.onRoomUpdate.bind(this);
});
func();
}
// 停止帧同步
stopFrameSync() {
if (Global.room.frameSyncState === Global.ENUM.FrameSyncState.STOP) {
return;
}
const func = () => Global.room.stopFrameSync({}, event => {
if (event.code !== Global.ErrCode.EC_OK) {
this.dialog("操作失败，是否重试？", () => func());
}
});
func();
}
onRoomUpdate() {
if (Global.room.roomInfo.frameSyncState === Global.ENUM.FrameSyncState.STOP &&
!Global.room.roomInfo.playerList.find(player => player.customPlayerStatus === 1)) {
```

```
return this.open(Global.RoomView);
}
}
// 修改用户状态
changeCustomPlayerStatus(customPlayerStatus) {
const changeCustomPlayerStatusPara = { customPlayerStatus };
const func = () => Global.room.changeCustomPlayerStatus(changeCustomPlayerStatusPara,
event => {
if (event.code !== Global.ErrCode.EC_OK) {
this.dialog("操作失败，是否重试？", () => func());
}
});
func();
}
// 玩家发送帧消息
sendFrame() {
const data = {
name: Global.name,
action: "random",
number: Math.ceil(Math.random() * 100),
}
Global.room.sendFrame({ data });
}
frameId = 0;
frameItems = [];
onRecvFrame(event) {
// 在这里处理帧广播
const frameId = event.data.frame.id;
// 每隔 15 帧发送一次帧消息
if (frameId > this.frameId + 15) {
this.frameId = frameId;
this.sendFrame();
}
// 记录帧广播消息
if (event.data.frame.items) {
this.frameItems = this.frameItems.concat(event.data.frame.items);
}
}
```

```
drawFrameItems() {
// 只显示5行
const max = 5;
if (this.frameItems.length > max) this.frameItems = this.frameItems.slice(this.frameItems.le
ngth - max);
let msg = "";
this.frameItems.forEach(item => {
msg += item.data.name + " : " + item.data.number + "\n";
});
this.msgBox.setText(msg);
}
onUpdate() {
// 渲染层不断更新页面
this.drawFrameItems();
}
onDestroy() {
Global.room.onUpdate = null;
Global.room.onFrame = null;
}
}
Global.GameView = GameView;
```

至此，整个 HelloWorld 示例结束，单击 这里 下载完整代码。

案例讲解
Cocos 引擎案例

最近更新时间：2022-03-29 14:11:37

> ⚠ 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。

案例说明

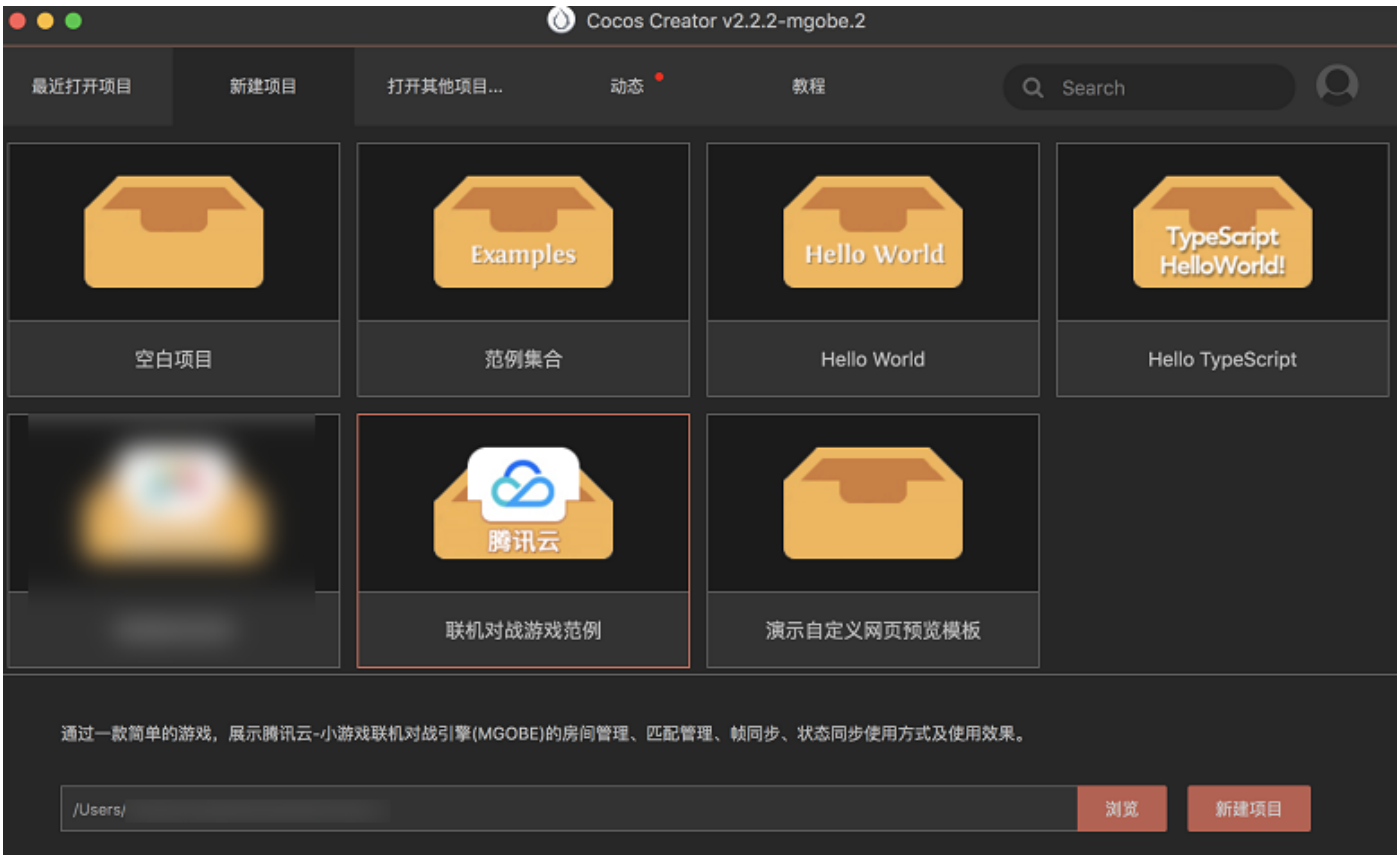
本文档通过一个完整的 DEMO，指导您如何通过 Cocos 引擎使用游戏联机对战引擎。

前提条件

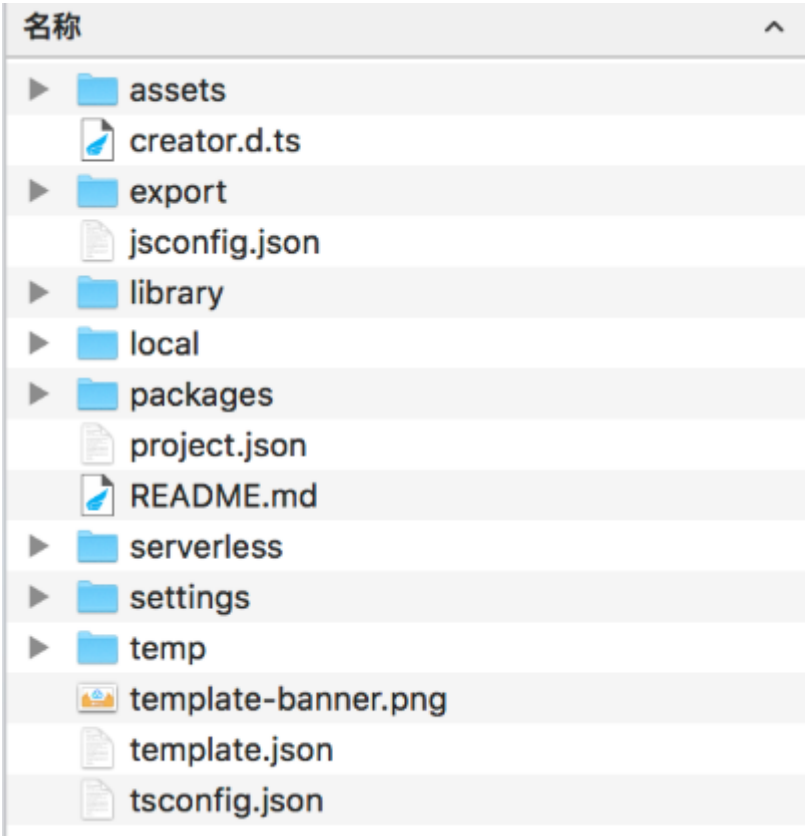
已 下载并安装 Cocos Creator（V2.1.0以上版本）。

操作步骤

1. 打开 Cocos Creator，新建一个“腾讯云联机对战游戏范例”工程。（您也可下载 游戏范例工程代码，自行导入工程）



2. 新建完成后，您将得到一个工程包，结构如下图所示：
   - **assets**：该目录下为所有客户端代码，游戏联机对战引擎的客户端 SDK 已导入该目录。所有的客户端逻辑，可在 assets\script 目录下查看。
   - **serverless**：该目录下为工程的服务端代码，包括游戏联机对战引擎的实时服务器框架。



3. 进入 Cocos Creator 主界面，单击顶部工具栏的“ ”按键，运行当前 DEMO。通过浏览器打开的界面如下图所示：

注意

- 由于当前 DEMO 演示联机功能，请您复制地址，在两个浏览器窗口运行当前 DEMO，以便您体验通信同步效果。
- 当前窗口展示的游戏 ID、游戏 Key、域名，为体验 DEMO 专用的游戏信息，当同时体验的用户较多时，可能产生通信延迟或卡顿。为了更好的体验效果，建议您替换为私有的游戏 ID、游戏 Key、域名。

4. 单击**体验游戏联机对战引擎**进入如下页面，并单击**初始化**，开始体验。

- **通过“创建房间”、“随机匹配”、“加入房间”三种方式，体验加入一个游戏房间功能**
  a. 单击**创建房间**，您可直接加入一个游戏房间。
  b. 单击**随机匹配**，MGOBE 将按照预设的匹配规则为您匹配其他玩家并加入游戏房间。您需要在另一个浏览器运行页面同步单击**随机匹配**，使两个浏览器被分配进同一个房间。
  c. 当您已经加入一个游戏房间后，可获取当前房间号，并在其他浏览器窗口通过指定的房间号加入游戏房间。

- **体验房间内消息发送功能**

选择“房间内发消息”，输入文字并发送，在两个浏览器窗口可以看到消息发送的同步效果。（只有房主可以切换房间内的消息通信方式）

- **体验实时服务器状态同步功能**

a. 选择“实时服务器状态同步”，当前页面有4个方向按键，可以控制玩家运动。此处运动逻辑在服务端进行计算，默认的运动逻辑为一次点击运动一格。您可在 serverless 框架下查看修改实时服务器代码。

b. 如需体验修改实时服务器逻辑代码，可将 DEMO 首页的游戏信息替换为您的私有游戏信息，并修改实时服务器代码，通过 Cocos Creator 服务面板一键发布，或在腾讯云控制台发布您的实时服务器代码。

- **体验帧同步功能**

选择“帧同步”，当前页面的操作面板有4个按键，开始帧同步、跑、停、停止帧同步；您需要先单击开始帧同步，方可操作跑/停控制玩家移动。

LayaAir 引擎案例

最近更新时间：2022-03-29 14:11:41

> ⚠ 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。

案例说明

本文主要介绍通过 LayaAir 引擎开发实践游戏联机对战的帧同步和状态同步。

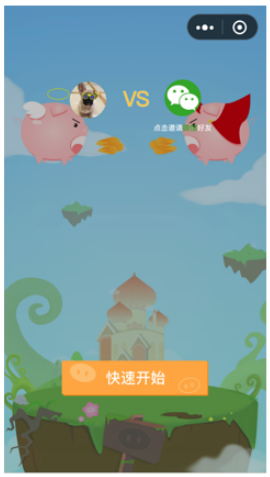
帧同步开发实践 -《猪猪对战》

开发实践帧同步的双人对战小游戏《猪猪对战》。

1. 下载 猪猪对战游戏源码，解压得源码文件。
2. 在游戏联机对战引擎控制台，开通游戏联机对战引擎服务，获取游戏 ID、游戏 KEY、域名等。详情请参见 开通服务。
3. 在游戏联机对战引擎控制台，新建匹配、创建匹配规则，获取匹配 code。详情请参见 匹配配置。
4. 使用 LayaAir 直接打开源码文件 Demo，即可快速构建猪猪对战游戏。详细的构建步骤见猪猪对战 视频教程。

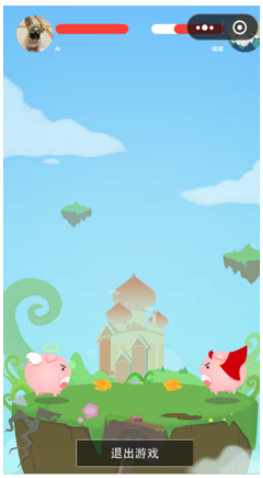
游戏部分页面展示

授权页



首页

房间页



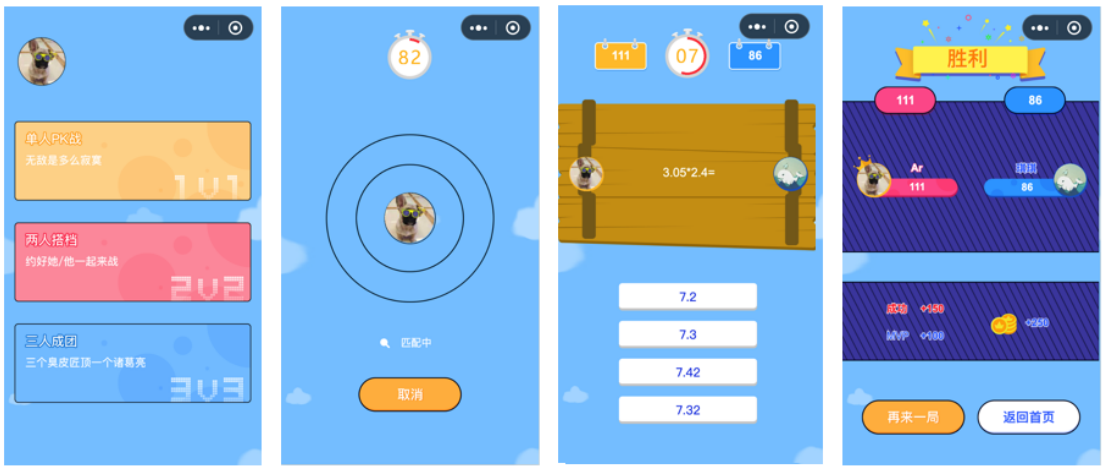
对战页

状态同步开发实践 -《题题对战》

开发实践状态同步的答题游戏《题题对战》。

1. 下载 题题对战游戏源码，解压得源码文件。
2. 在游戏联机对战引擎控制台，开通游戏联机对战引擎服务，获取游戏 ID、游戏 KEY、域名等。详情请参见 开通服务。
3. 在游戏联机对战引擎控制台，新建匹配、创建匹配规则，获取匹配 code。详情请参见 匹配配置。
4. 在游戏联机对战引擎控制台，创建 实时服务器，上传您的代码，发布服务，最终实现游戏的状态同步。
5. 用 LayaAir 直接打开代码文件中的客户端文件，即可快速构建题题对战游戏。详细的构建步骤见题题对战 视频教程。

游戏部分页面展示



首页　匹配页　答题页　结算页

对战答题小游戏案例

最近更新时间：2022-01-10 15:46:43

> **注意：**
> 因产品策略调整，游戏联机对战引擎后续将与云开发 CloudBase 整合为新产品形态，现将该产品保持维护状态，不再接收新用户使用申请，老用户仍可正常使用。

案例说明

本文主要介绍一款使用 MGOBE 开发实现的对战答题游戏。这款小游戏已在微信平台上线，作为科学防疫知识的科普工具用于公益用途。现将这款小游戏开源，方便更多开发者学习和使用 MGOBE 的能力。

- 您可 点击这里 下载游戏源码。

> **说明**
> 本文主要从玩法和交互角度进行简单介绍，具体的使用说明请您参考代码包中的 Readme 文档。

- 您可使用手机扫描下方小游戏码进行体验。

游戏玩法

- 双人对战答题。
- 个人测评小考场。

游戏交互流程

获取用户信息权限

1. 进入小游戏活动界面，游戏加载完成后，点击**开始**。
2. 在弹出获取微信个人信息权限提示页面，点击**允许**。（请通过个人信息授权并保存个人信息，否则您将为游客状态。）

进入游戏主界面

授权成功后，即可进入游戏主界面。主界面功能模板一共分为4部分显示：

- 个人信息：头像、积分、答题比拼正确率。
- 排行榜：好友排行、世界排行。
- 双人对战答题。

- 个人测评小考场。

双人对战答题

双人对战答题，玩家可以邀请好友或在线匹配，进行1v1的答题对战。每局6题，根据每题的正确作答时间进行计分，得分高者获胜并赢得积分。玩家作答时，双方的作答状态都会进行同步，营造良好的交互体验和游戏紧张感。

1. 点击**答题比拼**，即可进入答题比拼主界面，玩家可以邀请好友或在线匹配，进行1v1的答题对战。
   - **邀请好友**

     点击**邀请好友**，即可分享邀请链接给微信好友，邀请好友答题比拼。

- **在线匹配**

  点击**在线匹配**，即可匹配网友答题比拼。

2. 进入答题界面后，倒计时开始，玩家只需选择正确答案，即可获得积分，可实时观察双方答题进度以及积分情况，在点击答题选项的同时就会展现出正确答案以及获得的积分，答得越快越正确答题积分就越高。

个人测评小考场

个人测评小考场是针对某一类知识设计的一份10道题的考卷，玩家单人作答，答完后可获得所有题目的错误详解。测评小考场的题目包括单选和多选，相比于对战玩法强调的趣味性，测评小考场的难度更高，知识点更全面，适合希望快速学习知识的玩家。

1. 点击**防疫达人小考场**，即可进入个人测评小考场，一共10题。

2. 选择答案后，单击**提交**，即可获取结果，答题结束后可获得个人成绩单、题目详解。

背景音乐音效

为了使游戏表现形式丰富，这款小游戏使用 正版曲库直通车 添加音效和背景音乐。在游戏首页、对战答题页和个人测评小考场页，分别挑选三首自带版权的背景音乐，来呈现不同玩法页面的氛围。