Game Platform Developer Manual Document

Product Name: Game Platform

Service Name:

Developer Manual

Team: [ ]

Document Last Updated Date: (25-March-2025)

Confidentiality

Document ID & Revision History

Version	Date	Description of Changes	Author
v1.0.0	2025-1-22	Initial draft	

Detailed change log:

v1.0.0

- Initial version of the document.

Table of Contents

1. 快速开始

1.1 下载gpcli脚手架工具

gpcli脚手架是开发者用来创建和管理游戏的工具，请先下载脚手架工具，下载地址：

`http://www.ltonlinegame.com/web/gpcli/gpcli.exe`

1.2 创建开发者账号

1.2.1 获取用户ID和密码

首先请先联系平台管理员或者商务对接人获取您（公司或组织）的账号（UserID）和密码，使用 `GPCLI` 脚手架工具登录您的账号

1.2.2 用户登录

使用下面的命令进行认证登录开发者账号，登陆成功后以后使用脚手架工具都无需再次登录

```
./gpcli.exe login <UserID> <Password>
```

1.3 创建游戏

使用下面的命令创建一个游戏，GameID由开发者自己定义，一般建议游戏类型+游戏名称来命名，比如`老虎机游戏xxx`，建议命名为 `slot_xxx`

```
./gpcli.exe dev game create <GameID>
```

验证是否创建游戏成功，使用下面的命令可以看到当前开发者创建的所有游戏列表，如果看到创建的游戏的GameID表示创建成功

```
./gpcli.exe dev game list
```

执行命令后可以看到下面的打印信息，则表示 `slot_xxx` 游戏创建成功

```
+----------------------+-------------+-----------------+-------------+---------------------+
|        GAMEID        |  GAMETYPE   |     VERSION     |   STATUS    |      CREATEAT       |
+----------------------+-------------+-----------------+-------------+---------------------+
| slot_xxx             |             |                 | Unpublished | 2025-01-16 10:46:31 |
+----------------------+-------------+-----------------+-------------+---------------------+
```

1.4 前端对接游戏接口

前端需要对接后端的几个API接口，具体的接口和参数请参考 `API` 文档 游戏API接口

1.5 自定义游戏后端接口

开发者可以使用 `javascript` 实现自己定义的服务端接口，具体实现细节请参考 `API` 文档 4. 游戏后端开发 以及 2. 游戏API接口

1.6 准备游戏包

1.6.1 游戏包结构

游戏包是一个zip格式的压缩包，该压缩包必须按照以下结构组织，以确保正确解析和使用：

```
xxx.zip
└── xxx
    ├── client
        └── index.html
        └── other files...
    ├── server
        └── app.js
    └── app.json
```

ZIP 包的根目录名称必须与文件名一致，比如zip包的名字是 `xxx.zip`，那么zip包里面的根目录中必须且只能包含一个名为 `xxx` 的目录

1.6.2 server目录

如果游戏的后端由开发者自己使用 `javascript` 实现，则需要将 `javascript` 代码放入 `server` 目录，服务端的入口文件为 `app.js`

1.6.3 client目录

`client` 目录为游戏的前端资源文件， `index.hmtl` 是前端页面的的默认访问入口文件

app.json文件

`app.json` 是游戏的配置文件，上传游戏包的时候 `app.json` 文件是必须包含的文件， `client` 和 `server` 目录可以只有其中的任意一个

`app.json` 目前可以包含以下内容：

下面是一个游戏配置的示例

```
{
  "backend": {
    "type": "go"
  }
}
```

1.6.4 配置说明

配置	类型	说明
backend	object	游戏后端配置
backend.type	string	游戏后端的类型，目前支持的类型有 `js` 和 `go` 还有 `remote` 三种类型

关于游戏后端类型，请参考 4. 游戏后端开发章节

1.7 上传游戏包

使用下面的命令上传游戏包，其中GameID对应之前创建游戏时的游戏ID，`D:\xxx.zip` 是游戏包的文件路径

```
./gpcli.exe dev game upload <GameID> D:\xxx.zip
```

上传游戏包成功，脚手架会打印如下信息

```
Upload completed successfully!
GameID:    slot-xxx
Version:   2025.01.17.0001
URL:
http://www.ltonlinegame.com/sandbox/devsz/werben/2025.01.17.0001/client
```

配置	类型	说明
GameID	string	游戏ID
Version	string	当前上传游戏包的版本
URL	string	游戏包前端对应的的沙箱访问地址

1.8 发布游戏

使用下面的命令发布游戏，其中 `<GameID>` 对应之前创建游戏时的游戏ID，`<Version>` 是上传游戏包后打印的游戏版本号

```
./gpcli.exe dev game upload <GameID> <Version>
```

验证游戏是否发布成功

使用下面命令查看游戏列表

```
./gpcli.exe dev game list
```

可以看到下面的信息，对照 `GAMEID` 和 `VERSION`，如果 `STATUS` 的值为 `Published` 则表明成功发布游戏

```
+----------------------+-------------+-----------------+-------------+---------------------+
|       GAMEID         |   GAMETYPE  |     VERSION     |   STATUS    |      CREATEAT       |
+----------------------+-------------+-----------------+-------------+---------------------+
| slot_xxx             | gp_slot_xxx | 2025.01.16.0002 | Published   | 2025-01-16 10:46:31 |
+----------------------+-------------+-----------------+-------------+---------------------+
```

2. 游戏API接口

2.1 API概述

2.1.1 基础信息

所有API接口统一使用POST方法，采用JSON数据格式：

```
{{host}}/api.json
```

2.1.2 请求参数

参数名	类型	必填	说明
route	string	是	路由名称
data	object/array/string	是	请求数据

下面是post的数据示例：

```
{
    "route": "服务名.路由名",
    "data": {
        "参数1": "值1",
        "参数2": "值2"
    }
}
```

2.1.3 响应参数

所有API响应都遵循以下结构：

参数名	类型	必填	说明
code	int	是	错误码，0表示成功，其他表示失败
data	object/array/string	是	响应数据

响应示例：

```
{
    "code": 0, // 0表示成功
    "data": {
        // 具体接口的响应数据
    }
}
```

2.2 登录接口

提供用户认证功能。

2.2.1 请求路由

路由： `AuthService.Login`

2.2.2 请求参数

```
{
    "route": "AuthService.Login",
    "data": {
        "identifier": "xxxxxx",
        "credential": ""
    }
}
```

请求参数说明

参数名	类型	必填	说明
identifier	string	是	用户ID
credential	string	否	用户凭证

2.2.3 响应参数

参数名	类型	说明
code	int	响应码，0表示成功
data.accessToken	string	访问令牌，用于后续接口调用的身份验证

2.3 游戏初始化接口

用于创建新的游戏会话并进行初始化。

2.3.1 请求路由

路由： `GameService.InitGame`

2.3.2 请求参数

请求示例

```
{
    "route": "GameService.InitGame",
    "data": {
        "gameId": "slot_zhaocaijinbao",
        "accessToken": "{{accessToken}}"
    }
}
```

参数名	类型	必填	说明
gameId	string	是	游戏ID，如`slot_zhaocaijinbao`
accessToken	string	是	用户访问令牌

2.3.3 响应参数

响应示例


```
{
    "code": 0,
    "data": {
        "config": {
            "stake": {
                "bets": [800, 1800, 3800, 6800, 8800],
                "default": {
                    "bet": 8800,
                    "line": 1,
                    "multiple": 1
                },
                "lines": [1],
                "multiples": [1,2,3,6,10]
            }
        },
        "game": {
            "bets": [],
            "currentFreeTimes": 0,
            "currentWinAmount": 0,
            "initWindows": [
                {"list": [{"symbolId": 6},{"symbolId": 15},{"symbolId": 15}]},
                {"list": [{"symbolId": 12},{"symbolId": 13},{"symbolId": 14}]},
                {"list": [{"symbolId": 16},{"symbolId": 10},{"symbolId": 13}]},
                {"list": [{"symbolId": 14},{"symbolId": 13},{"symbolId": 15}]},
                {"list": [{"symbolId": 13},{"symbolId": 13},{"symbolId": 11}]}
            ],
            "jackpot": null,
            "jackpots": [
                {"amount": 7500035, "name": "super"},
                {"amount": 1000070,  "name": "major"}
            ],
            "totalFreeTimes": 0,
            "transactionId": ""
        },
        "playInfo": {
            "gameId": "slot_zhaocaijinbao",
            "id": "slot_zhaocaijinbao-xxxxxx-CgUivjpXVi6BQNeO",
            "jackpotId": "JP-nw2HFBE62H96CC4G",
            "playToken": "InO2hMIuwqMsUWVMzbcDEZA1kouQHKv7HakPXOa87Aw=.qwh1n5-
e5vKu6XAvERCx7OixcuNgm_-h_OQv3LnrtmI=",
            "version": "v1.0.0"
        },
        "walletAmount": 139806821
    }
}
```

基础响应参数
所有游戏类型都会返回以下参数：

参数名	类型	说明
code	int	响应码，0表示成功
data.playInfo	object	游戏会话信息
data.playInfo.gameId	string	游戏ID
data.playInfo.version	string	游戏版本号
data.playInfo.playToken	string	游戏会话令牌
data.playInfo.id	string	游戏会话唯一标识
data.walletAmount	int	钱包最新余额
data.config	object	游戏配置信息

游戏业务响应参数

以下是以招财进宝游戏为例的游戏业务响应参数：

投注配置

参数名	类型	说明
config.stake	object	投注配置
config.stake.debits	array/int	投注额列表
config.stake.multiples	array/int	倍数列表
config.stake.lines	array/int	线数列表
config.stake.default.bet	int	默认投注额
config.stake.default.multiple	int	默认倍数
config.stake.default.line	int	默认线数

Jackpot 奖池配置

参数名	类型	说明
game.jackpots	array	jackpots奖池
game.jackpots[].amount	int	奖池金额
game.jackpots[].name	string	奖池名称

游戏状态数据

参数名	类型	说明
game.transactionId	string	交易ID
game.totalFreeTimes	int	总免费游戏次数
game.currentFreeTimes	int	当前剩余免费次数
game.currentWinAmount	int	当前累计赢取金额
game.jackpot.isGameOver	bool	jackpot是否结束
game.jackpot.rounds	array	jackpot每轮数据

游戏窗口数据

参数名	类型	说明
game.initWindows	array	初始窗口数据
game.initWindows[].list	array	窗口列数据
game.initWindows[].list[].symbolId	int	图标ID

2.4 下注接口

用于玩家下注并开始游戏。该接口支持基础游戏和免费游戏模式的下注操作。

2.4.1 请求路由

路由： `GameService.PlaceBets`

2.4.2 请求参数


```
{
    "route": "GameService.PlaceBets",
    "data": {
        "transactionId": "{{UUID}}",
        "playInfo": {
            "gameId": "slot_zhaocaijinbao",
            "id": "slot_zhaocaijinbao-xxxxxx-CgUivjpXVi6BQNeO",
            "jackpotId": "JP-nw2HFBE62H96CC4G",
            "playToken": "In02hMIuwqMsUWVMzbcDEZA1kouQHKv7HakPX0a87Aw=.qwh1n5-
e5vKu6XAvERCx7OixcuNgm_-h_OQv3LnrtmI=",
            "version": "v1.0.0"
        },
        "bets": [
            {
                "index": 1,
                "amount": 8800,
                "options": {
                    "bet": "8800",
```

```
                    "multiple": "1"
                }
            }
        ],
        "extraData":{
            "flag": 1
        }
    }
}
```

请求参数说明

基础请求参数

参数名	类型	必填	说明
transactionId	string	是	交易唯一标识，用于防重复提交
playInfo	object	是	游戏会话信息，来自初始化接口
playInfo.gameId	string	是	游戏ID标识
playInfo.version	string	是	游戏版本号
playInfo.playToken	string	是	游戏会话令牌
playInfo.id	string	是	游戏会话唯一标识
bets	array	是	投注信息列表
extraData	object	是	额外游戏参数

投注参数说明

参数名	类型	必填	说明
bets[].index	int	是	投注序号，从1开始
bets[].amount	int	是	总投注金额（基础金额 * 倍数）
bets[].options	object	是	投注选项
bets[].options.bet	string	是	基础投注金额
bets[].options.multiple	string	是	投注倍数

额外的参数

参数名	类型	必填	说明
extraData.flag	int	是	游戏标识,1:基础游戏,2:免费游戏

2.4.3 响应参数

响应示例


```
{
    "code": 0,
    "data": {
        "bets": [],
        "extraData": {
            "tourney": {
                "data": {
                    "message": "the tournament has ended or been closed and
cannot be found"
                },
                "error": "",
                "timestamp": "2025-01-16T10:16:10.994111919Z"
            }
        },
        "game": {
            "isCompleted": true,
            "jackpot": null,
            "jackpots": [
                {"amount": 1000140, "name": "major"}
            ],
            "lineWins": [
                {
                    "count": 3,
                    "direction": 1,
                    "multiple": 2,
                    "symbolId": 14,
                    "totalWinCash": 10000,
                    "winCash": 5000,
                    "winSymbols": [
                        { "column": 0, "row": 1, "symbolId": 14 },
                        { "column": 0, "row": 2, "symbolId": 14 },
                        { "column": 1, "row": 1, "symbolId": 14 },
                        { "column": 2, "row": 0, "symbolId": 14 }
                    ]
                }
            ],
            "scatterWin": {
                "freeTimes": 8,
                "symbolId": 17,
                "winSymbols": [
                    { "column": 0, "row": 2 },
                    { "column": 2, "row": 1 },
                    { "column": 4, "row": 2 }
                ]
            },,
            "stopWindows": [
                {"list": [{"symbolId": 6},{"symbolId": 15},{"symbolId": 15}]},
                {"list": [{"symbolId": 12},{"symbolId": 13},{"symbolId": 14}]},
                {"list": [{"symbolId": 16},{"symbolId": 10},{"symbolId": 13}]},
                {"list": [{"symbolId": 14},{"symbolId": 13},{"symbolId": 15}]},
                {"list": [{"symbolId": 13},{"symbolId": 13},{"symbolId": 11}]}
            ],
```

```
            "totalFreeTimes": 0,
            "totalWinCash": 0
        },
        "status": 1,
        "transactionId": "62aae26f-4450-47c3-966c-ad754340acb4",
        "walletAmount": 139798021,
        "walletChangeCredits": 8800
    }
}
```


基础响应参数

参数名	类型	说明
transactionId	string	交易ID，与请求一致
walletAmount	int	钱包最新余额
walletChangeCredits	int	本次交易积分变动额
status	int	参考下面的状态码描述

游戏状态码描述：

状态码	描述
1	下注和转轮成功
2	重复下注，未转轮
3	下注失败，未转轮
4	下注成功，转轮失败
5	转轮成功，积分入账失败
6	Bonus选择失败
7	Bonus选择成功，入账失败
8	Jackpot选择失败
9	Jackpot选择成功，入账失败

游戏业务响应参数

参数名	类型	说明
game.initWindows	array	初始窗口数据
game.stopWindows	array	停止后窗口数据
game.totalWinCash	int	总中奖金额
game.totalFreeTimes	int	获得的免费游戏次数

奖池数据

参数名	类型	说明
game.jackpots	array	jackpots奖池
game.jackpots[].amount	int	奖池金额
game.jackpots[].name	string	奖池名称

中奖线数据

参数名	类型	说明
game.lineWins	array	线条中奖信息列表
game.lineWins[].count	int	中奖符号数量
game.lineWins[].direction	int	中奖方向（1=从左到右）
game.lineWins[].multiple	int	中奖倍数
game.lineWins[].symbolId	int	中奖符号ID
game.lineWins[].totalWinCash	int	该线总中奖金额
game.lineWins[].winCash	int	单线中奖金额
game.lineWins[].winSymbols	array	中奖符号位置列表

Scatter中奖数据

参数名	类型	说明
game.scatterWin	object	scatter中奖信息
game.scatterWin.count	int	scatter中奖数量
game.scatterWin.multiple	int	scatter中奖倍数
game.scatterWin.totalWinCash	int	scatter总中奖金额
game.scatterWin.winSymbols	array	scatter中奖位置列表

2.5 错误码说明

错误码	说明
0	成功
1001	参数错误
1002	余额不足
1003	重复提交
2001	游戏会话无效
2002	游戏状态错误

3. GPCLI脚手架工具

3.1 配置脚手架

3.1.1 签名和验签

APIKey是一种用于身份验证和授权的密钥，通常由服务端生成并分发给客户端。客户端在进行API请求时，使用APIKey参与签名生成，以确保请求的完整性和安全性。APIKey本身不需要传递给服务端，只需在客户端参与签名计算即可。

1. **客户端**：使用APIKey和其他请求参数生成签名。
2. **服务端**：使用相同的APIKey和接收到的请求参数重新计算签名，并与客户端传递的签名进行比对，以验证请求的合法性。

3.1.2 获取APIKey

首先请先联系平台管理员或者商务对接人获取您（公司或组织）的账号和密码，使用 `GPCLI` 脚手架工具登录您的账号

```
./gpcli.exe login <UserID> <Password>
```

如果账号和密码都正确，会显示您的APIKey：

```
.\gpcli.exe login User-XXX Pwd-XXX
Login successful!
Organization APIKey: abcdx8baztbtOjq2hk52v69Odmdgz123
Please keep your organization's APIKey safe and do not disclose it to third
parties.
```

3.1.3 多配置管理

使用以下命令切换到指定的配置文件：

```
./gpcli.exe config use <ConfigName>
```

例如，如果您有三个角色配置文件：admin（管理员）、developer（开发者）和 channel（渠道商），可以通过以下命令切换到开发者的配置文件：

```
./gpcli.exe config use xxx-dev
```

临时配置

若仅需临时使用特定配置，可在执行子命令时通过 `-c` 参数指定：

```
./gpcli.exe <子命令> -c admin
```

通过 `-c` 指定的配置仅对当前指令生效，不会修改全局配置，也不会影响 `./gpcli.exe use <ConfigName>` 设置的默认配置。

3.1.4 其他配置命令

```
./gpcli.exe config <子命令>
```

子命令	功能描述	英文说明
use	切换到指定的配置文件	Switch to the specified config
info	查看当前正在使用的配置	Display the currently active config
list	查看所有可用配置	List all available configs
remove	删除某个配置	Remove a specific config

3.2 使用脚手架命令

3.2.1 执行命令方式

开发者命令必须是开发者账号才能使用，想要查看命令帮助，可以输入 `./gpcli.exe dev -h` ，执行命令方式如下，

```
./gpcli.exe dev <子命令>
```

game 子命令

子命令	功能描述	英文说明
game	开发者游戏管理	Manage games for the developer

3.2.2 开发者子命令列表

```
./gpcli.exe dev game <子命令>
```

子命令	功能描述	英文说明
create	创建一个新游戏	Create a new game
download	下载游戏包到本地文件系统	Download game package to the local file system
list	列出当前用户的所有游戏	List all games of the current user
publish	将游戏发布到线上	Publish a game online
remove	删除一个游戏版本	Remove a game version
upload	上传本地的 zip 文件到游戏	Upload a local zip file to the game
update	更新游戏某个版本的zip包	update a local zip file to the game
versions	列出指定游戏的所有版本	List all versions of the specified game

3.2.3 其他通用命令

```
./gpcli.exe <命令>
```

命令	功能描述	英文说明
version	打印脚手架版本信息	Print the current CLI version information
update	检查并更新脚手架	Check for updates and upgrade the CLI

3.4 命令描述

3.4.1 创建游戏

在终端中执行下面的命令创建游戏

```
./gpcli.exe dev game create <GameID>
```

其中GameID由开发者自定义，如slot_alice，需要注意的是平台会保证`GameID`的唯一性，如果创建游戏时提示`GameID`已经存在，请替换其他的名字，创建游戏成功后会终端会输出如下信息：

```
 .\gpcli.exe dev game create xxx

 [INFO] Create Game successfully!
GameID  : xxx
```

3.4.2 获取游戏列表

在终端中执行下面的查看开发者的游戏列表

```
./gpcli.exe dev game list
```

终端会输出如下信息：

```
+----------------------+--------------------------+----------------+------------+---------------------+
|        GAMEID        |         GAMETYPE         |     VERSION    |   STATUS   |      CREATEAT       |
+----------------------+--------------------------+----------------+------------+---------------------+
| slot_zhaocaijinbao   | gp_slot_zhaocaijinbao    | 2025.01.10.0004 | Published  | 2024-12-16 01:19:26 |
| slot_mayan           | gp_slot_mayan            | 2025.01.16.0001 | Published  | 2025-01-02 03:08:18 |
| slot_jackpotmultiplier | gp_slot_jackpotmultiplier | 2025.02.08.0001 | Published  | 2025-02-08 06:01:49 |
| xxx                  |                          |                | Unpublished | 2025-02-11 01:48:51 |
+----------------------+--------------------------+----------------+------------+---------------------+
```

其中GAMETYPE列表示游戏类型，`GAMETYPE`为空表示游戏的服务端是由javascript实现，请参考：1.6 准备游戏包中上包的说明

3.4.3 上传游戏包

使用下面的命令上传游戏包，其中GameID对应之前创建游戏时的游戏ID，`D:\xxx.zip`是游戏包的文件路径

```
./gpcli.exe dev game upload <GameID> D:\xxx.zip
```

上传游戏包成功，脚手架会打印如下信息

```
Upload completed successfully!
GameID:    werben
Version:   2025.01.17.0001
URL:
http://www.ltonlinegame.com/sandbox/devsz/werben/2025.01.17.0001/client
```

配置	类型	说明
GameID	string	游戏ID
Version	string	当前上传游戏包的版本
URL	string	游戏包前端对应的的沙箱访问地址

3.4.4 发布游戏

使用下面的命令发布游戏，其中 `<GameID>` 对应之前创建游戏时的游戏ID，`<Version>` 是上传游戏包后打印的游戏版本号

```
./gpcli.exe dev game upload <GameID> <Version>
```

验证游戏是否发布成功

使用下面命令查看游戏列表

```
./gpcli.exe dev game list
```

可以看到下面的信息，对照 `GAMEID` 和 `VERSION`，如果 `STATUS` 的值为 `Published` 则表明成功发布游戏

```
+----------------------+-------------+-----------------+-------------+---------------------+
|       GAMEID         |   GAMETYPE  |     VERSION     |   STATUS    |      CREATEAT       |
+----------------------+-------------+-----------------+-------------+---------------------+
| slot_xxx             | gp_slot_xxx | 2025.01.16.0002 | Published   | 2025-01-16 10:46:31 |
+----------------------+-------------+-----------------+-------------+---------------------+
```

3.4.5 更新游戏包

使用下面的命令可以更新已经存在的游戏包，其中 `<GameID>` 对应之前创建游戏时的游戏ID，`<Version>` 是已经存在游戏包的版本号，最后需要带上新的游戏包的路径。

```
./gpcli.exe dev game upload <GameID> <Version> D:\xxx.zip
```

注意：如果更新的游戏版本包已经发布了，如果需要同步新的游戏包到线上，需要重新执行一次发布命令，发布该游戏版本

3.4.6 删除游戏版本

在终端中执行下面的命令删除游戏版本，其中 `<GameID>` 对应之前创建游戏时的游戏ID，`<Version>` 是已经存在游戏包的版本号

```
./gpcli.exe dev game remove <GameID> <Version>
```

注意：如果删除游戏的版本已经发布了，删除这个游戏版本，线上的版本不会受影响。也就是说版本虽然删除了，但是还在线上运行。

3.4.7 获取某个游戏的所有版本号

在终端中执行下面的命令获取所有版本号

```
./gpcli.exe dev game versions <GameID>
```

会输出类似如下的信息：

```
+-----------------+----------------------+---------------+---------------------+
|    VERSIONID    |       GAMETYPE       |    STATUS     |     CREATED AT      |
+-----------------+----------------------+---------------+---------------------+
| 2025.01.10.0004 | gp_slot_zhaocaijinbao | Client        | 2025-01-10 06:11:57 |
| 2025.01.09.0003 | gp_slot_zhaocaijinbao | Client        | 2025-01-10 05:29:05 |
| 2025.01.09.0002 | gp_slot_zhaocaijinbao | Client        | 2025-01-10 05:06:41 |
| 2025.01.03.0007 |                       | Client+Server | 2025-01-03 06:03:26 |
| 2025.01.02.0006 |                       | Client+Server | 2025-01-03 05:55:43 |
+-----------------+----------------------+---------------+---------------------+
```

STATUS 表示该版本包中包含的内容，`Client` 表示只有前端的代码，`Client+Server` 表示游戏包中同时包含Client和Server（javascript）的代码。

命令默认只显示最新的10个版本号，如果想查看更老的版本号可以使用 -l 和 -o 来

- -l 或 --limit 限制显示多少条数据
- -o 或 --offset 限制显示的数据的偏移量

比如我想查看 最新的10-20条数据，则可以使用命令

```
./gpcli.exe dev game versions <GameID> --offset 10 --limit 10
```

4. 游戏后端开发

4.1 自定义后端

目前服务器游戏后端有三种方式，golang实现的是平台内置的后端，对接内置的后端只需要前端对接平台的API接口即可。开发者如果需要自定义自己的后端服务，目前支持的有两种方式：

1. 用 `javascript` 实现后端代码（`js`），并将代码和游戏包一起打包上传到平台，平台会自动识别并部署
2. 自定义远程后端服务(`remote`)，并将服务地址和游戏包一起上传到平台，平台会将游戏接口发送到远程服务，由远程服务处理游戏逻辑。

4.2 golang 后端开发

golang后端是内置在平台上的，开发者只需要前端对接平台的API接口。

4.3 javascript 后端开发

4.3.1 游戏包结构

自定义js游戏后端服务，游戏包是一个zip格式的压缩包，该压缩包必须按照以下结构组织，以确保正确解析和使用：

```
xxx.zip
└── xxx
    ├── client
        └── index.html
        └── other files...
    ├── server
        └── app.js
    └── app.json
```

ZIP 包的根目录名称必须与文件名一致，比如zip包的名字是 `xxx.zip`，那么zip包里面的根目录中必须且只能包含一个名为 `xxx` 的目录

4.3.2 server目录

如果游戏的后端由开发者自己使用 `javascript` 实现，则需要将 `javascript` 代码放入 `server` 目录，服务端的入口文件为 `app.js`

4.3.3 js服务端示例

app.js 示例

```
var slot = require("slot")

function handler(context, request) {
    console.log("handler info", "request", request)
    var payload = request.payload
    switch (request.route) {
        case "Init":
            console.log("handler Init", "payload", payload)
            return {
                playInfo: {
                    gameId: payload.gameId,
                    version: "1.0.0",
                },
                walletAmount: 100.0 // 钱包余额
            }
        case "PlaceBets":
            console.log("handler PlaceBets", "payload", payload)
            var sg = new slot.SlotGame(slot.Config)
            sg.spin({
                betInfo: {
                    line: payload.Bets[0].Options.line || 0,
                    bet: payload.Bets[0].Amount,
                    multiple: payload.Bets[0].Options.multiple
                }
            })
            return { transactionId: payload.TransactionId }
        case "PlayerTurn":
            console.log("handler PlayerTurn", "payload", payload)
            return {}
    }
}
```

4.4 远程后端开发

4.4.1 游戏包结构

`client` 目录是自定义 Remote 服务端游戏包中的前端资源文件夹，包含了游戏的前端代码和静态资源文件。该目录的结构和内容必须按照规范组织，以确保游戏能够正确加载和运行。

目录结构
自定义 Remote 服务端游戏的包是一个 ZIP 格式的压缩包，其结构如下：

```
my_game.zip
└── my_game
    ├── client
        └── index.html
        └── styles.css
        └── script.js
        └── images
            └── background.png
    └── app.json
```

关键文件说明

1. `index.html`
   - 这是前端页面的默认入口文件，游戏加载时会首先访问该文件。
   - 确保 `index.html` 文件中正确引用了其他前端资源（如 CSS、JavaScript 文件）。
2. **其他前端资源文件**
   - 包括但不限于：
     - CSS 文件：用于定义页面样式。
     - JavaScript 文件：用于实现页面交互逻辑。
     - 图片、字体等静态资源：用于丰富页面内容。
3. `app.json`
   - 这是游戏的配置文件，定义了游戏的基本信息和后端服务的配置。
   - 以下是 `app.json` 的配置示例：

```
{
  "backend": {
    "type": "remote",             # 后端服务类型，此处为 Remote 服务端
    "endpoint": "https://www.example.com:10888/endpoint"  # 后端服务的访问地址
  }
}
```

`app.json` 配置参数说明

参数	说明
`backend`	后端服务配置，包含后端服务类型和访问地址。
`type`	后端服务类型，此处固定为 `remote`，表示使用 Remote 服务端。
`endpoint`	后端服务的访问地址，游戏前端会通过该地址与后端服务进行通信。需确保地址可访问且协议正确。

4.4.2 后端实现接口

假设开发者配置的远程地址（`endpoint`）为 `https://www.example.com:10888/endpoint`，请求地址将按以下格式发起请求：

```
https://www.example.com:10888/endpoint?action=<Action>&player=<PlayerId>
```

其中：

- `action`：表示接口类型，目前支持的接口类型如下表。
- `player`：表示玩家 ID，用于标识当前操作的玩家。

要实现的Action接口类型

接口类型	说明
`InitGame`	初始化游戏
`PlaceBets`	玩家下注
`PlayerTurn`	玩家回合操作

4.4.3 签名和验签

1. **客户端**：使用APIKey和其他请求参数生成签名。
2. **服务端**：使用相同的APIKey和接收到的请求参数重新计算签名，并与客户端传递的签名进行比对，以验证请求的合法性。

获取APIKey

首先请先联系平台管理员或者商务对接人获取您（公司或组织）的账号和密码，使用 `GPCLI` 脚手架工具登录您的账号，参考3.1.2 获取APIKey

签名生成流程

在生成签名之前，客户端需要准备以下信息：

- **APIKey**：登录账号后分发的密钥，用于参与签名生成。
- **UserID**：客户端的唯一标识符，就是登录账号时使用的UserID。
- **Timestamp**：当前时间戳，单位为秒，用于防止重放攻击。
- **Nonce**：随机生成的字符串，确保每次请求的签名唯一。

生成签名

- **拼接字符串**：
  将 ClientID、Timestamp、Nonce 和 APIKey 按照固定顺序拼接成一个字符串。
  示例：`message = clientID + timestamp + nonce + apiKey`;
- **计算签名**
  使用SHA-256算法对拼接后的字符串进行哈希计算，生成签名。
  示例：signature = crypto.SHA256(message).toString(crypto.enc.Hex);

设置请求头

将生成的签名及相关参数添加到请求头中：

- X-Timestamp：时间戳。
- X-Client-ID：客户端ID。
- X-Nonce：随机数。
- X-Signature：生成的签名。

4.4.5 接口数据结构

- **InitGame**初始化游戏接口

初始化游戏接口对应的数据结构如下：

```
message InitRequest {
    option (pb.h_route) = "GameService.InitGame";
    string gameId = 1 [ json_name = "gameId" ];
    string accessToken = 2 [ json_name = "accessToken" ];
}
message InitResponse {
    GamePlayInfo playInfo = 1 [ json_name = "playInfo" ];
    google.protobuf.Struct config = 2 [ json_name = "config" ]; // 游戏规则配置
    google.protobuf.Struct game = 3 [ json_name = "game" ]; // 游戏数据
    int64 walletAmount = 4 [ json_name = "walletAmount" ];  // 钱包余额
}
```

请求参数对应上面的`InitRequest`，玩家ID需要从请求的url里面读取

响应参数对应上面的`InitResponse`结构，其中的`config`和`game`由开发者自定义，保证它们是一个json对象即可，用于描述游戏规则和游戏数据。`playInfo`不需要开发者返回，由平台管理注入。

- **PlaceBets**下注接口

下注接口对应的数据结构如下：

```
message GameBetInfo {
    int32 index = 1 [ json_name = "index" ]; // 同时多处下注场景的位置索引
    int64 amount = 2 [ json_name = "amount", jstype = JS_NUMBER ]; // 下注额
    map<string, string> options = 3 [ json_name = "options" ]; // 赌注,倍数,彩线数
量等
}

// 玩家下注
message PlaceBetsRequest {
    option (pb.h_route) = "GameService.PlaceBets";
    GamePlayInfo playInfo = 1 [ json_name = "playInfo" ];
    string transactionId = 2 [ json_name = "transactionId" ];
    repeated GameBetInfo bets = 3 [ json_name = "bets" ];
    google.protobuf.Struct extraData = 4 [ json_name = "extraData" ];
}

// 平台需要的结算信息
message GameWinInfo {
    int64 betAmount = 1 [ json_name = "betAmount" ];
    int64 gameWin = 2 [ json_name = "gameWin" ];
    int64 jackpotWin = 3 [ json_name = "jackpotWin" ];
}

message PlaceBetsResponse {
    string transactionId = 1 [ json_name = "transactionId" ];
    int32 status = 2 [ json_name = "status" ];
    google.protobuf.Struct game = 3 [ json_name = "game" ];
    repeated GameBetInfo bets = 4 [ json_name = "bets" ]; // 返回最新的下注信息，支持
多端同时游戏
    optional int64 walletAmount = 5 [ json_name = "walletAmount" ];
    optional int64 walletChangeCredits = 6 [ json_name = "walletChangeCredits"
];
    map<string, google.protobuf.Struct> extraData = 7 [ json_name = "extraData"
];
}
```

请求参数对应上面的`PlaceBetsRequest`，玩家ID需要从请求的url里面读取

响应参数对应上面的`PlaceBetsResponse`，**注意**: 平台只关心最终的结算结果，不处理中间状态（如free game, bonus game），平台会判断`PlaceBetsResponse.extraData`里面是否有`gameWinInfo`字段，如果有，则表示游戏已经结束，平台会结算游戏结果，将最终的奖金发放到玩家的钱包中。如果是中间状态的游戏（如Free Game），开发者的远程服务端不需要往`extraData`里面写入`gameWinInfo`。

4.4.6 后端服务器示例

```
package main

import (
    "GamePlatform/pb"
    "crypto/sha256"
    "encoding/hex"
    "encoding/json"
    "fmt"
    "net/http"
    "strings"

    "github.com/gin-gonic/gin"
)

// APIKey 定义用于签名验证的密钥
const (
    APIKey = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
)

// PlayInfo 玩家信息
type PlayInfo struct {
    GameId    string  `json:"gameId"`    // 游戏ID
    Version   string  `json:"version"`   // 游戏版本
    PlayToken string  `json:"playToken"` // 玩家令牌
    Id        string  `json:"id"`
    JackpotId string  `json:"jackpotId"` // 奖池ID
    TourneyId *string `json:"tourneyId"` // 赛事ID
}

// InitRequest 初始化请求
type InitRequest struct {
    GameId    string `json:"gameId"`    // 游戏ID
    ChannelId string `json:"channelId"` // 渠道ID
    PlayerId  string `json:"playerId"`  // 玩家ID
}

// InitResponse 初始化响应
type InitResponse struct {
    PlayInfo PlayInfo       `json:"playInfo"`
    Config   map[string]any `json:"config"`
    Game     map[string]any `json:"game"`
}

// Bet 玩家下注
type Bet struct {
    Index   string            `json:"index"`
    Amount  int               `json:"amount"`
    Options map[string]string `json:"options"`
}

// PlaceBetRequest 下注请求
type PlaceBetRequest struct {
    PlayInfo      PlayInfo          `json:"playInfo"`
    TransactionId string            `json:"transactionId"`
    Bets          []*pb.GameBetInfo `json:"bets"`
```

```
    ExtraData     map[string]any    `json:"extraData"`
}

// GameWinInfo 游戏赢取信息
type GameWinInfo struct {
    // 游戏赢取，当前spin赢取的钱，如free game，只需记录，不需要结算
    BetAmount int64 `json:"betAmount"`
    // 实际赢取，需要结算的钱，free game要等所有free次数完成后才结算
    GameWin int64 `json:"gameWin"`
    // jp 赢取
    JackpotWin int64 `json:"jackpotWin"`
}

// PlaceBetResponse 下注响应
type PlaceBetResponse struct {
    PlayInfo            PlayInfo                    `json:"playInfo"`
    TransactionId       string                      `json:"transactionId"`
    Game                map[string]any              `json:"game"`
    Bets                []*pb.GameBetInfo           `json:"bets"`
    ExtraData           map[string]json.RawMessage  `json:"extraData"`
    WalletChangeCredits *int64                      `json:"walletChangeCredits"`
    WalletAmount        *int64                      `json:"walletAmount"`
    Status              int32                       `json:"status"`
}

// authMiddleware 签名认证中间件
func authMiddleware(c *gin.Context) {
    // 获取请求头中的签名相关信息
    clientID := c.GetHeader("X-Client-ID")
    signature := c.GetHeader("X-Signature")
    nonce := c.GetHeader("X-Nonce")
    timestamp := c.GetHeader("X-Timestamp")

    // 检查必要的请求头是否存在
    if clientID == "" || signature == "" || nonce == "" || timestamp == "" {
        c.JSON(http.StatusUnauthorized, gin.H{"error": "缺少必要的请求头"})
        c.Abort()
        return
    }

    // 生成签名
    body := fmt.Sprintf("%s%s%s%s", clientID, timestamp, nonce, APIKey)
    hash := sha256.Sum256([]byte(body))
    text := hex.EncodeToString(hash[:])
    // 验证签名
    if text != signature {
        c.JSON(http.StatusUnauthorized, gin.H{"error": "签名无效"})
        c.Abort()
        return
    }
    c.Next()
}

// endpoint 处理游戏相关请求的主入口
func endpoint(c *gin.Context) {
    action := c.Query("action")
    // 根据命名空间分发处理
    var rsp map[string]any
```

```
    if strings.EqualFold(action, "InitGame") {
        rsp = handlerInitGame(c)
    } else if strings.EqualFold(action, "PlaceBets") {
        rsp = handlerPlaceBets(c)
    } else {
        c.JSON(http.StatusMethodNotAllowed, rsp)
        return
    }
    c.JSON(http.StatusOK, rsp)
}

// handlerPlaceBets 处理下注请求
func handlerInitGame(c *gin.Context) map[string]any {
    var req InitRequest
    if err := c.ShouldBind(&req); err != nil {
        c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
        return nil
    }

    return map[string]any{
        "playInfo": PlayInfo{
            GameId:  req.GameId,
            Version: "1.0.1",
        },
        "game": map[string]any{
            "name": "xxx",
            "age":  22,
        },
        "config": map[string]any{
            "version": "1.0.1",
            "line":    []int{1, 2, 3, 4, 5},
        },
    }
}

// handlerInitGame 处理游戏初始化请求
func handlerPlaceBets(c *gin.Context) map[string]any {

    var req PlaceBetRequest
    if err := c.ShouldBind(&req); err != nil {
        c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
        return nil
    }

    betAmout := int64(0)
    for _, bet := range req.Bets {
        betAmout += bet.Amount
    }

    return map[string]any{
        "game": map[string]any{
            "name": "xxx",
            "age":  22,
        },
        "extraData": map[string]any{
            "gameWinInfo": GameWinInfo{
                GameWin:    100,
                JackpotWin: 0,
```

```
                BetAmount:  betAmout,
            },
        },
    }
}

// main 程序主入口
func main() {
    r := gin.Default()

    // 使用签名认证中间件
    r.Use(authMiddleware)
    r.POST("/endpoint", endpoint)
    r.Run(":10888")
}
```