方块游戏平台API说明

**WebAPI**

* 获取订单信息

api.cubejoy.com/ICubeInAppPurchase/OrderInfo?orderid=a

ordereid: 订单号

方法:GET:

返回:

{

"result": 1,

"data": {

"orderId": "",

"state": 1,

"products": [

{

"apiName": "",

"quantity": 1

}

]

}

}

result返回值:

0: 不使用(Invalid)

1: 成功

101: token已失效或错误

201: 错误的appId

603: 错误的orderId

订单state状态码:

0: 不使用(Invalid)

1: 未支付

2: 已支付,但未发货

3: 已完成(finish)

* 设置订单完成

api.cubejoy.com/v1.0/ICubeInAppPurchase/FinishOrder

方法:POST

Content-Type: application/json

提交数据:

{

"token": "",

"orderId": "a",

"appId": 100001

}

返回:

{

"result": 1,

"data": {

"orderId": "a",

"appId": 100001

}

}

result返回值:

0: 不使用(Invalid)

1: 成功

101: token已失效或错误

201: 错误的appId

603: 错误的orderId

**API概述**

**最简单示例:**

int *main*() {

// 应首先调用IsNeedRestartApp,传入游戏id

if (Cube::IsNeedRestartApp(1000147)) {

return 0;

}

// 初始化,并验证授权

if (!Cube::CubeInitialize()) {

return 0;

}

// Do something

// 关闭

Cube::CubeShutdown();

return 0;

}

**接口概述**

**启动检查：** IsNeedRestartApp

**函数原型**

bool IsNeedRestartApp(CubeAppId\_t appId);

**游戏启动后，必须首先调用。**

**该函数会检查平台是否运行，如游戏不是从平台启动，返回true，详见返回值说明。**

**参数说明**

|  |  |  |
| --- | --- | --- |
| **参数名** | **输入/输出** | **说明** |
| **appId** | **输入** | **由平台分配给该游戏的唯一ID** |

**返回值说明**

**返回true，当前游戏应立即自行退出，平台将重新启动游戏。如未退出，后续API调用将失败或产生异常也可能造成平台无法重新启动游戏。**

**返回false，游戏可继续运行。**

**初始化：**CubeInitialize

**启动检查完成后，调用**CubeInitialize**初始化。同时完成授权验证。**

**失败时返回false，游戏应立即退出。**

函数原型 bool CubeInitialize();

关闭：CubeShutdown

游戏退出时，调用CubeShutdown释放资源。

函数原型 void CubeShutdown();

检查平台是否仍在运行IsCubeRunnging

函数原型 bool IsCubeRunnging();

平台未运行时返回false，游戏应立即退出。

获取平台安装路径GetCubeInstallPath

函数原型 const char\* GetCubeInstallPath();

返回安装平台的绝对路径，请勿持有返回的指针。

获取接口工厂：GetCubeFactory

函数原型 ICubeFactory\* GetCubeFactory();

返回ICubeFactory实例指针。

获取当前API版本号：GetApiVersion

函数原型 const char\* GetApiVersion();

返回当前API版本号。

**中国大陆游戏接入指南**

**防沉迷系统接入**

防沉迷弹窗及通知触发时间最大可能会有约3分钟延迟，属正常现象，不影响送审。

在中国大陆上线的所有游戏，均需要接入防沉迷系统，限制防沉迷账户的游戏时长和游戏时间。

**游戏启动时防沉迷细则及提示文案：**

|  |  |  |
| --- | --- | --- |
| 时间段 | 文案 | 说明 |
| 工作日游戏满1小时 | 根据健康系统限制，由于您是未成年玩家，非节假日仅能游戏1小时30分钟。您今天已经进行游戏1小时，请注意休息。 | 提醒 |
| 工作日游戏满1.5小时 | 根据健康系统限制，由于您是未成年玩家，非节假日仅能游戏1小时30分钟。您今天已经进行游戏1小时30分钟，不能继续游戏，请注意休息。 | 禁玩 |
| 节假日游戏满2小时 | 根据健康系统限制，由于您是未成年玩家，节假日仅能游戏3小时。您今天已经进行游戏2小时，请注意休息。 | 提醒 |
| 节假日游戏满3小时 | 根据健康系统限制，由于您是未成年玩家，节假日仅能游戏3小时。您今天已经进行游戏3小时，不能继续游戏，请注意休息。 | 禁玩 |
| 每日22:00~次日8:00 | 根据健康系统限制，由于您是未成年玩家，每天22:00~次日8:00无法登录游戏，请注意休息。 | 禁玩 |

**游戏中防沉迷提示弹窗：**

SDK不提供防沉迷弹框提示功能，游戏必须自己绘制弹框。

当游戏剩余时间达到规则要求时，SDK发送kCubeSystemNotify通知，此时游戏应弹出消息窗口，同时可以考虑暂停游戏。通知结构体的GetResult方法返回值为kAntiAddictionTip，GetResultData返回弹框文案和相关数据。具体格式如下：

{

"title": "温馨提示", // 弹框标题文案

"content": "根据健康系统限制，由于您是未成年玩家，非节假日仅能游戏1小时30分钟。您今天已经进行游戏1小时，请注意休息。", // 弹框显示的具体文案

"confirmButton": "确认" // 确定按钮的显示文本

}

**游戏强制退出：**

当未成年玩家达到规则限制需要强制退出游戏时，SDK发送kCubeSystemNotify通知。其GetResult()方法返回值为kAntiAddictionSystemShutdown，指明游戏因为防沉迷限制需退出。此后3分钟游戏将自动关闭，文案和数据格式与**提示弹窗**相同。

***注：如无特殊需要，开发者无需关注防沉迷具体规则和文案，只需监听并处理通知即可。具体代码可以参考AntiAddictionDemo工程***

**各模块说明**

**CubeUtils**

使用Cube::GetCubeFactory()->Utils()获取ICubeUtils接口实例指针。

**函数说明**

GetTimeCountSinceAppLaunch

函数原型：int64\_t GetTimeCountSinceAppLaunch() ();

说明：返回游戏本次运行时长,单位:秒.

GetServerUnixTime

函数原型：int64\_t GetServerUnixTime();

说明：返回服务器的UnixTime时间

**CubePersona**

使用Cube::GetCubeFactory()->Persona()获取ICubePersona接口实例指针。

**函数说明**

GetMyCubeId

函数原型：CubeId\_t GetMyCubeId();

说明：获取当前用户方块ID

返回值：当前用户的方块ID

GetPersonaNickname

函数原型：const char\* GetPersonaNickname(CubeId\_t userCubeId);

说明：获取指定用户的昵称。

参数：

userCubeId:用户的方块ID

返回值：返回指定用户的昵称，函数返回的指针可能失效，请勿持有。

GetPersonaAvatarFile

函数原型：const char\* GetPersonaAvatarFile(CubeId\_t userCubeId, AvatarQuality quality);

说明：获取指定用户的头像

参数：

userCubeId:用户的方块ID

quality:头像图片质量，详见AvatarQuality。

返回值：返回指定用户的头像文件的本地路径，函数返回的指针可能失效，请勿持有。

GetPersonaState

函数原型：PersonaState GetPersonaState(CubeId\_t userCubeId);

说明：获取用户状态

参数：

userCubeId:用户的方块ID

返回值：返回改用户状态,详见PersonaState。

**CubeFriends**

使用Cube::GetCubeFactory()->Friends()获取ICubeFriends接口实例指针。

**函数说明**

GetFriendCount

函数原型：int32\_t GetFriendCount();

说明：获取好友数量，之后可以使用GetFriendByIndex遍历获取好友方块ID。

返回值：返回好友数量。

GetFriendByIndex

函数原型：CubeId\_t GetFriendByIndex(int32\_t index);

说明：获取好友的方块ID。

参数：

index：取值应在[0, GetFriendCount()]范围内。

返回值：返回好友的方块ID

GetFriendRelationship

函数原型：FriendRelationship GetFriendRelationship(CubeId\_t friendCubeId);

参数：

friendCubeId：方块ID。

返回值：返回好友关系, 详见emun FriendRelationship

InviteUserToApp

函数原型：bool InviteUserToApp(CubeId\_t friendCubeId, const char\* connectString);

参数：

friendCubeId：方块ID。

connectString: 连接字符串

返回值：成功时返回true

调用后,

SetListenForFriendsMessages

函数原型：bool SetListenForFriendsMessages(bool isInterceptEnabled);

isInterceptEnabled为true时,允许获取用户好友间聊天内容,但仅允许获取启动期间的的内容

ReplyToFriendMessage

函数原型：bool ReplyToFriendMessage(CubeId\_t friendCubeId, const char\* messageToSend);

回复好友消息

GetFriendMessage

函数原型：int32\_t GetFriendMessage(CubeId\_t friendCubeId, uint64\_t messageId, void\* dataBuff, int32\_t dataBuffSize, ChatMessageType\* messageType);

获取好友消息

**CubeDlcs**

使用Cube::GetCubeFactory()->DLCs()获取ICubDLCs接口实例指针。

**函数说明**

IsDlcInstalled

函数原型：bool IsDlcInstalled(CubeDlcId\_t dlcId);

说明：查看该dlc是否安装。

返回值：dlc已安装时(安装即代表已有授权)返回true。

GetDlcCount

函数原型：int32\_t GetDlcCount();

说明：获取当前游戏的dlc数量，之后可以使用GetDlcDataByIndex遍历获取dlc数据。

返回值：返回当前游戏的dlc数量。

GetDlcDataByIndex

函数原型：bool GetDlcDataByIndex(int32\_t dlcIndex, CubeDlcId\_t\* dlcId, bool\* isAvailable, char\* name, int nameBufferSize);

说明：

参数：

dlcIndex：取值应在[0, GetDlcCount()]范围内。

dlcId：输出参数，用于获取dlcId

isAvailable： 输出参数，用于获取dlc是否可用。

name： 输出参数，用于获取dlc名。

nameBufferSize：name缓冲大小。

返回值：获取成功时返回true

**CubeAchievment**

使用Cube::GetCubeFactory()->Achievement()获取ICubeAchievement接口实例指针。

**函数说明**

SetAchievement

函数原型：bool AsyncSetAchievement(const char\* apiName);

说明：异步解锁当前用户用户成就

参数：

apiName: 成就的接口名

返回值：

SetAchievementProgress

函数原型：bool AsyncSetAchievementProgress(const char\* apiName, int64\_t currentValue, int64\_t unlockValue);

说明：异步设置当前用户成就进度

参数：apiName: 成就的接口名

currentValue: 将要设置的值

unlockValue: 解锁值,此值应与在开发者后台填写的解锁值一致,否则会导致设置进度失败.

返回值：

ClearAchievement

函数原型：bool ClearAchievement(const char\* apiName);

说明：异步清除用户成就

参数：

apiName: 成就的接口名

返回值：

GetAchievement

函数原型：bool GetAchievement(const char\* apiName, bool\* isUnlock, uint64\_t\* unlockTime, int64\_t\* currentValue);

说明：获取成就的解锁时间等信息.

参数：

apiName: 成就的接口名

isUnlocked: 是否解锁, 输出参数

unlockTime: 解锁时间, 输出参数

当前值:进度, 输出参数

返回值：

GetUserAchievement

函数原型：bool GetUserAchievement(CubeId\_t userCubeId, const char\* apiName, bool\* isUnlocked);

说明：获取其他用户的成就信息

参数：

返回值：

**ICubeInAppPurchase**

使用Cube::GetCubeFactory()->InAppPurchase()获取ICubeInAppPurchase实例指针

**函数说明**

AsyncRequestAllPurchasableProducts

函数原型：CubeResult AsyncRequestAllPurchasableProducts(const char\* userData);

说明：异步获取全部可购买商品信息

参数：userData

返回值：

GetProductInfo

函数原型：bool GetProductInfo(const char\* apiName, CubeInAppPurchaseProductInfo\* product);

说明：获取商品信息

参数：

返回值：

AsyncCreateOrder

函数原型：CubeResult AsyncCreateOrder(const CubeProductItem\* cartItems, int32\_t itemSize, const char\* userData);

说明：异步创建订单

参数：

返回值：

说明:

创建订单完成后将产生Cube::kCreateOrderResponse通知,通过变量resultData可获取支付地址,用户支付成功后将产生Cube::kPurchaseProductsResponse通知,此时应调用AsyncFinishOrder结束订单并为用户增加授权,对于未设置为Finish状态的订单,sdk将定期产生Cube::kPurchaseProductsResponse通知.在订单完成后可以使用AsyncGetOrderInfo来查询订单状态(也可以使用web api来检查订单状态,见文档第一部分),再决定是否给用户授权.

以下是内购回调类的简单实现:

class InAppPurchaseNotifyHandler :public Cube::ICubeNotifyHandler {

public:

InAppPurchaseNotifyHandler() {

Cube::RegisterNotify(Cube::kCreateOrderResponse, this);

Cube::RegisterNotify(Cube::kPurchaseProductsResponse, this);

Cube::RegisterNotify(Cube::kFinishOrderResponse, this);

}

~InAppPurchaseNotifyHandler() {

Cube::UnRegisterNotify(Cube::kCreateOrderResponse);

Cube::UnRegisterNotify(Cube::kPurchaseProductsResponse);

Cube::UnRegisterNotify(Cube::kFinishOrderResponse);

}

virtual void OnNotify(Cube::ICubeNotify\* param) override {

switch (param->notifyId) {

case Cube::kCreateOrderResponse:

::*ShellExecuteA*(NULL, "open", param->resultData.c\_str(), "", "", *SW\_SHOWNORMAL*);

std::*cout* << "PayUrl: " << param->resultData.c\_str() << std::*endl*;

break;

case Cube::kPurchaseProductsResponse: Cube::GetCubeFactory()->InAppPurchase()->AsyncFinishOrder(param->resultData.c\_str(), "");

break;

case Cube::kFinishOrderResponse:

{

Cube::CubeProductItem items[] = {

{"500003", 1},

};

auto inapp = Cube::GetCubeFactory()->InAppPurchase();

inapp->AsyncCreateOrder(items, 1, "");

}

break;

}

}

};

AsyncFinishOrder

函数原型：CubeResult AsyncFinishOrder(const char\* orderId, const char\* userData);

说明：异步设置订单状态为完成状态

参数：

返回值：

AsyncGetOrderInfo

函数原型：CubeResult AsyncGetOrderInfo(const char\* orderId, const char\* userData);

说明：异步查询订单状态

参数：

返回值：

说明:

**CubeMatchmaking**

使用Cube::GetCubeFactory()->Matchmaking()获取ICubeMatchmaking接口实例指针。

**函数说明**

SearchLobby

函数原型：CubeResult SearchLobby();

说明：根据条件获取大厅列表,该函数为异步函数,调用后,将重置筛选器

参数：

返回值：

AddSearchLobbyStringFilter

函数原型：void AddSearchLobbyStringFilter(const char\* fieldToSearch, const char\* valueToMatch, LobbyComparisonType comparisonType);

说明：添加字符串筛选器

参数：

返回值：

AddSearchLobbyIntFilter

函数原型：void AddSearchLobbyIntFilter(const char\* fieldToSearch, int32\_t valueToMatch, LobbyComparisonType comparisonType);

说明：添加整形筛选器

参数：

返回值：

AddSearchLobbyIntRangeFilter

函数原型：void AddSearchLobbyIntRangeFilter(const char\* fieldToSearch, int32\_t minValueToMatch, int32\_t maxValueToMatch);

说明：添加整形范围(闭区间)筛选器

参数：

返回值：

SetSearchLobbyDistanceFilter

函数原型：void SetSearchLobbyDistanceFilter(LobbyDistanceFilterType distanceFilter);

说明：设置距离筛选器,基于ip定位

参数：

返回值：

SetSearchLobbyResultCount

函数原型：void SetSearchLobbyResultCount(int32\_t maxResults);

说明：设置搜索结果的期望数, 该值大小与搜索速度成正比, 默认为10, 最低为1

参数：

返回值：

CreateLobby

函数原型：CubeResult CreateLobby(const char\* name, LobbyAuthorityType lobbyAuthority, int32\_t maxMembers);

说明：创建大厅

参数：

返回值：

JoinLooby

函数原型：CubeResult JoinLooby(CubeId\_t lobbyId);

说明：加入大厅

参数：

返回值：

LeaveLooby

函数原型：void LeaveLooby(CubeId\_t lobbyId);

说明：离开大厅

参数：

返回值：

InviteUserToLooby

函数原型：bool InviteUserToLooby(CubeId\_t lobbyId, CubeId\_t inviteeCubeId);

说明：邀请玩家进入大厅

参数：

返回值：

KickUser

函数原型：void KickUser(CubeId\_t lobbyId, CubeId\_t kickUserCubeId));

说明：将对方踢出大厅

参数：

返回值：

GetLoobyMemberCount

函数原型：int32\_t GetLoobyMemberCount(CubeId\_t lobbyId);

说明：获取大厅成员数量

参数：

返回值：

GetLoobyMemberByIndex

函数原型：CubeId\_t GetLoobyMemberByIndex(CubeId\_t lobbyId, int32\_t memberIndex);

说明：获取成员的方块id

参数：

返回值：

GetLoobyData

函数原型：bool GetLoobyData(CubeId\_t lobbyId, const char\* field, char\* valueBuffer, int32\_t valueBufferSize);

说明：获取大厅数据

参数：

返回值：

ReloadLoobyData

函数原型：bool ReloadLoobyData(CubeId\_t lobbyId);

说明：重新读取大厅数据

参数：

返回值：

SetLoobyData

函数原型：bool SetLoobyData(CubeId\_t lobbyId, const char\* field, const char\* value);

说明：设置大厅数据

参数：

返回值：