GAME ANALYTICS

TalkingData 用数据说话

接入指南 2.0.3

最后修正：2013-6-3

目录

一、 综述

1. 适用范围

TalkingData GameAnalytics 帮助游戏开发者解决玩家数据收集至数据标准化分析的全部繁琐问题，以行业标准指标形式将数据展现于报表中。

2. 统计标准

为了与游戏自有体系更好的结合，统计中我们采用游戏自身的帐户来做为一个玩家单元。数据中除了玩家基础信息外，主要帮助处理游戏过程中的升级、任务、付费、消费等详细行为数据。行为标准可参考以下说明：

- 玩家
  游戏自身的帐户，通常为一个唯一号、唯一名或一份唯一存档号。是平台中计算数据的最基础单元。
- 设备
  指一台安装了游戏包的终端。
- 玩家的一次游戏
  玩家从打开游戏界面至离开游戏界面的完整过程，如果玩家在离开游戏界面后 30 秒内重新回到游戏中，将被认为是上次游戏被打扰后的延续，记为一次完整游戏。
- 付费
  特指玩家充值现金换取虚拟币的过程。充值的现金将作为收入进行统计。
- 消费
  指玩家在一个消费点上消耗虚拟币的过程。

二、 接入流程

STEP 1、为游戏申请 APP ID

进入 talkinggame.com 网站，使用您的注册账号登录后，请预先创建一款游戏，您将获得一串 32 位的 16 进制 APP ID，用于唯一标识您的一款游戏。

STEP 2、解压缩

解压缩目录最好和 cocos2d-x 放到平级目录下，对于配置一些路径会方便很多。

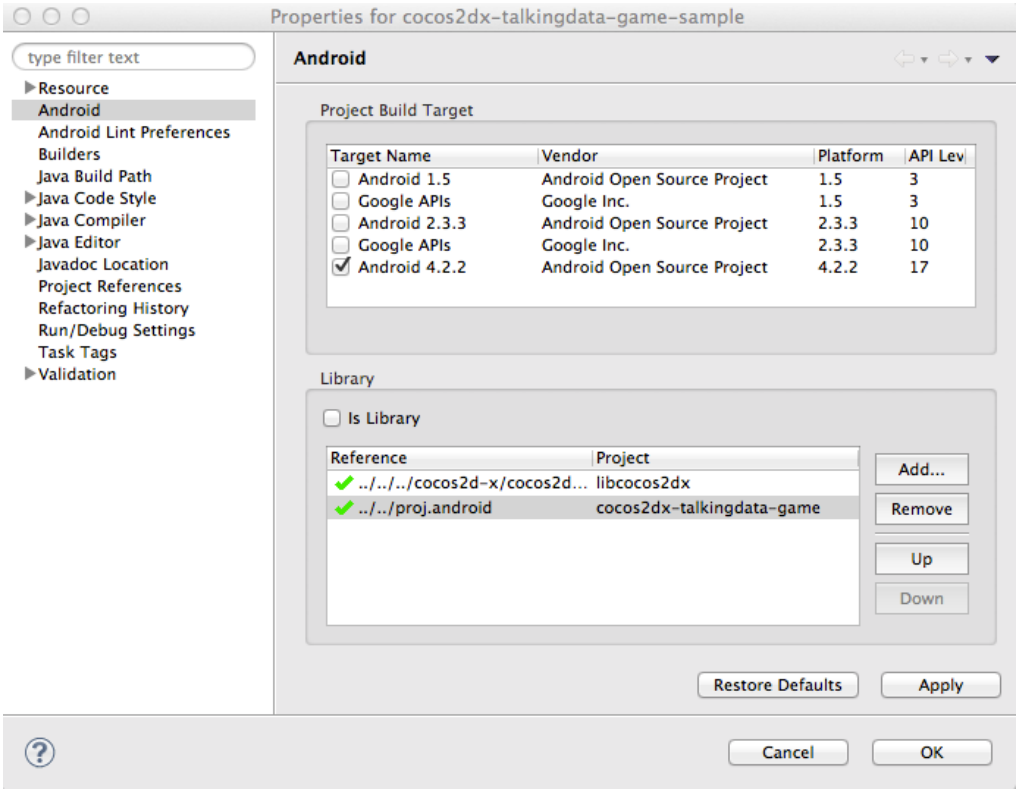
STEP 3、导入依赖工程

A). Android

1. 在 Eclipse 里选择 File->Import 选择 cocos2dx-talkingdata/proj.android 目录，并导入到 Eclipse 中。

2. 在游戏工程中添加对上一步引入的工程的依赖。

选中工程点 File->Preperties->android



3. 修改游戏项目中的 Android.mk

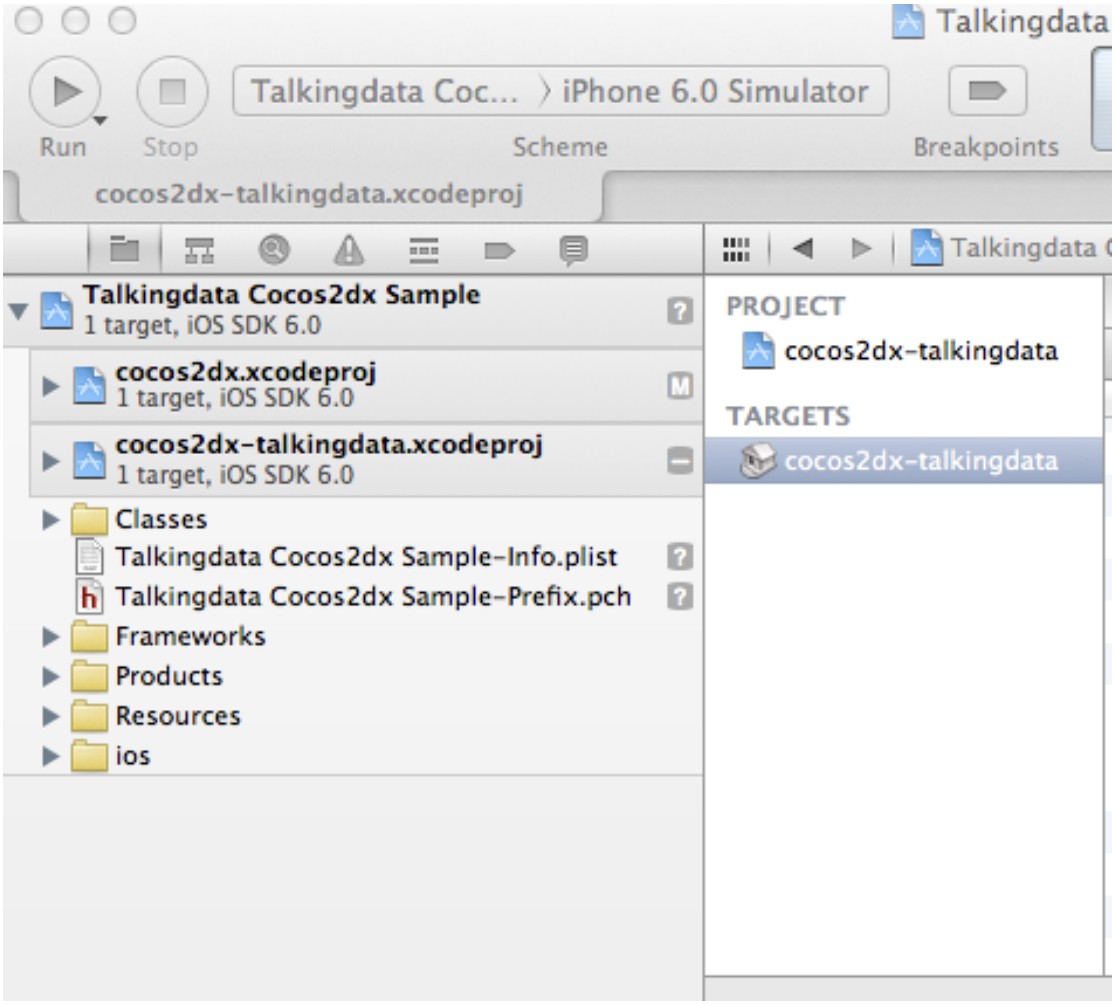
LOCAL_C_INCLUDES 添加依赖工程中的 include 目录.

LOCAL_WHOLE_STATIC_LIBRARIES 添加依赖工程编译的 module 名字.

可以参考 cocosdx-talkingdata/sample/proj.android

b). iOS

1. 导出工程. 直接拖拽 cocos2dx-talkingdata/proj.ios 下的 ios 工程到 xcode 的游戏工程中.



2. 修改游戏项目中的 Build Setting

Search->Header Search Paths

Search->Library Search Paths

可以参考 cocos2dx-talkingdata/sample/proj.ios

STEP 4、添加调用方法

参考“三、添加调用方法“的指导来完成开发。

STEP 5、进行数据测试

调用方法添加完毕后，应当对游戏打包并进行数据测试，以确保打包的正确。

三、 添加调用方法

1. 游戏启动和关闭

用途和用法

- 用于准确跟踪玩家的游戏次数，游戏时长等信息。
- Android
  - 在启动的 activity 中的 `TalkingDataGA.onResume`, `TalkingDataGA.onPause` 中添加统计调用代码。
- iOS
  - 游戏启动时调用 `TDCCTalkingDataGA::onStart`
- 可以选择性的在 channelId 中填入推广渠道的名称 ,数据报表中则可单独查询到他们的数据。每台设备仅记录首次安装激活的渠道，更替渠道包安装不会重复计量。不同的渠道 ID 包请重新编译打包。

接口说明：（TDCCTalkingDataGA 类）

static void onStart(const char* appID, const char* channelId);　(只用于 iOS)

参数说明：

参数	类型	描述
appId	char	填写创建游戏时获得的 App ID，用于唯一识别您的游戏。
channelId	char	用来标注游戏推广渠道，区分玩家的来源来查看统计。 格式：32 个字符内，支持中文、英文、数字、空格、英文点和下划线，请勿使用其他符号。

示例代码：

```
TDCCTalkingDataGA::onStart("APP_ID", "CHANNEL_ID");
```

2．统计玩家帐户

用途和用法

- 定义一个玩家单元，更新玩家最新的属性信息。
- 在可确定玩家帐户后尽早调用 TDCCAccount::setAccount(const char* accountId)方法来传入 accountId ,对一个玩家的分析依赖于这个 accountId。
- accountId 需保证全局唯一，且终生不变。在多设备中传入同样 accountId，数据将归入同一个玩家内，玩家数不增加。
- 在玩家某属性（含等级属性）被确定或发生了变化时，尽快调用对应 set 方法来更新帐户属性。

接口说明：（TDCCAccount 类）

```
//返回用户对象
TDCCAccount* setAccount(const char* accountId)
//设置帐户类型
static void setAccountType (TDCCAccountType accountType)
//设置帐户的显性名
static void setAccountName(const char* accountName)
//设置级别
static void setLevel (int level)
//设置性别
static void setGender (TDCCGender gender)
//设置年龄
static void setAge (int age)
//设置区服
static void  区服：setGameServer（const char* gameServer)
```

参数说明：

参数	类型	描述
accountId	char	设定帐户唯一标识，用于区分一个玩家，最多 64 个字符。其他调用依赖于此 ID，未变更此 ID 前，属性和行为均计入此 ID。
accountType	TDCCAccountType	传入帐户的类型。系统预定义的类型为： 匿名: TDCCAccountType. kAccountAnonymous

		自有帐户显性注册：TDCCAccountType. kAccountRegistered 国内主流的第三方帐号: 新浪微博：TDCCAccountType. kAccountSianWeibo QQ：TDCCAccountType. kAccountQQ 腾讯微博：TDCCAccountType. kAccountTencentWeibo 网龙 91：TDCCAccountType. kAccountND91 另外系统预留了 10 种自定义的帐户类型，分别为 TDCCAccountType. kAccountType1 到 TDCCAccountType. kAccountType10
accountName	char	在帐户有显性名时，可用于设定帐户名，最多支持 64 个字符。
Level	int	设定玩家当前的级别，在玩家等级可确定和有变化时尽早调用。平台默认的初始等级为“0”，支持的最大级别为 1000。
gender	TDCCGender	设定玩家性别： 男: TDCCGender. kGenderMale 女: TDCCGender. kGenderFemale 未知: TDCCGender. kGenderUnknown
age	int	设定玩家年龄，范围为 0-120。
gameServer	char	传入玩家登入的区服，最多 16 个字符。

示例 1：

如一个游戏内 UID 为 10000 的匿名游戏玩家以匿名（快速登录）方式在国服 2 区进行游戏，并在游戏中由 1 级升至 2 级，之后玩家使用 QQ 号 5830000 在游戏中进行显性注册，并设定为 18 岁男玩家的整个过程。

```
TDCCAccount* account = TDCCAccount::setAccount(“10000”);
account->setAccountType(kAccountAnonymous);
account->setLevel(1);
account->setGameServer(“国服 2”);
```

在玩家升级时，做如下调用

```
account->setLevel(2);
```

在玩家显性注册成功时做如下调用

```
account->setAccountName(“5830000@qq.com”);
account->setAccountType(kAccountAnonymous);
account->setAge(18);
account->setGender(kGenderMale);
```

示例 2：

如一款必须在注册后进行的不分区服的游戏中，玩家使用其已注册的帐号：xiaoming@163.com（游戏服务器中为玩家分配了 101111 的内部号）登录进行游戏。

```
TDCCAccount * account = TDCCTalkingDataGA::setAccount(“101111”);
account->setAccountName(“xiaoming@163.com”);
account->setAccountType(kAccountAnonymous);
```

示例 3：

如在一款类似愤怒小鸟的休闲游戏中，玩家直接进入进行游戏。

您可自行决定唯一 ID 规则，如设备 IMEI、MAC 地址等，也可以使用 TalkingData 提供的设备唯一 ID 接口，例如：

```
TDCCAccount* account =
TDCCTalkingDataGA::setAccount(TDCCTalkingDataGA::getDeviceId());
account->setAccountType(kAccountAnonymous);
```

3．跟踪玩家充值

用途和用法

- 跟踪玩家充值现金而获得虚拟币的行为，充入的现金将反映至游戏收入中。
- 充值过程分两个跟踪阶段，发送出有效的充值请求，和服务端确认充值已成功。在发起充值请求时调用 onChargeRequest；在玩家充值成功时调用 onChargeSuccess。

接口说明：（TDCCVirtualCurrency 类）

```
//充值请求
static void onChargeRequest(const char* orderId, const char* iapId, double currencyAmount, const char* currencyType, double virtualCurrencyAmount, const char* paymentType)
//充值成功
static void OnChargeSuccess (const char* orderId)
```

参数说明：

参数	类型	描述
orderId	char	订单 ID，最多 64 个字符。
iapId	char	充值包 ID，最多 32 个字符。 例如：VIP3 礼包、500 元 10000 宝石包
currencyAmount	double	现金金额或现金等价物的额度
currencyType	char	请使用国际标准组织 ISO 4217 中规范的 3 位字母代码标记货币类型。点击查看参考 例：人民币 CNY；美元 USD；欧元 EUR （如果您使用其他自定义等价物作为现金，亦可使用 ISO 4217 中没有的 3 位字母组合传入货币类型，我们会在报表页面中提供汇率设定功能）
virtualCurrencyAmount	double	虚拟币金额
paymentType	char	支付的途径，最多 16 个字符。 例如：“支付宝” “苹果官方” “XX 支付 SDK”

示例 1：

玩家使用支付宝方式成功购买了“大号宝箱”（实际为 100 元人民币购入 1000 元宝的礼包），该笔操作的订单编号为 order001。可以如下调用：

1）在向支付宝支付 SDK 发出请求时，同时调用：

```
TDCCVirtualCurrency::onChargeRequest(“order001”, “大号宝箱”, 100, “CNY”, 1000, “AliPay”);
```

2）订单 order001 充值成功后调用：

```
TDCCVirtualCurrency::onChargeSuccess(“order001”);
```

示例 2：

在一款与 91 联运的游戏中，游戏使用了 91 的支付聚合 SDK，玩家购买一个“钻石礼包 1”（10 个 91 豆购买 60 钻石），该笔操作的订单号为“7837331”。由于此类聚合 SDK 往往要求使用其自有的“代币”（91 使用 91 豆，兑换人民币比例 1：1）做充值依据，建议将“代币”折算为人民币后再调用统计：

1）在向 91 支付 SDK 发出请求时，进行调用

```
TDCCVirtualCurrency::onChargeRequest(“7837331”，“钻石礼包 1”，10,
“CNY”，60，“91 SDK”);
```

2）订单 order001 充值成功：

```
TDCCVirtualCurrency::onChargeSuccess(“7837331”);
```

4. 跟踪获赠的虚拟币（可选）

用途和用法

- 游戏中除了可通过充值来获得虚拟币外，可能会在任务奖励、登录奖励、成就奖励等环节免费发放给玩家虚拟币，来培养他们使用虚拟币的习惯。开发者可通过此方法跟踪全部免费赠予虚拟币的数据。
- 在成功向玩家赠予虚拟币时调用 onReward 方法来传入相关数据。
- 只获得过赠予虚拟币的玩家不会被记为付费玩家。赠予的虚拟币会计入到所有的虚拟币产出中，也计入到留存虚拟币中。

接口说明：（TDGAVirtualCurrency 类）

```
//赠予虚拟币
public static void onReward (double virtualCurrencyAmount, String reason)
```

参数说明：

参数	类型	描述
virtualCurrencyAmount	double	虚拟币金额。
reason	String	赠送虚拟币原因/类型。 格式：32 个字符内的中文、空格、英文、数字。 不要带有任何开发中的转义字符，如斜杠。 注意：最多支持 100 种不同原因。

示例 1：

玩家在完成了新手引导后，成功获得了免费赠送的 5 个钻石：

```
TDGAVirtualCurrency.onReward(5, “新手奖励”);
```

示例 2：

玩家在游戏竞技场中排名较高，而获得了 100 消费券奖励：

```
TDGAVirtualCurrency.onReward(100, “竞技场 Top2”);
```

5．跟踪游戏消费点

用途和用法

- 跟踪游戏中全部使用到虚拟币的消费点，如购买虚拟道具、VIP 服务、复活等
- 跟踪某物品或服务的耗尽
- 在任意消费点发生时尽快调用 onPurchase，在某个道具/服务被用掉（消失）时尽快调用 onUse
- 消费点特指有价值的虚拟币的消费过程，如果游戏中存在普通游戏金币可购买的虚拟物品，不建议在此处统计。

接口说明：（TDCCItem 类）

```
//记录付费点
static void onPurchase(const char* item, int itemNumber, double
priceInVirtualCurrency)
//消耗物品或服务等
static void onUse(const char* item, int itemNumber)
```

参数说明：

参数	类型	描述
item	char	某个消费点的编号，最多 32 个字符。
itemNumber	int	消费数量
priceInVirtualCurrency	double	虚拟币单价

示例 1：

玩家以 25 元宝/个的单价购买了两个类别号为“helmet1”的头盔，可以调用：

```
TDCCItem::onPurchase(“helmet1”, 2, 25);
```

其中一个头盔在战斗中由于损坏过度而消失。

```
TDCCItem::onUse(“helmet1”, 1);
```

示例 2：

玩家在某关卡中死亡，使用 5 个钻石进行复活。可调用：

```
TDCCItem.onPurchase(“revival”, 1, 5);
```

6. 任务、关卡或副本

用途和用法

- 跟踪玩家任务/关卡/副本的情况。

接口说明：（TDCCMission 类）

```
//接受或进入
static void onBegin(const char* missionId)
//完成
static void onCompleted(const char* missionId)
//失败
static void onFailed(const char* missionId, const char* cause)
```

参数说明：

参数	类型	是否必填	描述
missionId	char	必填	任务、关卡或副本的编号，最多 32 个字符。
cause	char	必填	失败原因 ,最多 16 个字符。共支持 100 种原因，此处可填写 ID，别名可在报表编辑。

示例 1：

玩家进入名称为“蓝色龙之领地”的关卡。可调用：

```
TDCCMission::onBegin(“蓝色龙之领地”);
```

玩家成功打过了关卡：

```
TDCCMission::onCompleted(“蓝色龙之领地”);
```

示例 2：

玩家接到了“主线任务 5”后，又接受了某个直线任务“赚钱 1”，之后他在赚钱任务 1 进行中因为觉得任务过难，放弃任务而失败。

```
TDCCMission::onBegin(“主线任务 5”);
TDCCMission::onBegin(“赚钱 1”);
TDCCMission::onFailed(“赚钱 1”， “quit”);
```

7. 自定义事件

用途和用法

- 用于统计任何您期望去跟踪的数据，如：点击某功能按钮、填写某个输入框、触发了某个广告等。
- 可以自行定义 eventId，在游戏中需要跟踪的位置进行调用，注意 eventId 中不要加空格或其他的转义字符。
- 除了可以统计某自定义 eventId 的触发次数外，还可以通过 key-value 参数来对当时触发事件时的属性进行描述。如定义 eventId 为玩家死亡事件，可添加死亡时关卡、死亡时等级、死亡时携带金币等属性，通过 key-value 进行发送。
- 每款游戏可定义最多 200 个不同 eventId，每个 eventId 下，可以支持 20 种不同 key 的 500 种不同 value 取值（char 类型），并且注意每个单次事件调用时，最多只能附带 10 种不同 key。

接口说明

- 在游戏程序的 event 事件中加入下面格式的代码，也就成功的添加了一个简单的事件到您的游戏程序中了：

  ```
  static void onEvent(const char* eventId, EventParamMap* map = NULL);
  ```

示例 1：

使用自定义事件跟踪玩家的死亡情况，并记录死亡时的等级、场景、关卡、原因等信息，可在玩家死亡时调用：

```
EventParamMap paramMap;
paramMap.insert(EventParamPair("level ", "50-60"));
paramMap.insert(EventParamPair("map ", "沼泽地阿卡村"));
paramMap.insert(EventParamPair("mission ", "屠龙副本"));
paramMap.insert(EventParamPair("coin ", " PK 致死"));
TDCCTalkingDataGA::onEvent("dead ", &paramMap);
```

注：在某 key 的 value 取值较离散情况下，不要直接填充具体数值，而应划分区间后传入，否则 value 不同取值很可能超过平台最大数目限制，而影响最终展示数据的效果。

如：示例中金币数可能很离散，请先划分合适的区间。

示例 2：

使用自定义事件跟踪玩家在注册过程中每个步骤的失败情况：

```
void XXXX::onClick() {
TalkingDataGA.OnEvent("eventId", dic);
EventParamMap paramMap;
paramMap.insert(EventParamPair("step ", "第一步"));
TDCCTalkingDataGA::onEvent("eventId ", &paramMap); // 在注册环节的每一步完成时，以步骤名作为 value 传送数据
}
```