天梯游戏统计SDK

Unity 接入文档

版本：1.7.0

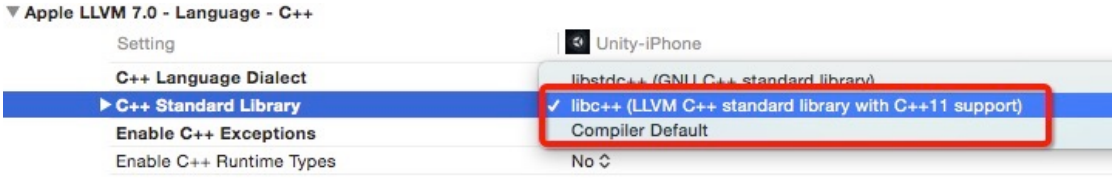
时间：2016/12/19

1. Unity SDK 导入

在 Unity 编译器中选择 Assets -> Import Package -> Custom Package 找到本地目录下的 AppLogger_c11_XXX.unitypackage 文件（或：AppLogger_c99_XXX.unitypackage 两者区别见下文），点击 Open 按钮，然后 Import 即可导入成功。

由于开发者选用的 c++ 标准库可能有所不同（是否支持 C++0x），所以我们对 liblogger 也做了对应的区分，分别为：

1.1 如果使用LLVM C++ standard library with C++11 support (如下图) 或者 Complier Default,使用 AppLogger_c11.unitypackage



1.2 如果使用GNU C++ standard library (如下图),使用 AppLogger_c99.unitypackage

2. Unity SDK 初始化

2.1 定义

void init (const char* appId, const char* appChannel=NULL);

在初始化游戏的 MonoBehaviour 子类中添加如下代码，Awake()和 Start() 中均可以添加:

```
using Tianti;

void Awake() {
    …

    AppLogger.init ("appId", "appChannel");
}
```

appId: 应用的唯一标识，在天梯游戏统计后台创建应用时自动生成。
appChannel: 应用分发渠道名

2.2 功能

初始化数据统计服务

2.3 用法

必须指定 appId 和 appChannel

2.4 备注

确保在程序启动的时候调用初始化接口，仅在第一次调用时有效。
Android和iOS的appkey不一样,注意做好区分

如果需要临时禁用天梯游戏统计相关服务，不需要逐一注释相关代码，只需要注释掉 init 函数调用，天梯游戏统计 将不会记录，保存，传输任何数据，所有接口调用直接忽略。

现在可以登录天梯游戏统计后台，然后运行您的游戏，即可实时看到当前启动游戏的记录，实时数据。如图：

恭喜，基本的统计 SDK 已经接入完成！

如果您的项目是关卡类型游戏或者需要统计分析移动应用的具体页面数据，您可以继续接入“关卡与页面”。

3. 关卡与页面

3.1 关卡或页面开始 onSubStart

3.1.1 定义

void onSubStart (const char* name);

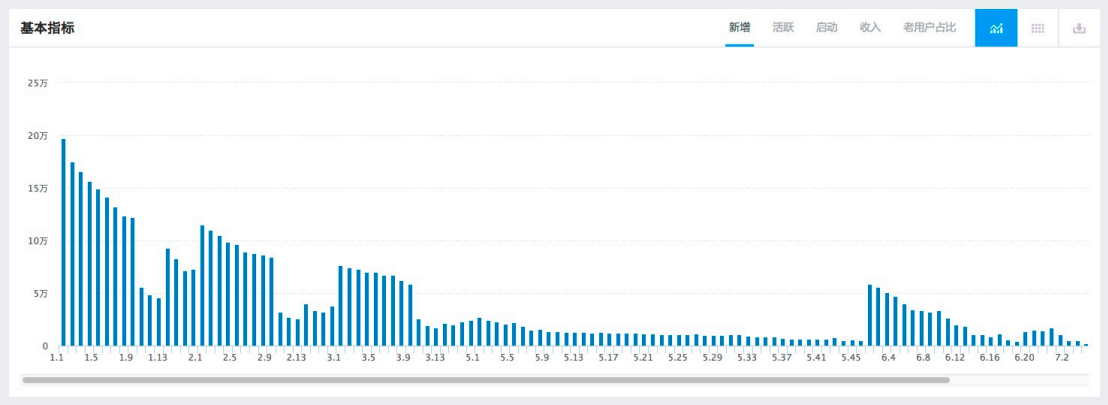
3.1.2 功能

进入一个关卡或页面

3.1.3 用法

在进入一个关卡或页面时，调用 onSubStart 接口

onSubStart 事件接口可以帮助统计系统获得关卡 新增用户，活跃用户，启动次数，进入次数，离开次数，滞留人数，最大关卡，最后关卡 等各种指标数据。如图：



3.1.4 备注

关卡/页面可以嵌套，但不可以交叉

onSubStart 的第一个参数 stage 是关卡的名称，用版本号的形式。例如："1"; "2"; "2.1"; "3"; "3.2"; "3.3" ...， 如果是字母开始的名称（例如：“MainScene”），将作为页面统计

3.2 关卡或页面结束 onSubEnd

3.2.1 定义

void onSubEnd (const char* name);

3.2.2 功能

退出指定关卡或页面

3.2.3 用法

在退出一个关卡或页面时，调用 onSubEnd 接口

3.2.4 备注

onSubEnd 必须与 onSubStart 匹配

onSubEnd 不能用 STAGE_LAST 参数，必须指定和对应 onSub-Start 一样的关卡名称

3.3 关卡过关或失败 onPassFail

3.3.1 定义

void onPassFail (bool bPass);

3.3.2 功能

过关或失败

3.3.3 用法

关卡过关或失败时，立即调用 onPassFail 接口

3.3.4 备注

onPassFail 在 onSubStart 与 onSubEnd 之间调用

onPassFail 事件接口可以帮助统计系统获得关卡的难度等各种细节信息，例如，过关时间，失败时间，难度 等数据

关卡过关失败数据对于分析关卡数据非常有意义，强烈建议开发者及时并正确调用 on-PassFail 接口

4. 关卡与页面子事件

所有子事件缺省都算在最后关卡中（STAGE_LAST）

如果需要把子事件计算到特定关卡或页面，请指定子事件中的 stage 参数

4.1 现金购买 onBuy

4.1.1 定义

void onBuy (const char* payService, const char* item, int count, float value, const char* stage = STAGE_LAST);

4.1.2 功能

现金购买虚拟币， onBuy 事件接口提交的数据，将作为收入数据，在后端报表中用各种收入图表展现。

4.1.3 用法

当用现金进行购买时，请调用 onBuy 事件接口

onBuy 事件缺省统计到最后关卡中，所以你可以看到各个关卡的各种与收入相关的数据，如，收入，新增付费用户，活跃付费用户，当天购买用户

4.1.4 备注

payService: 支付服务类型，例如：支付宝, paypal, appstore, …

item: 购买的项目

count: 购买数量，通常是1

value: 购买总价

stage: 发生事件时的关卡位置，缺省为最后关卡（STAGE_LAST）

4.2 虚拟货币交易 onExchange

4.2.1 定义

void onExchange (const char* item, int count, const char* stage = STAGE_LAST);

4.2.2 功能

虚拟币兑换为道具

4.2.3 用法

数据分析人员可以观察到各个关卡的兑换次数，兑换项目等详细数据，以及各个数据的分布状况

4.2.4 备注

item: 兑换的项目
count: 兑换数量，通常是1
stage: 发生事件时的关卡位置，缺省为最后关卡（STAGE_LAST）

4.3 道具使用 onUse

4.3.1 定义

void onUse (const char* item, int count, const char* stage = STAGE_LAST);

4.3.2 功能

使用道具或点数

4.3.3 用法

4.3.4 备注

item: 使用的物品(或点数)名称，例如道具名称
count: 使用的物品的数量
stage: 使用的位置，关卡或页面名称

4.4 收集 onCollect

4.4.1 定义

void onCollect (const char* item, int count, const char* stage = STAGE_LAST);

4.4.2 功能

收集点数/道具

4.4.3 用法

4.4.4 备注

item: 收集的物品(或点数)名称，例如道具名称
count: 收集的物品的数量
stage: 收集的位置，关卡或页面名称

4.5 奖励 onReward

4.5.1 定义

void onReward (const char* item, int count, const char* stage = STAGE_LAST);

4.5.2 功能

奖励虚拟币或道具

4.5.3 用法

发生奖励行为时，立即调用 onReward 接口

4.5.4 备注

item: 奖励的物品(或点数)名称，例如道具名称
count: 奖励的物品的数量
stage: 奖励的位置，关卡或页面名称

4.6 分享 onShare

4.6.1 定义

void onShare (const char* service, const char* item, const char* stage = STAGE_LAST);

4.6.2 功能

分享到社交服务上

4.6.3 用法

当用户进行分享操作时，立即调用 onShare 事件接口

4.6.4 备注

service: 分享服务的名称，例如，weibo, qq, weixin, …
item: 分享的项目名称
stage: 分享的位置，关卡或页面名称

4.7 自定义事件 onEvent

4.7.1 定义

void onEvent (const char* name, const char* stage = STAGE_LAST);

4.7.2 功能

用户自定义事件

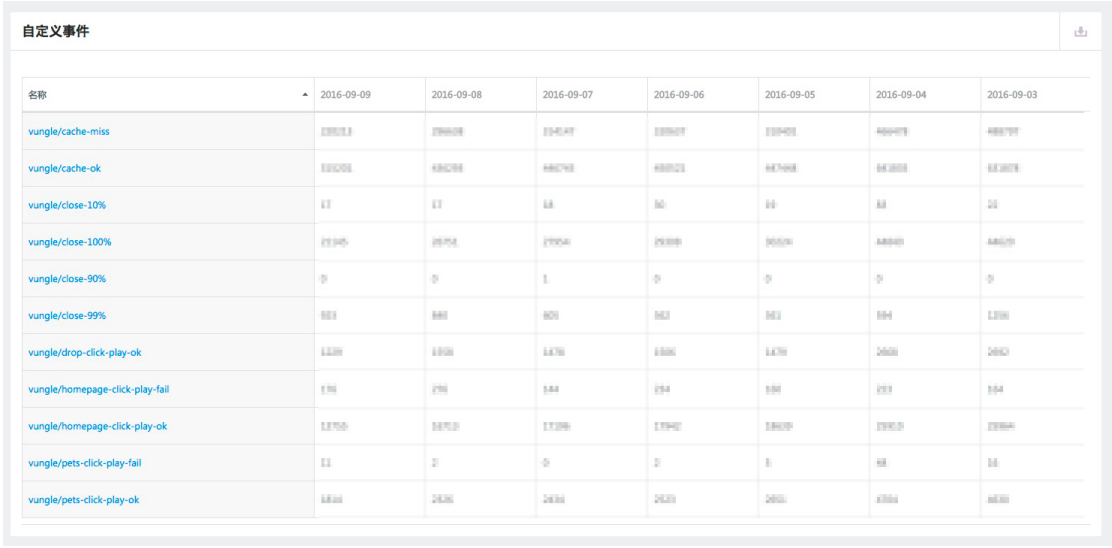
4.7.3 用法

自定义事件将以列表形式罗列出来，并显示最近几天的自定义事件变化

自定义事件可以是一个简单的名称，例如 dm_show，也可以是包含 / 的多级名称，例如 main/dm_show, main/dm_close, …

对于 main/dm_show 形式的事件，在统计报表中将把所有 main/ 开头的时间一起对比显示。所以如果想对比观察一组事件，可以给这些事件一个一样的分组名称作为前缀，并用 / 分隔。

事件列表：



相同前缀的事件之间对比：

5. 在线参数

5.1 启用在线参数

5.1.1 定义

void enableOnlineConfig ();

5.1.2 功能

启用在线参数同步功能，缺省不启用在线参数功能

5.1.3 用法

在 init 接口后调用 enableOnlineConfig

5.1.4 备注

在线参数在应用统计的后台管理，添加参数时可以指定当前参数的有效期。
客户端每次启动或者从后台切换到前台时自动更新在线参数，并且最小更新间隔为15分钟。

5.2 获取在线参数值

5.2.1 定义

string getOnlineConfig (string key, string defaultValue="");

5.2.2 功能

获得某个在线参数的具体数据，如果没有指定，返回 defaultValue

5.2.3 用法

先调用 enableOnlineConfig，然后可以任意多次调用 getOnline-Config

5.2.4 备注

服务器可以指定在线参数的 key 和 数据，类型由客户端自己处理
服务器端可以指定在线参数的有效期，如果在有效期之外，相当于服务器没有定义该参数，返回缺省值

6. 状态数据统计

6.1 清除状态数据

6.1.1 定义

void clearStatus();

6.1.2 功能

清除所有状态数据

6.1.3 用法

6.1.4 备注

在需要清除所有状态数据时，通常不需要调用

6.2 设置状态数据

6.2.1 定义

void setStatus(const char* key, int value, const char* distribution = NULL);

6.2.2 功能

设置状态数据，服务器端将对状态数据自动分区统计，或根据指定的区统计

例如：玩家当前有多少宝石，用了多少宝石，收集了多少宝石等等这类长期累积的数据可以通过 setStatus 的接口统计。

6.2.3 用法

通常在 init 之后尽快调用 setStatus 接口，把客户端的累计状态数据汇报给服务器

状态数据每天第一次启动的时候提交到服务器

6.2.4 备注

状态数据设置后会保存下来，下次启动如果没有更新，自动使用之前设置的数据，除非重新设置或clearStatus

通常在应用启动后，客户端持续状态数据加载后立即调用 AppLog-ger::setStatus 接口

持续状态变更时也立即调用 AppLogger::setStatus 接口

SDK 在每天第一次启动时自动同步数据到服务器，不会导致数据重复统计

7. 开发者

7.1 启用错误统计

7.1.1 定义

void enableCrashReport();

7.1.2 功能

启用错误统计，捕捉应用中发生崩溃时的堆栈信息，用于定位程序中引起错误的代码，缺省为不启用崩溃统计

7.1.3 用法

在 init 接口之后调用 enableCrashReport

7.1.4 备注

需要把其他统计服务的错误收集功能关闭，例如：

友盟：setCrashReportEnabled

TalkingData：setExceptionReportEnabled

7.2 日志输出设置

7.2.1 定义

void setDebugLog (int mode, int level);

7.2.2 功能

调试信息输出控制

7.2.3 用法

mode 与 level 具体定义：

```
#define DEBUG_OFF       0    //不输出调试信息
#define DEBUG_LOGCAT    1    //输出到 logcat(用于联机调试)
#define DEBUG_LOGFILE   2    //输出到 文件（applogger.log）, 用于脱机调试
#define DEBUG_LOGALL    3    //输出到 logcat 和 文件（applogger.log）

#define DEBUG_LEVEL_VERBOSE    0    //输出所有级别调试信息
#define DEBUG_LEVEL_INFO       1    //输出INFO级别以上调试信息
#define DEBUG_LEVEL_WARN       2    //输出WARN级别以上调试信息
#define DEBUG_LEVEL_ERROR      3    //输出ERROR级别以上调试信息
```

7.2.4 备注

开发者可以使用 AppDebug::info, AppDebug::warn, AppDebug::error 等接口输出调试信息到指定输出上

8. 其他

8.1 启动间隔控制参数

8.1.1 定义

void setSessionInterval (int v);

8.1.2 功能

设置合并间隔，当上次退出时间与本次启动时间间隔小于该参数所指定的值时，合并上次与本次启动

8.1.3 用法

8.1.4 备注

为与其他统计工具保持一致，缺省情况下，在iOS 下间隔时间为0（不合并），Android 下间隔时间为30秒

8.2 联网控制

8.2.1 定义

void setOnline ();

8.2.2 功能

用于使用邀请码的小范围测试模式的时候，验证完激活码后，才算正常初始化 SDK

8.2.3 备注

一般不需要使用该接口，通常用于使用邀请码的小范围测试模式的时候

启动时，调用 setOnline 接口设为 false， setOnline (false);

用邀请码并登录游戏，调用 setOnline 接口设为 true，setOnline (true);

8.3 玩家/用户参数

8.3.1 定义

void setUser (const char* level, int age = -1, const char* gender = NULL, const char* userId = NULL, const char* userService = NULL);

8.3.2 功能

设置用户信息，包括用户级别，年龄，性别, …

8.3.3 用法

8.3.4 备注

8.4 SDK 接口调用检查

~~onStart // 应用启动~~

onSubStart // 关卡或页面的开始

// 关卡中发生的事件

onBuy, onUse, onEvent … 等事件接口

onPassFail 关卡成功或失败

onSubEnd // 关卡或页面的结束

// 关卡外发生的事件

~~onEnd // 应用退出，Back按键强制退出调用onExit~~

onStart, onEnd 在 Unity 版本的SDK中自动调用

onSubStart, onSubEnd 表示一个关卡/页面的开始与结束，可以嵌套，但不能交叉

onSubStart 的第一个参数stage是关卡名称，用版本号的形式，例如 “1”,”2”, “2.1”, “2.2”,…，如果是字母开始，作为页面