GT API UserGuide For Android

目录

前言

本说明书针对 GT Demo for Android 工程讲解如何使用 GT SDK。

GT Demo 的功能就是从网上下载 10 张图片并显示，通过 GT 可以实时查看下载的带宽、单张速度、CPU、MEM 等指标，还能通过 GT 修改下载线程数、超时时间等等。该 Demo 主要是为了演示 GT 是如何脱机调试一个 APP 的。

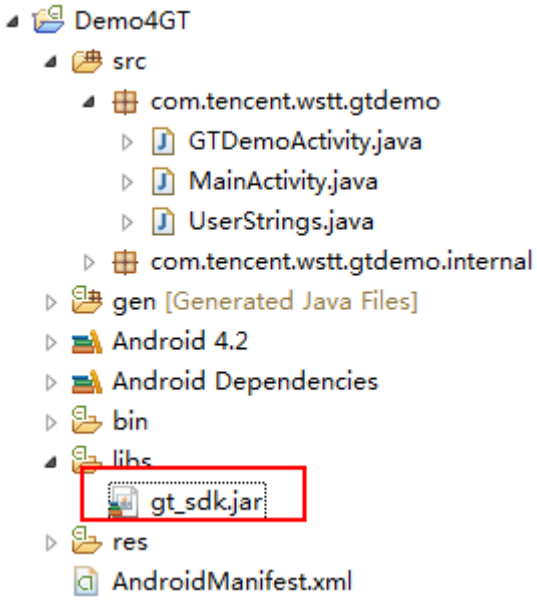
说明：GT Demo for Android 工程中使用 GT SDK 的地方可以通过搜索 GT Usage 查找。

合入 GT

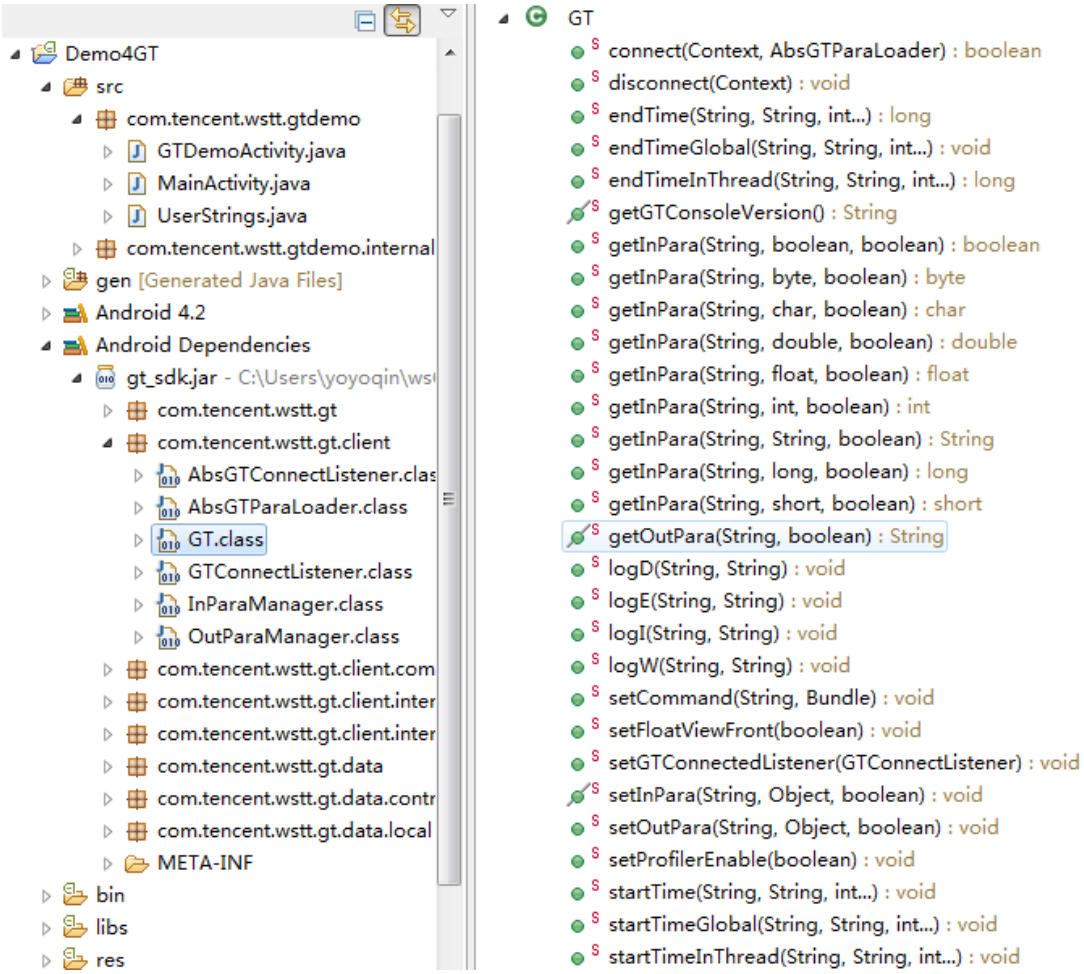
使用 GT 在被测应用中插桩这一高级功能需要在被测工程中合入 GT SDK，合入 GT SDK 分三步，首先要在手机中安装 GT.apk，然后将 SDK jar 包添加到工程中；最后是在合适的位置插入 GT 初始化的代码 (demo 工程、SDK jar 包、GT.apk 请到 GT 官网下载) 。

添加 SDK jar 包

- 拷贝 SDK jar 包到工程中的 libs 目录中，如下图：



此时，demo 工程就包括了 GT 的 API：



GT 初始化

- 在合适的位置插入 GT 初始化的代码：

GT SDK 的初始化需要在被测工程中调用 GT.connect(Context, AbsGTParaLoader)方法建立被测应用与 GT 的联系（见用户手册）。调用 GT.connect 方法的合适位置通常是被测应用 Application 的 onCreate()或闪屏 Activity 的 onCreate()，不过在 demo 中为了方便演示，是通过点击一个按钮触发调用该方法。

具体调用代码在 demo 中的位置在类 MainActivity 中（可搜索 GT.connect 关键字找到代码具体位置），大体结构如下：

```
/*
 *  GT usage
 * 与GT控制台连接，同时注册输入输出参数
 */
GT.connect(getApplicationContext(), new AbsGTParaLoader() {

    @Override
    public void loadInParas(InParaManager inPara) {

    }

    @Override
    public void loadOutParas(OutParaManager outPara) {

    }
});
```

完成测试后，需要在被测工程中调用 GT.disconnect(Context)方法断开被测应用与 GT 的联系（见用户手册）。调用 GT. disconnect 方法的合适位置通常是被测应用退出的方法中。在本 demo 中，是通过点击一个按钮触发调用该方法。

具体调用代码在 demo 中的位置在类 MainActivity 中：

```
/*
 * disconnect按钮按下后，会立即与GT控制台断开连接，同时退出本应用。
 */
OnClickListener disconnect = new OnClickListener() {
    @Override
    public void onClick(View v) {
        // GT usage
        GT.disconnect(getApplicationContext());

        finish();
        System.exit(0);
    }
};
```

输入参数

相关 API

输入参数相关 API 如下(API 的具体说明详见 API 指引)：

API	说明

InParaManager. register	注册一个输入型参数
InParaManager. defaultInParasInAC	定义默认显示在悬浮窗的入参
InParaManager. defaultInParasInDisableArea	设置所有/部分 GT 定义的入参不生效
GT. setInPara	设置输入参数值
GT. getInPara	获取输入参数值

下面以 Demo 为例说明。

举例说明

使用输入参数有两个步骤，如下。

1. 在 GT SDK 初始化后注册所需的输入参数。
   注册输入参数的 API 是 InParaManager.register 方法。
2. 在需要使用输入参数的代码逻辑上获取输入参数。
   获取输入参数的 API 是一组重载方法，都叫 GT.getInPara，可以返回所有基本类型和字符串的输入参数值。

注册输入参数

- 在 GT.connect 方法中进行输入参数的注册：

```
/*
 *  GT usage
 * 与GT控制台连接，同时注册输入输出参数
 */
GT.connect(getApplicationContext(), new AbsGTParaLoader() {

    @Override
    public void loadInParas(InParaManager inPara) {
        /*
         * 注册输入参数，将在GT控制台上按顺序显示
         */
        inPara.register(并发线程数, "TN", "1", "2", "3");
        inPara.register(KeepAlive, "KA", "false", "true");
        inPara.register(读超时, "超时", "5000", "10000","1000");
        inPara.register(连接超时, "连超时", "5000", "10000","1000");

        // 定义默认显示在GT悬浮窗的3个输入参数
        inPara.defaultInParasInAC(并发线程数, KeepAlive, 读超时);
    }

    public void loadOutParas(OutParaManager outPara) {
});
```

（可选）注册的同时可以选择最多 3 个 APP 启动时默认在 GT 悬浮窗显示的输入型参数：

```
/*
 *  GT usage
 * 与GT控制台连接，同时注册输入输出参数
 */
GT.connect(getApplicationContext(), new AbsGTParaLoader() {

    @Override
    public void loadInParas(InParaManager inPara) {
        /*
         * 注册输入参数，将在GT控制台上按顺序显示
         */
        inPara.register(并发线程数, "TN", "1", "2", "3");
        inPara.register(KeepAlive, "KA", "false", "true");
        inPara.register(读超时, "超时", "5000", "10000","1000");
        inPara.register(连接超时, "连超时", "5000", "10000","1000");

        // 定义默认显示在GT悬浮窗的3个输入参数
        inPara.defaultInParasInAC(并发线程数, KeepAlive, 读超时);
    }

    public void loadOutParas(OutParaManager outPara) {
});
```

使用输入参数

在 Demo 工程中下载网络图片时使用并发线程数，并发线程数取 GT 上注册的值：（详情请在 Eclipse 中搜索“并发线程数”变量的使用场景）：

```
/*
 * GT usage
 * 使用输入参数"开启线程数"初始化线程池，默认线程数为代码逻辑的原值max_thread_num
 * 当输入参数设置失效时，默认值取max_thread_num就不会改变原有代码业务逻辑
 */
max_thread_num = GT.getInPara(并发线程数, max_thread_num, false);

ExecutorService t = Executors.newFixedThreadPool(max_thread_num);
```

建议

一个小技巧，`GT.getInPara` 方法的第二个参数默认值，大部分情况下建议使用业务逻辑中的原值，这样当输入参数设置失效时，默认值取原值就不会改变原有代码业务逻辑。如上面例子即是如此。

输出参数

相关 API

API	说明
OutParaManager. register	注册一个输出型参数
OutParaManager. defaultOutParasInAC	定义默认显示在悬浮窗的输出型参数
OutParaManager.defaultOutParasInDisableArea	设置所有/部分 GT 定义的入参不生效
GT. setOutPara	设置输出参数值
GT. getOutPara	获取输出参数值

下面以 Demo 为例说明。

举例说明

使用输出参数有两个步骤，如下。

1. 在 GT 初始化后注册所需的输出参数。
   注册输出型参数的 API 是 OutParaManager. register 方法。
2. 在需要的地方更改输出参数值。
   更改输出参数的 API 是 GT.setOutPara 方法。

注册输出参数

- Android 版 GT 在 GT.connect 方法中进行输出参数的注册：

```
/*
 *  GT usage
 * 与GT控制台连接，同时注册输入输出参数
 */
GT.connect(getApplicationContext(), new AbsGTParaLoader() {

    public void loadInParas(InParaManager inPara) {

    @Override
    public void loadOutParas(OutParaManager outPara) {
        /*
         * 注册输出参数，将在GT控制台上按顺序显示
         */
        outPara.register(下载耗时, "耗时");
        outPara.register(实际带宽, "带宽");
        outPara.register(singlePicSpeed, "SSPD");
        outPara.register(NumberOfDownloadedPics, "NDP");

        // 定义默认显示在GT悬浮窗的3个输出参数
        outPara.defaultOutParasInAC(下载耗时, 实际带宽, singlePicSpeed);
    }
});
```

（可选）注册的同时可以选择最多 3 个 APP 启动时默认在 GT 悬浮窗显示的输出型参数：

```
/*
 *  GT usage
 * 与GT控制台连接，同时注册输入输出参数
 */
GT.connect(getApplicationContext(), new AbsGTParaLoader() {

    public void loadInParas(InParaManager inPara) {

    @Override
    public void loadOutParas(OutParaManager outPara) {
        /*
         * 注册输出参数，将在GT控制台上按顺序显示
         */
        outPara.register(下载耗时, "耗时");
        outPara.register(实际带宽, "带宽");
        outPara.register(singlePicSpeed, "SSPD");
        outPara.register(NumberOfDownloadedPics, "NDP");

        // 定义默认显示在GT悬浮窗的3个输出参数
        outPara.defaultOutParasInAC(下载耗时, 实际带宽, singlePicSpeed);
    }
});
```

使用输出参数

在 Demo 工程中确认是否下载完成判断代码逻辑里统计下载耗时，实际带宽：

```
/*GT Usage start*/
GT.endTime(图片下载到UI展示, String.valueOf(picId));
if (completeSum.get() == img_Count && totalTime > 0) {
    long tempTime = (endTime - startTime)/1000000; // 纳秒转毫秒
    if (tempTime > 0) {
        double second = DoubleUtils.div(tempTime, 1000L, 3);
        GT.setOutPara(下载耗时, second + "s", true);

        long speed = totalSize.get() / tempTime;
        GT.setOutPara(实际带宽, speed + "k/s", false);
    }
}
GT.logI(UI处理图片, "按成功情况处理图片完成:" + picId);
/*GT Usage end*/
```

profiler 功能

相关 API

API	说明
GT. startTimeInThread	开始一次区分线程的耗时统计
GT. endTimeInThread	结束一次区分线程的耗时统计
GT. startTime	开始一次进程内部分区分线程的耗时统计
GT. endTime	结束一次进程内部分区分线程的耗时统计
GT.startTimeGlobal	开始一次可跨进程的耗时统计
GT.endTimeGlobal	结束一次可跨进程的耗时统计

下面以 Demo 为例说明。

举例说明

使用 profiler 进行耗时分析有两个步骤，如下。

1. 在需要开始计时的代码逻辑设置开始计时。对应区分线程的是 GT. startTimeInThread，不区分线程的是 GT. startTime，可跨进程的是 GT.startTimeGlobal。
2. 在需要结束计时的代码逻辑设置结束计时。对应区分线程的是 GT. endTimeInThread，

不区分线程的是 GT. endTime，可跨进程的是 GT.endTimeGlobal。

这里开始和结束调用的接口需要对应，若开始计时使用跨进程的 API，则结束计时也使用跨进程的 API；若开始计时使用区分线程的 API，则结束计时也需要使用区分线程的 API。

打开 profiler 功能

目前使用 profiler 前需要先手动开启 profiler 功能（详见使用手册相关章节）。如果实际使用中需要在被测应用启动后立即打开 profiler 功能，而手动开启来不及，那么可以先行手动打开 GT 应用，之后手动打开 profiler 功能，然后再启动被测应用即可。

使用 profiler 功能——线程内统计

这里要统计单张图片的下载时间，在网络访问时记录开始时间，网络数据接收完成时记录结束时间，开始到结束之间即为单张图片的下载时间。另外 Demo 里下载图片使用线程池有并发的可能，因此调用 GT. startTimeInThread 和 GT. endTimeInThread，用于区分线程。

线程开始是记录的开始时刻，下图中红色框中代码；网络访问数据接收结束是记录的结束时刻，下图中紫色框中代码：

```
public void run() {
    // GT usage
    GT.startTimeInThread(线程内统计, 图片下载);
    InputStream is = getInputStream(url);

    /*GT start*/
    GT.startTime(图片下载到UI展示, String.valueOf(picId));
    /*GT end*/

    try
    {
        // 下载
        final byte[] data = getBytesFromIS(is);

        /*GT usage start*/
        endTime = System.nanoTime();
        totalSize.addAndGet(data.length);
        GT.logI(速度统计, "length=" + data.length + "Byte totalSize=" + totalSize + "Byte");
        long et = GT.endTimeInThread(线程内统计, 图片下载);
        GT.logI(图片下载, "完成图片下载并渲染步骤（失败或成功），id:" + picId);
        et = et/1000000; // endTime取回来的是纳秒级，转成毫秒级别
        totalTime += et;
        GT.logI(速度统计, "pic" + picId + "=" + et + "ms" + " total=" + totalTime + "ms");

        if (et > 0) {
            long speed = data.length / et;
            GT.setOutPara(singlePicSpeed, speed + "k/s", true);
        }

        completeSum.incrementAndGet();
        /*GT usage end*/
```

注：

GT. startTimeInThread 和 GT. endTimeInThread 要和上面例子一样成对使用，道理

很简单，时间要有个开始时刻和结束时刻才能计算。每次统计值都会在 profiler 界面显示，详情请参考用户手册。

GT. endTimeInThread 的 long 型返回值是一次统计的时间间隔，单位是纳秒，即使在不启动 profiler 功能的情况，本返回值也是有效的，只是不会在 GT 控制台的 profiler 模块记录。该返回值在异常情况下会是-1。

使用 profiler 功能——应用内跨线程统计

Demo 中图片下载完成后到 UI 展示的时间统计，其统计开始时刻是下载并解析图片生成 bitmap 完成后，但其结束时刻应是 Android 的 UI 线程中，这种跨线程的应该用全局统计，并且要区别不同的图片：

统计起点是下载线程的开始时刻：

```
// 独立线程解码图片
decodePool.execute(new Runnable() {

    @Override
    public void run() {
        try{
            bitmaps[picId] = BitmapFactory.decodeByteArray(data, 0, data.length);
            Message msg = handler.obtainMessage();
            Bundle bundle = new Bundle();
            bundle.putInt("picId", picId);
            msg.setData(bundle);
            msg.what = LOAD_IMG_OK;

            /*GT usage start*/
            GT.startTime(下载完成后到UI展示, String.valueOf(picId));
            /*GT usage end*/

            msg.sendToTarget();
        }
        catch(Exception e)
        {
            GT.logE(图片下载, "图片解码失败:" + picId);
            e.printStackTrace();
        }

    }});
```

其结束时刻在 UI 线程的 Handler 中：

```
gridview.setAdapter(saImageItems);
/*GT usage start*/
GT.endTime(下载完成后到UI展示, String.valueOf(picId));
```

打印日志

相关 API

API	说明
GT. logD	打印日志，DEBUG 级别
GT. logI	打印日志，INFO 级别
GT. logW	打印日志，WARNING 级别
GT. logE	打印日志，ERROR 级别

下面以 Demo 为例举例说明。

举例说明

打印日志功能类似 logcat，有区分 DEBUG，INFO，WARNING 和 ERROR 四个级别，用户根据实际情况调用对应级别的接口。

具体例子在 demo 中随处可见，搜索“GT. Log”即可。