转载请注明链接: https://blog.csdn.net/feather_wch/article/details/88087170

RePlugin的使用

版本号:2019-03-13(22:00)

接入RePlugin(9题)

1-添加Gradle依赖

1、根目录的build.gradle

```
buildscript {
    dependencies {
        classpath 'com.qihoo360.replugin:replugin-host-gradle:2.2.4'
        ...
    }
}
```

2-添加Library依赖

2、app的build.gradle

> 1. 应用 `replugin-host-gradle` 插件
> 2. 添加依赖 `replugin-host-lib`
> 3. `applicationId` 必须要使用完整包名，且必须在 `apply plugin: 'replugin-host-gradle'` 的上面。避免报错 `Failed to find provider info for com.ss.android.auto.loader.p.main` 。
> 4. 使用配置项(可选)，是否使用AppCompat，等等。

```
android {
    // ATTENTION!!! Must CONFIG this to accord with Gradle's standard, and avoid some error
    defaultConfig {
        applicationId "com.hao.iday"
        ...
    }
    ...
}

// ATTENTION!!! Must be PLACED AFTER "android{}" to read the applicationId
apply plugin: 'replugin-host-gradle'

/**
 * 配置项均为可选配置，默认无需添加
 * 更多可选配置项参见replugin-host-gradle的RepluginConfig类
 * 可更改配置项参见 自动生成RePluginHostConfig.java
 */
repluginHostConfig {
    /**
     * 是否使用 AppCompat 库
     * 不需要个性化配置时，无需添加
     */
    useAppCompat = true
    /**
     * 背景不透明的坑的数量
     * 不需要个性化配置时，无需添加
     */
    countNotTranslucentStandard = 6
    countNotTranslucentSingleTop = 2
    countNotTranslucentSingleTask = 3
    countNotTranslucentSingleInstance = 2
}

dependencies {
    compile 'com.qihoo360.replugin:replugin-host-lib:2.2.4'
    ...
}
```

配置项

3、配置项都是可选配置，按需选择。

> 1. 更多可选配置项参见replugin-host-gradle的RepluginConfig类
> 2. 可更改配置项参见 自动生成RePluginHostConfig.java

```
repluginHostConfig {
    // xxx
}
```

No signature of method: com.android.build.gradle.internal.scope.VariantScopeImpl.getMergeAssetsTask() is applicable for argument types: () values: []

4、报错:

```
No signature of method: com.android.build.gradle.internal.scope.VariantScopeImpl.getMergeAssetsTask() is applicable for argument types: () valu
```

> 1. `2.2.4` 强制使用4.6的gradle,现在插件还没兼容4.6
> 2. 官方还未解决该故障。只能采用网友的已经修复的库。

```
maven{
    url "https://dl.bintray.com/soli/maven"
}
// root, build.gradle
classpath 'com.qihoo360.replugin:replugin-host-gradle:2.3.1'
// app, build.gradle
implementation 'com.qihoo360.replugin:replugin-host-lib:2.3.1'
```

IllegalStateException: You need to use a Theme.AppCompat theme (or descendant) with this activity.

5、报错 `IllegalStateException: You need to use a Theme.AppCompat theme (or descendant) with this activity.`

> 1. 需要在配置项中开启AppCompat

```
repluginHostConfig {
    useAppCompat = true
}
```

3-配置Application

继承式配置

6、继承式配置Application

```
public class MainApplication extends RePluginApplication {
}
```

> AndroidManifest.xml

```
<application
      android:name=".MainApplication"
      ... />
```

非继承式配置

7、非继承式配置Application

> 1. 所有方法必须在UI线程来“同步”调用。切勿放到工作线程，或者通过post方法来执行
> 2. 所有方法必须一一对应，例如 RePlugin.App.attachBaseContext 方法只在Application.attachBaseContext中调用
> 3. 请将RePlugin.App的调用方法，放在“仅次于super.xxx()”方法的后面

```
public class MainApplication extends Application {

    @Override
    protected void attachBaseContext(Context base) {
        super.attachBaseContext(base);

        RePlugin.App.attachBaseContext(this);
        ....
    }

    @Override
    public void onCreate() {
        super.onCreate();

        RePlugin.App.onCreate();
        ....
    }

    @Override
    public void onLowMemory() {
        super.onLowMemory();

        RePlugin.App.onLowMemory();
        ....
    }

    @Override
    public void onTrimMemory(int level) {
        super.onTrimMemory(level);

        RePlugin.App.onTrimMemory(level);
        ....
    }

    @Override
    public void onConfigurationChanged(Configuration config) {
        super.onConfigurationChanged(config);

        RePlugin.App.onConfigurationChanged(config);
        ....
    }
}
```

java.lang.ExceptionInInitializerError

8、报错 `java.lang.ExceptionInInitializerError`

> 需要换成非继承的形式

4-插件App接入

9、在build.gradle中进行配置

> 1-根目录-build.gradle中接入

```
buildscript {
    dependencies {
        classpath 'com.qihoo360.replugin:replugin-plugin-gradle:2.2.4'
        ...
    }
}
```

> 2-app的build.gradle中配置

```
apply plugin: 'replugin-plugin-gradle'

dependencies {
    compile 'com.qihoo360.replugin:replugin-plugin-lib:2.2.4'
    ...
}
```

安装插件(50题)

1、无论是内置，还是外置插件，不是所有的APK都能作为 RePlugin 的插件并安装进来的。

1. 必须要严格按照《插件接入指南》中所述完成接入，其编译出的APK才能成为插件，且这个APK同时也可以被安装到设备中。

外置插件

2、什么是外置插件?

是指可通过“下载”、“放入SD卡”等方式来安装并运行的插件。

安装插件

3、外置插件的安装

1. 只需使用 RePlugin.install 方法，传递一个“APK路径”即可。

```
RePlugin.install("/sdcard/exam.apk");
```

1. 无论安装还是升级，都会将“源文件” 移动 到 插件的安装路径 上，这样可大幅度节省安装和升级时间，但显然的，“源文件”也就会消失

插件下载

4、如果插件需要下载，需要覆写 `RePluginCallbacks.onPluginNotExistsForActivity` 方法，并在此打开自定义的下载页面并控制其逻辑

升级插件

5、外置插件的升级

1. 方法和“安装”一致
2. 如果插件正在运行，则不会立即升级，而是“缓存”起来。直到所有“正在使用插件”的进程结束并重启后才会生效
3. 不支持“插件降级”，但可以“同版本覆盖”

```
RePlugin.install("/sdcard/exam_new.apk");
```

6、升级可以提前进行预加载

```
PluginInfo pi = RePlugin.install("/sdcard/exam_new.apk");
if (pi != null) {
        RePlugin.preload(pi);
}
```

7、如果插件正在运行，则会有两种场景：

1. 若是遇到严重问题，需要“强制升级”，则应立即提示用户，待同意后则重启进程
2. 通常情况下，建议在“锁定屏幕”后重启进程，让其在后台生效

安装/升级失败

8、安装或者升级失败的原因(返回值为null，表示失败)?

1. **是否开启了“签名校验”功能且签名不在“白名单”之中**?
   - 通常在Logcat中会出现“verifySignature: invalid cert: ”。如是，则请参考“安全与签名校验”一节，了解如何将签名加白，或关闭签名校验功能（默认为关闭）
2. **是否将replugin-host-lib升级到2.1.4及以上**?
   - 在2.1.3及之前版本，若没有填写“meta-data”，则可能导致安装失败，返回值为null。我们在 2.1.4 版本中已经修复了此问题（卫士和其它App的所有插件都填写了meta-data，所以问题没出现）
3. **APK安装包是否有问题**?
   - 请将“插件APK”直接安装到设备上（而非作为插件）试试。如果在设备中安装失败，则插件安装也一定是失败的。
4. **是否没有SD卡的读写权限**?
   - 如果您的插件APK放到了SD卡上，则请务必确保主程序中拥有SD卡权限（主程序Manifest要声明，且ROM允许），否则会出现权限问题，当然，放入应用的files目录则不受影响。
5. **设备内部存储空间是否不足**?
   - 通常出现此问题时其Logcat会出现“copyOrMoveApk: Copy/Move Failed”的警告。如是，则需要告知用户去清理手机。

卸载插件

9、卸载插件

1. 使用 RePlugin.uninstall 方法。只需传递一个“插件名”。对于正在运行的插件，只是记录卸载请求，后续才会进行卸载。
2. 内置插件无法卸载
3. 插件正在运行时： 1. 提示用户进行重启app. 2. 锁定屏幕后后台卸载

```
RePlugin.uninstall("exam");
```

内置插件

10、内置插件是什么?

> 1. 内置插件是指可以“随着主程序发版”而下发的插件，通常这个插件会放到主程序的Assets目录下。
> 2. 针对内置插件而言，开发者可无需调用安装方法，由RePlugin来“按需安装”。
> 3. “内置插件”是可以被“升级”的。升级后的插件等同于“外置插件”

添加内置插件

11、添加内置插件

> 只需两步即可：
>
> 1. 将APK改名为：[插件名].jar
> 2. 放入主程序的assets/plugins目录

12、内置插件的处理原理

> 1. 当编译主程序时，“动态编译方案”会自动在 `assets目录` 下生成一个名叫“plugins-builtin.json”文件，记录了其内置插件的主要信息，方便运行时直接获取。
> 2. 必须改成“[插件名].jar”后，才能被 `RePlugin-Host-Gradle` 识别，进而成为“内置插件”。
> 3. [插件名]可以是“包名”，也可以是“插件别名”。见《插件的信息》中“插件命名”

升级

13、内置插件的升级

> 内置插件的升级分为两种情况：随主程序升级、通过install方法升级
>
> 1. 随主程序升级：
>   - 当用户升级了带“新版本内置插件”的主程序时，则RePlugin会在使用插件前先做升级
> 2. 通过install方法升级：
>   - 通过 `RePlugin.install` 方法升级.

预加载

14、插件的预加载是什么?

> 1. 就是将插件的dex“提前做释放”，并将Dex缓存到内存中.
> 2. 在下次启动插件时，可无需走dex2oat过程，速度会快很多。

15、预加载不会做下列事情：

> 1. 不会“启动插件”
> 2. 不会加载其Application对象
> 3. 不会打开Activity和其它组件等。

16、预加载当前安装的插件

> 1. 直接预加载当前安装的插件即可。
> 2. RePlugin.preload(pluginName):

```
RePlugin.preload("exam");
```

17、预加载新安装的插件

> 1. 此场景主要用于“后台升级某个插件”。
> 2. 如果此插件“正在被使用”，则必须借助 RePlugin.install 方法返回的新插件信息，来做预加载。
> 3. 可使用 RePlugin.preload(PluginInfo)，例如：

```
PluginInfo pi = RePlugin.install("/sdcard/exam_new.apk");
if (pi != null) {
        RePlugin.preload(pi);
}
```

18、预加载需要在工作线程中进行，不能在UI线程中进行.

> 1. RePlugin.preload 方法的调用放到“工作线程”中进行。
> 2. 由于此方法是“同步”的，所以直接在UI线程中调用时，可能会卡住，甚至导致ANR问题。

19、正在预加载的插件，如果加载该插件会导致卡顿

> 1. 如果“正在preload”某插件，则无论在哪个进程和线程，在过程中加载这个插件时，可能会出现卡顿，原因：
>   - 为了安全起见，做了 进程锁 。
> 2. 应该在preload做完后再打开此插件。

插件的运行

20、插件运行的场：

> 1. 打开插件的四大组件
> 2. 获取插件的PackageInfo/Context/ClassLoader等
> 3. 预加载（preload）
> 4. 使用插件Binder

21、如何判断插件是否在运行?

> 1. 使用 `RePlugin.isPluginRunning` 方法判断。

安全与签名校验

22、动态加载dex方案可能存在的安全问题

> 1. 由于没有对外来的Dex和Apk做“校验”导致。
> 2. 一旦不做校验，则不排除恶意人会劫持DNS或网络，并通过网络来下发恶意插件.

打开签名校验开关

23、第一步：打开签名校验开关

> 1-继承RePluginApplication，则在创建 RePluginConfig 时调用其 `setVerifySign(!BuildConfig.DEBUG)` 即可。

```
public class BaseApplication extends RePluginApplication {
    // xxx

    @Override
    protected RePluginConfig createConfig() {
        RePluginConfig config = new RePluginConfig();
        // 若为Debug环境下则无需校验签名，只有Release才会校验。
        config.setVerifySign(!BuildConfig.DEBUG);
        return config;
    }
}
```

> 2-非继承式Application，则需要在调用 RePlugin.App.attachBaseContext() 的地方，进行处理。

```
public class BaseApplication extends RePluginApplication {
    @Override
    protected void attachBaseContext(Context base) {
        super.attachBaseContext(base);
        RePluginConfig config = new RePluginConfig();
        config.setVerifySign(!BuildConfig.DEBUG);
        RePlugin.App.attachBaseContext(base, config);
    }
}
```

24、第二步：加入合法签名

> 1. 打开开关后，应该将“合法的签名”加入到RePlugin的“白名单”中：

```
// 其中，其参数传递的是签名证书的MD5，必须去掉“：”。且不要传递SHA1或其它非签名MD5内容.
RePlugin.addCertSignature("379C790B7B726B51AC58E8FCBCFEB586");
```

25、出于性能考虑，内置插件无需做“签名校验”，仅“外置插件”会做.

> 一旦签名校验失败，则会在Logcat中提示 “`verifySignature: invalid cert`” ，且install方法返回null。

26、若在调用 install 方法前就已对APK做了校验，则可关闭，以避免重复校验。

> 不要使用和“主程序”一样的签名，而是单独创建一个。

插件管理进程

27、插件管理进程

> 1. 用于支持独特的“跨进程安全通讯”（见IPC类）以及复杂的插件管理机制
> 2. 为保证插件能统一由“一个中心”来管理，提高每个进程的启动、运行速度
> 3. 所有插件、进程等信息均在 `插件管理进程` 中被记录，各进程均从此中获取、修改等。 `RePlugin的这种做法有点像AMS`
> 4. 无需像其它框架一样，要求“每个进程各自初始化信息”。

28、目前有两种进程可以作为“插件管理进程”：

> 1. 常驻进程
> 2. 主进程

常驻进程

29、默认以“常驻进程”作为“插件管理进程”

> 1. 在RePlugin 2.1.7及以前版本，这是唯一的方式。
> 2. RePlugin默认的“常驻进程”名为“ `:GuardService` ”， `通常在后台运行，存活时间相对较久` 。
> 3. 这样的最大好处是：应用“冷启动”的概率被明显的降低， `大部分都变成了“热启动”，速度更快` 。

30、适合作为常驻进程的场景包括：

> 1. 以后台服务为主要业务的应用，例如：手机安全类、健身和健康监控类、OS内应用等
> 2. 需要有常驻通知栏的应用，例如：音乐类、清理类等
> 3. 需保持常连接（例如Push等）的应用，如：即时通讯类、泛社交类等
> 4. 目前市面上多数应用都集成了推送功能（例如友盟、极光推送），常驻进程可以挂载在那里。

31、以“常驻进程”作为“插件管理进程”的优点

> - 这是结合“常驻进程”长期存活的特点而展开的：
>   1. 各进程启动时，插件信息的获取速度会更快（因直接通过Binder从常驻进程获取）
>   2. 只要常驻进程不死，其它进程杀掉重启后，仍能快速启动（热启动，而非“冷启动”）
>   3. 如果做得好的话，甚至可以做到“0秒启动”，如360手机卫士。

38、以“常驻进程”作为“插件管理进程”的缺点：

> 1. 若应用为“冷启动”（无任何进程时启动），则需要同时拉起“常驻进程”，时间可能有所延长
> 2. 若应用对“进程”数量比较敏感，则此模式会无形中“多一个进程”

主进程

32、以“主进程”作为“插件管理进程”

> 1. 自RePlugin 2.2.0开始，主进程也可以作为“插件管理进程”。
> 2. 这样做的最大好处是：应用启动时，可以做到“只有一个进程”（注意，这不代表你不能开启其它插件进程，这里只是说没有“常驻进程”了而已）。
> 3. 当然，代价是享受不到“常驻进程”时的一些好处。

33、“主进程”的适用场景、

> 只要是不符合上述“常驻进程”中所涉及到的场景的，本模式都适合。

34、以“主进程”作为“插件管理进程”的优点

> 1. 无需额外启动任何进程.
> 2. 应用冷启动的时间会短一些，因为无需再拉起额外进程

35、以“主进程”作为“插件管理进程”的缺点

> 1. “冷启动”的频率会更高，更容易被系统回收
> 2. 再次启动的速度略慢于“热启动”

如何使用?

36、设置为“常驻进程”

> 若不设置，则默认是以“常驻进程”作为“插件管理进程”。

37、切换到以“主进程”作为“插件管理进程”

> 1. 需要在宿主的 `app/build.gradle` 中添加下列内容:

```
apply plugin: 'replugin-host-gradle'
repluginHostConfig {
    // ... 其它RePlugin参数

    // 设置为“不需要常驻进程”
    persistentEnable = false
}
```

插件的目录结构

38、为了保证稳定性，会把经过验证的插件放到一个特殊的目录下，以防止“源文件”被删除后的一些问题。

> 为简化起见，将“/data/data/[你的主程序包名]”统一简化成“主程序路径”

39、外置插件（现在只有这一种目录）：

> 1. APK存放路径：主程序路径/app_p_a
> 2. Dex存放路径：主程序路径/app_p_od
> 3. Native存放路径：主程序路径/app_p_n
> 4. 插件数据存放路径：主程序路径/app_plugin_v3_data

文件的组织形式

40、外置插件：为了方便使用，插件会有一个JSON文件，用来记录所有已安装插件的信息。

> 位于“主程序路径/app_p_a/p.l”中。

41、内置插件：内置插件的JSON文件只存放于主程序“assets/plugins-builtin.json”文件下。每次会从那里获取信息。

插件命名

42、RePlugin的插件可以有两种名字，分别是：插件包名、插件别名。

> 1. 插件包名：顾名思义，就是插件的PackageName
> 2. 插件别名：为了“精简包名”而设计的别名

插件包名

43、插件包名可以任意起名，不受限制。

> 若APK既作为单品，又作为插件，则建议分成两个包名，且Provider的Authority也建议改名，这样可针对不同的场景（插件还是单品）来做不同的处理。

插件别名

44、插件别名如何使用

> 要声明插件别名，需要在插件的 `AndroidManifest.xml` 中声明以下 `Meta-data`：

```
    <meta-data
           android:name="com.qihoo360.plugin.name"
           android:value="[你的插件别名]" />
```

45、针对内置插件而言, 可以不填写插件别名

> 1. 若不填写插件别名，则会将内置插件的“文件名”作为其插件名
> 2. 其优先级为：Meta-data声明的 > 文件名

46、针对外置插件而言，必须要指明插件包名

> 1. 若不填写插件别名，则只能允许使用“插件包名”

插件版本

VersionCode

47、插件的VersionCode(版本号)

1. 建议分为三位，如121。Integer类型不要查过32位最大值即可
2. 第一位大版本
3. 第二位功能版本: 新增了功能，优化了功能
4. 第三位修复版本: BUG需要修复等

协议版本号

48、插件的协议版本号的作用

1. 区分新旧插件，值低于协议版本号的插件不会被使用，防止出错

```
    <meta-data
          android:name="com.qihoo360.plugin.version.ver"
          android:value="[你的插件协议版本号]" />
```

插件框架版本号

49、插件的框架协议号的作用

1. 让该插件不能跑在新版app上防止出错

```
   <meta-data
          android:name="com.qihoo360.framework.ver"
          android:value="[你的框架版本号]" />
```

插件信息的获取

50、插件信息的获取方法

1. 获取任意正在运行的插件信息: `RePlugin.getPlugin()`，返回null表示未安装。
2. 获取所有已安装插件的信息列表: `RePlugin.getPluginInfoList()`
3. 获取安装成功后的信息(新插件的信息): `RePlugin.install()`
4. `pluginInfo.getPendingUpdate`：获取待更新的插件信息

```
// 1、获取到正在运行的插件信息
PluginInfo pi = RePlugin.getPluginInfo("exam");
if (pi != null) {
  // 2、获取到“新插件的信息(带更新的插件)”
        PluginInfo newPi = pi.getPendingUpdate();
        if (newPi != null) {
    // 还未更新
                ...
        } else {
                // 没有
                ...
        }
}
```

打开插件的Activity(31题)

多进程

1、RePlugin支持多个进程的分配，常见于下列的场景

1. 在单独进程中运行一个Service
   * 如“下载”服务等
2. 在“常驻进程”中运行长期工作的Service
   * 如“Push”服务等
3. 隔离“过于消耗资源”的Activity
   * 如“换肤主题”页面等
4. 对“非常复杂，不排除会出问题”的插件做“隔离”
   * 防止进程崩溃时，对主进程造成冲击
5. 双进程模型，可将大部分初始化操作放在常驻进程, 其它进程直接“获取”即可
   * 如用户帐号的加解密、文件IO操作等，大幅度提高使用效率

2、多进程的副作用主要有(“所有Android应用”都存在的副作用，而非RePlugin特有)

1. 首次开启进程会有性能消耗，打开Activity可能会有“短暂的黑屏”或“无响应”（根据Theme）。

    1. 大概在20ms~100ms不等（和Application类的复杂程度有关）
    2. 原因：系统需要Zygote这个新进程，然后和AMS交互，最后调用Application, 耗时。
2. 跨进程的“交互”只能依靠Binder、Socket、文件等操作来解决，不支持反射。
    1. 尤其是Binder，双方通信时需要写一些AIDL
    2. 从“应用的内存占用”来说，每多运行一个进程，则会多出一些应用内存的占用。一个空Application，无论单品还是插件，每增加一个进程大概多占用5M（Dalvik）~20M不等（ART）

分配策略

静态分配

3、静态分配策略:

1. 开发者在Meta-data中自行声明并决定这些“插件进程”应该跑在哪个“坑位进程”内。

4、进程坑位有限

1. 如果有的插件，具有十多个自定义进程（如“桌面”插件）
2. 则很可能出现“进程分配坑位不足”的情况

5、实例

```
<meta-data
           android:name="process_map"
           android:value="[
           {'from':'com.qihoo360.launcher:wff', 'to':'$ui'},
           {'from':'android.process.acore', 'to':'$p0'},
           {'from':'com.qihoo360.accounts', 'to':'$p1'},
           {'from':'com.qihoo360.launcher:livewallpaper', 'to':'$p2'}
           ]" />
```

1. from：原来声明的进程名是什么。例如有个Activity，其进程名声明为“com.qihoo360.launcher:wff”
2. to：要映射到的进程名，必须以“$”开头，表示“特殊进程”
    1. $ui：映射到UI进程
    2. $p0：映射到进程坑位0进程
    3. $p1：映射到进程坑位1进程
    4. 以此类推
3. 没有配置的进程，默认跑在主进程中。

动态分配

6、动态分配策略：

1. 如果没有配置“静态分配”的坑位，则默认采用“动态分配”方案

7、动态分配方案的特点是：

1. 无需声明Meta-data。自定义进程启动时，RePlugin会自动按顺序为其分配进程坑位
2. 当坑位不足时，无需开发者关心，RePlugin会自动处理进程情况

8、RePlugin 2.2.0 开始已完美支持“动态分配”进程。

如何使用组件

插件内组件

9、打开一个Activity

```
Intent intent = new Intent(v.getContext(), ThemeDialogActivity.class);
context.startActivity(intent);
```

10、打开一个Service

```
Intent intent = new Intent(v.getContext(), PluginDemoService1.class);
intent.setAction("action1");
context.startService(intent);
```

11、使用ContentProvider

```
Uri uri = Uri.parse("content://com.qihoo360.replugin.sample.demo1.provider2/test");

ContentValues cv = new ContentValues();
cv.put("address", "beijing");

Uri urii = context.getContentResolver().insert(uri, cv);
```

12、使用广播

```
Intent intent = new Intent();
intent.setAction("com.qihoo360.repluginapp.replugin.receiver.ACTION1");
intent.putExtra("name", "jerry");
context.sendBroadcast(intent);
```

插件外组件

13、插件外组件如何打开?

> 和“插件内”的基本一致，唯一不同的是：ComponentName为插件名（可以是包名，也可以是别名），也可以是Action。

```
// 方法1（最“单品”）
Intent intent = new Intent();
intent.setComponent(new ComponentName("demo2",
    "com.qihoo360.replugin.sample.demo2.databinding.DataBindingActivity"));
context.startActivity(intent);

// 方法2（快速创建Intent）
Intent intent = RePlugin.createIntent("demo2",
    "com.qihoo360.replugin.sample.demo2.databinding.DataBindingActivity");
context.startActivity(intent);

// 方法3（一行搞定）
RePlugin.startActivity(v.getContext(), new Intent(), "demo2",
    "com.qihoo360.replugin.sample.demo2.databinding.DataBindingActivity");
```

插件调用主程序组件

14、插件若要使用主程序的组件。唯一的区别是，需要传递“字符串”。

```
Intent intent = new Intent();
intent.setComponent(new ComponentName("com.qihoo360.replugin.sample.host", "com.qihoo360.replugin.sample.host.MainActivity"));
context.startActivity(intent);
```

15、插件获取主程序Context

```
Context hostContext = RePlugin.getHostContext();
```

> 获取其它内容（如ClassLoader等）也如法炮制，可直接调用RePlugin类中的相应方法即可。

主程序调用插件组件

16、主程序调用插件组件的做法中Activity、Service、ContentProvider保持一致

17、主程序调用插件组件的BroadcastReceiver

18、打开插件的Activity

> 1. 调用 `RePlugin.startActivity()` 方法。例如：

```
RePlugin.startActivity(MainActivity.this, RePlugin.createIntent("demo1",
    "com.qihoo360.replugin.sample.demo1.MainActivity"));
```

19、获取插件的Context

> 1. 调用 RePlugin.fetchContext() 方法。例如：

```
Context examContext = RePlugin.fetchContext("exam");
```

> 获取其它内容（如ClassLoader等）也如法炮制，可直接调用RePlugin类中的相应方法即可。

20、启动、绑定插件的Service

> 1. 可以使用 `PluginServiceClient`类 中的相应方法来操作。
> 2. 例如，若您想“绑定”一个服务，则可以：

```
PluginServiceClient.bindService(RePlugin.createIntent(
    "exam", "AbcService"), mServiceConn);
```

21、使用插件的Content-Provider

> 1. 使用 PluginProviderClient 类中的方法即可操作Provider

```
PluginProviderClient.query(xxx);
```

如何使用SO库

22、RePlugin 支持使用SO库，无论是放置SO的方法，或者使用SO库的办法，都和独立app一致。

23、插件支持“无缝”使用宿主的SO

> 1. 且作为开发者而言，无需关心SO是放在宿主还是插件中
> 2. 均只需要调用 Android API 中提供的方法即可实现。

24、32位/64位指令集问题

> 1. 32位和64位的SO库不能混用。经常碰见“ `UnsatisfiedLinkError` ”。
> 2. 位数不能混用（32和64位，他俩指令集完全不同）
> 3. 但同一位数下的指令集是向后兼容的，
> * 例如放置了armeabi-v7a和armeabi，是可以混用的，不会出现问题。

25、Android指令级的判断

> 1.Android在安装一个应用时，会根据主程序APK中的SO指令集信息，来判断该应用是否支持“64位”。如果您的手机支持64位指令集，且满足下列条件之一：

26、Android如何判断应用是否支持“64位”?

> 如果手机支持64位指令集，且满足下列条件之一：
> 1. 宿主内没有放置任何SO
> 1.放置了64位指令集的SO库（无论32位是否放置）

27、“64位模式”和“32位兼容模式”

> 1. 两者的Zygote进程不同（分别为Zygote和Zygote64）
> 2. 导致所处的运行时环境不同。若强行加载SO，则会出现指令集混用的异常。

同时支持32位和64位

28、如何同时支持32位和64位?

> 1. 无论是主程序还是插件，务必同时放入32位和64位的SO库。

只支持32位

29、如何只支持32位

> 1. 宿主务必只放入32位SO库
> - 若宿主没有SO库，则“务必放入一个空的32位SO库”
> 2. 对插件而言，64位的SO将不会生效，安装插件时也不会释放（因为主程序只在32位上运行）

30、如果应用和插件目前都没有SO库，建议按照“只支持32位”的做法来处理。

> 因为将来一旦有插件带SO，则不至于出现一些问题。

31、不推要求“只支持64位”

> 会导致部分机型无法使用

重要的工具(3题)

RePluginCallbacks

1、RePluginCallbacks的作用

> 1. 用来生成宿主和插件的ClassLoader
> 2. 要打开的插件不存在时，回调
> 3. 要打开的插件文件过大时，回调

RePluginEventCallbacks

2、RePluginEventCallbacks的作用

> 1. 插件化框架对外事件回调接口集
> 2. 可回调的接口包括: 安装插件成功、失败、启动插件Activity等

RePluginConfig

3、RePluginConfig的作用

> 1. 主要用于对Replugin的一些初始化设置，包括:
>   1. 是否开启插件的签名校检
>   2. 当插件中使用某个类不存在时是否使用宿主中的类
> 2. 内部还保持了对RepluginCallbacks和RepluginEventCallbacks的引用。

问题汇总

参考资料

1. RePlugin的github
2. 插件接入指南
3. 插件的管理
4. RePlugin原理
5. 详细教程