主题：RePlugin框架实现原理和最佳实践

团队：手机卫士
讲师：刘存栋

内容

1.RePlugin 介绍

2.ClassLoader处理

3.Context和Resources处理

4.插件加载

5.内部通信框架

6.Activity插件化流程

7.最佳实践

1.RePlugin 介绍

开源以后

6月底开源

发布10+个版本

解决260+个issue

447次commit

开发者群1000+成员

新闻类、视频类、金融类、电商类，办公类、工具类、社交类..

5+个非360系日活千万级产品接入

开发者平台

1.1 RePlugin介绍

Host Project

replugin-host-gradle

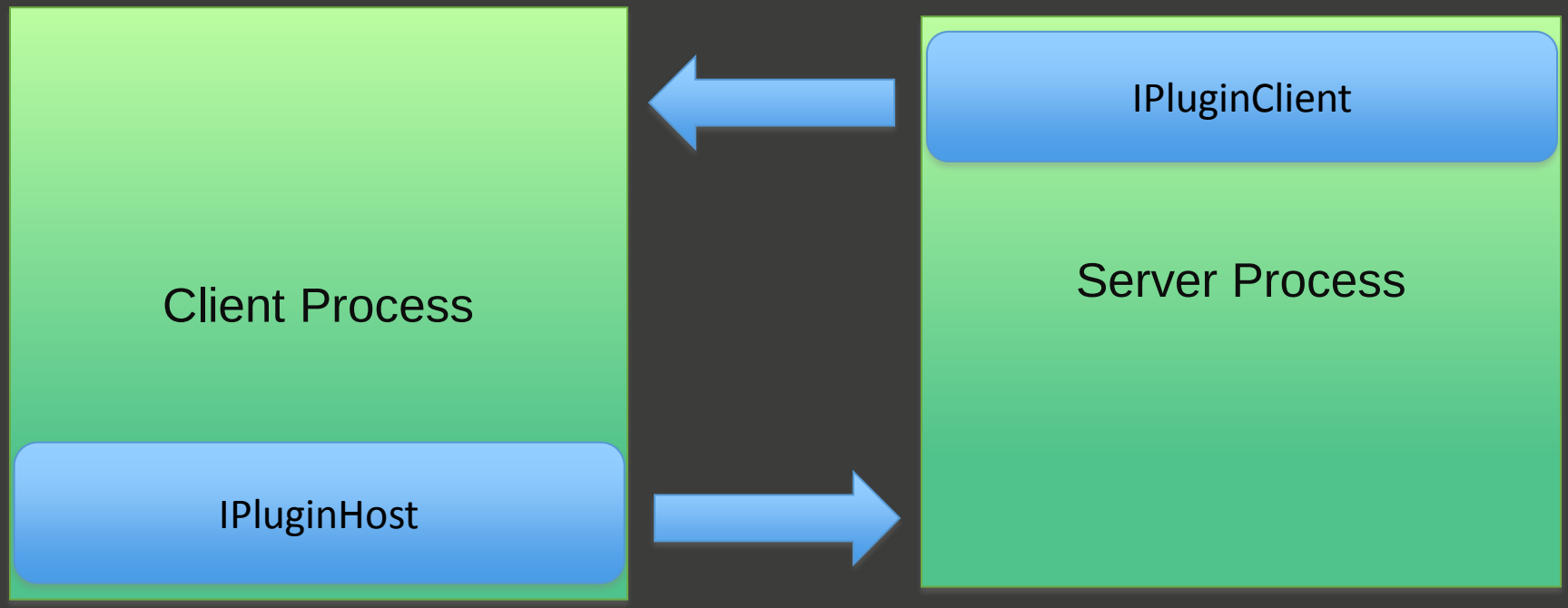
replugin-host-library

Plugin Project

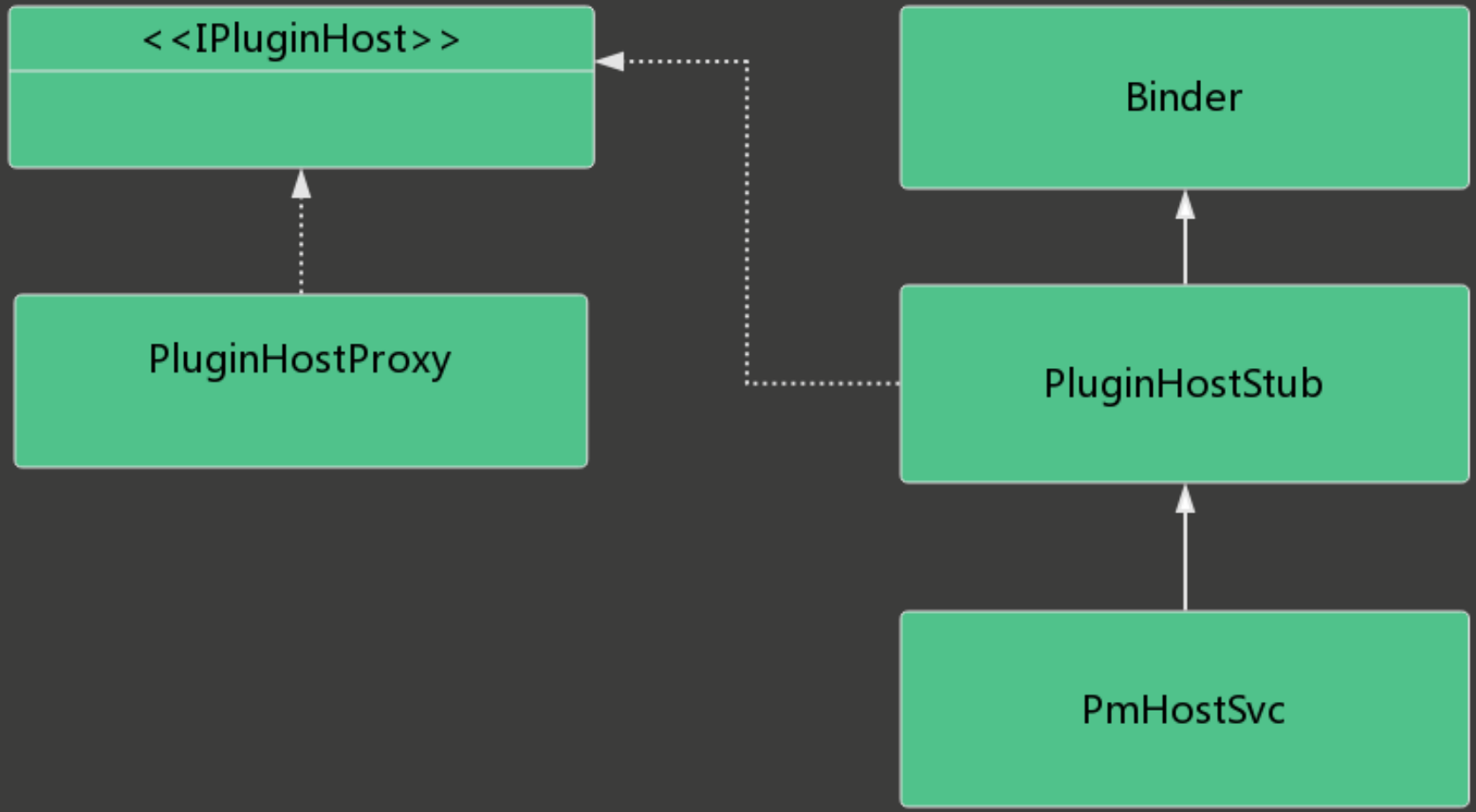
replugin-plugin-gradle

replugin-plugin-library

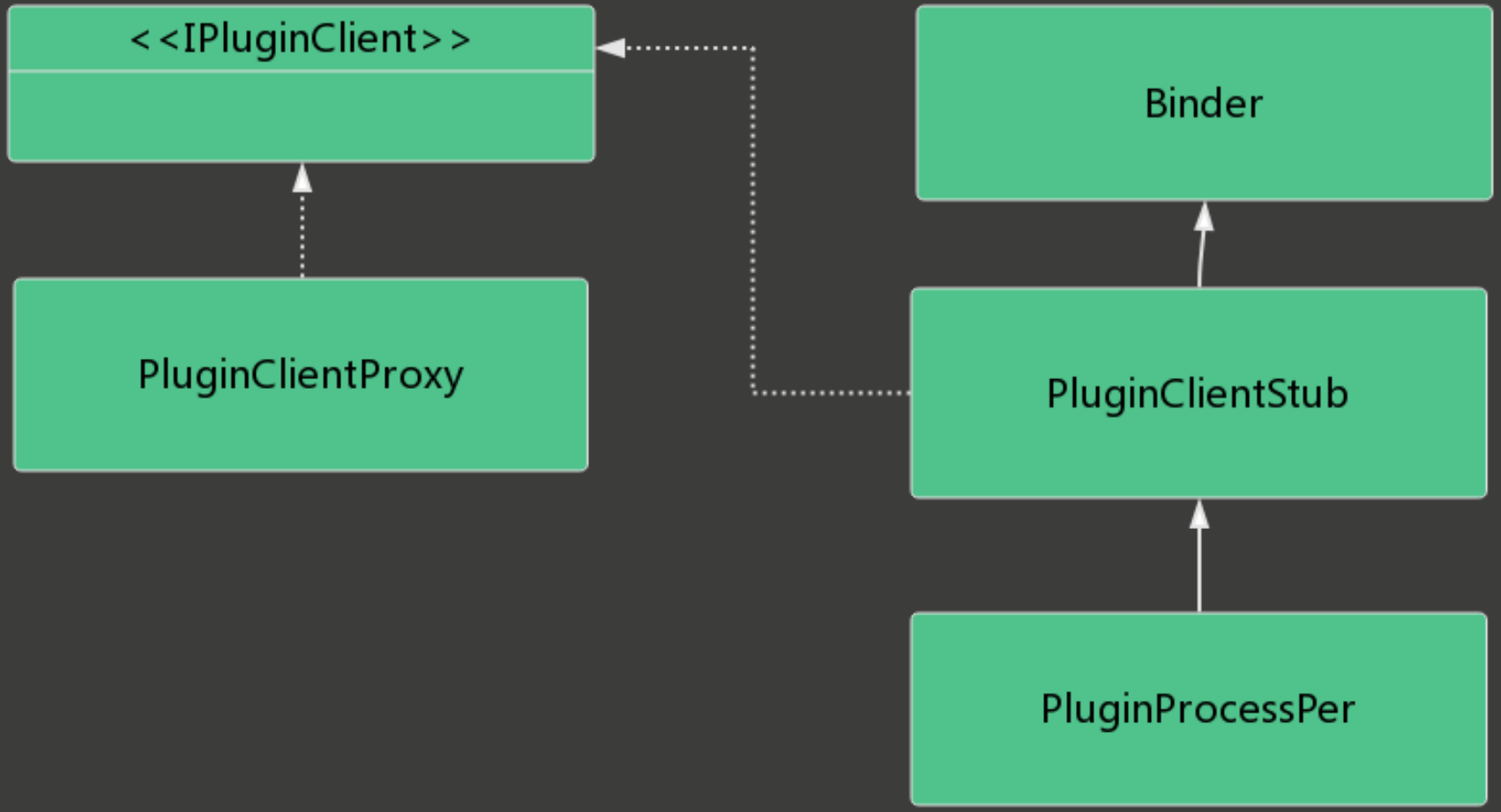
1.2 默认双进程架构


IPluginClient
Client Process
Server Process
IPluginHost

1.3 IPluginHost

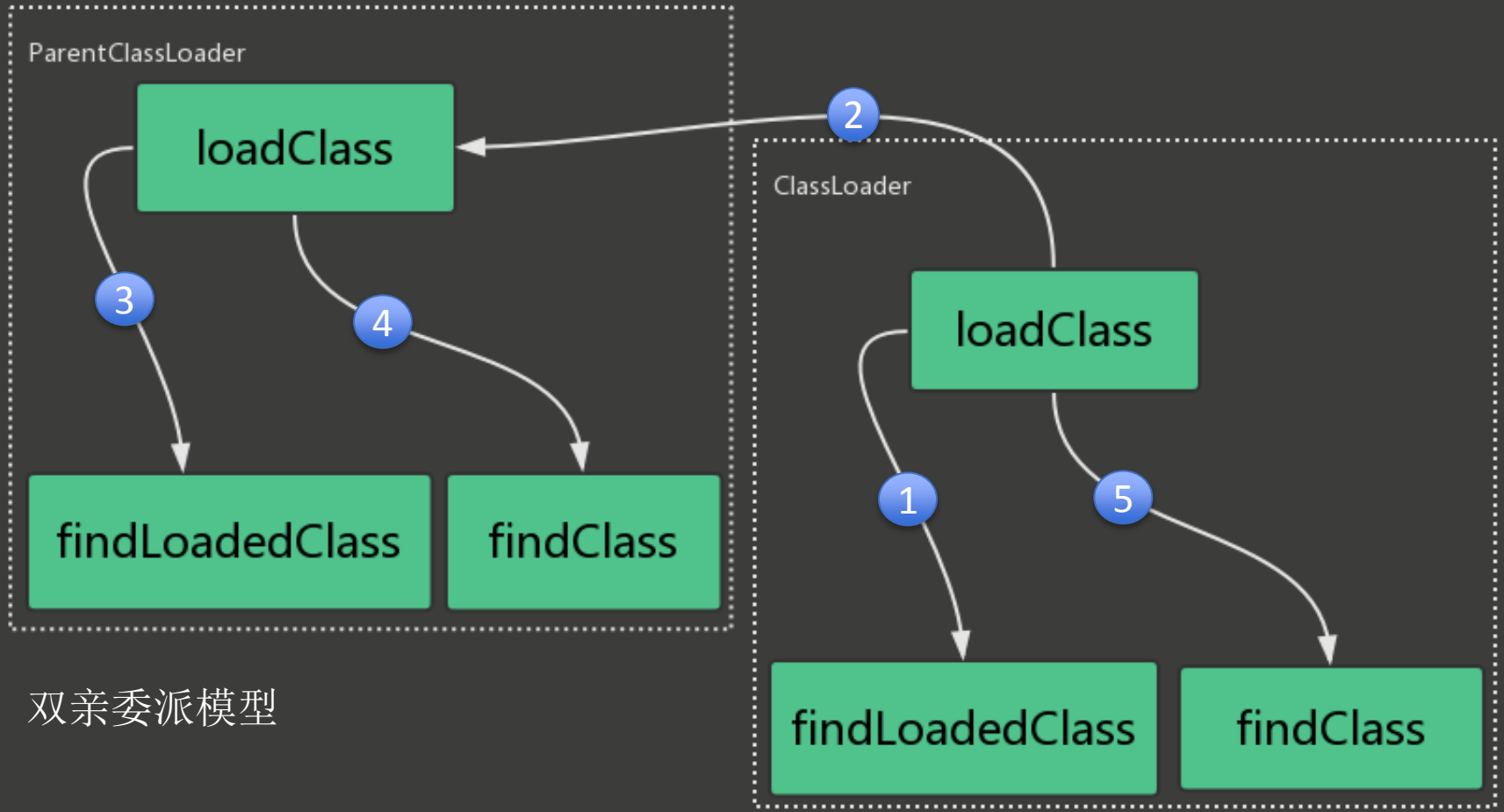


1.4 IPluginClient



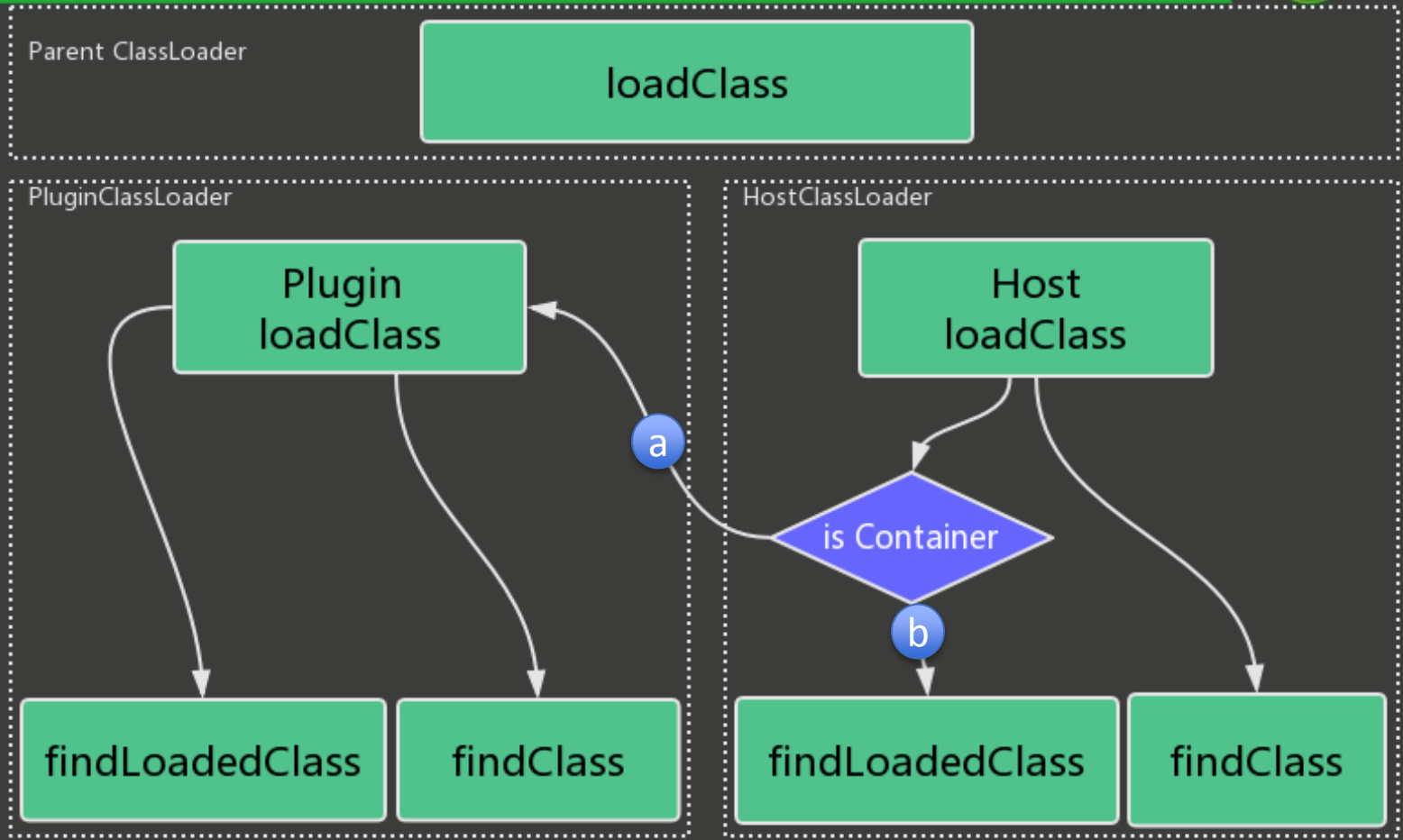
2.ClassLoader处理

2.1 ClassLoader 工作原理



双亲委派模型

2.2 唯一HOOK点—宿主ClassLoader



打破双亲委派
拦截坑位组件
平行ClassLoader

3.Context和Resources处理

内容

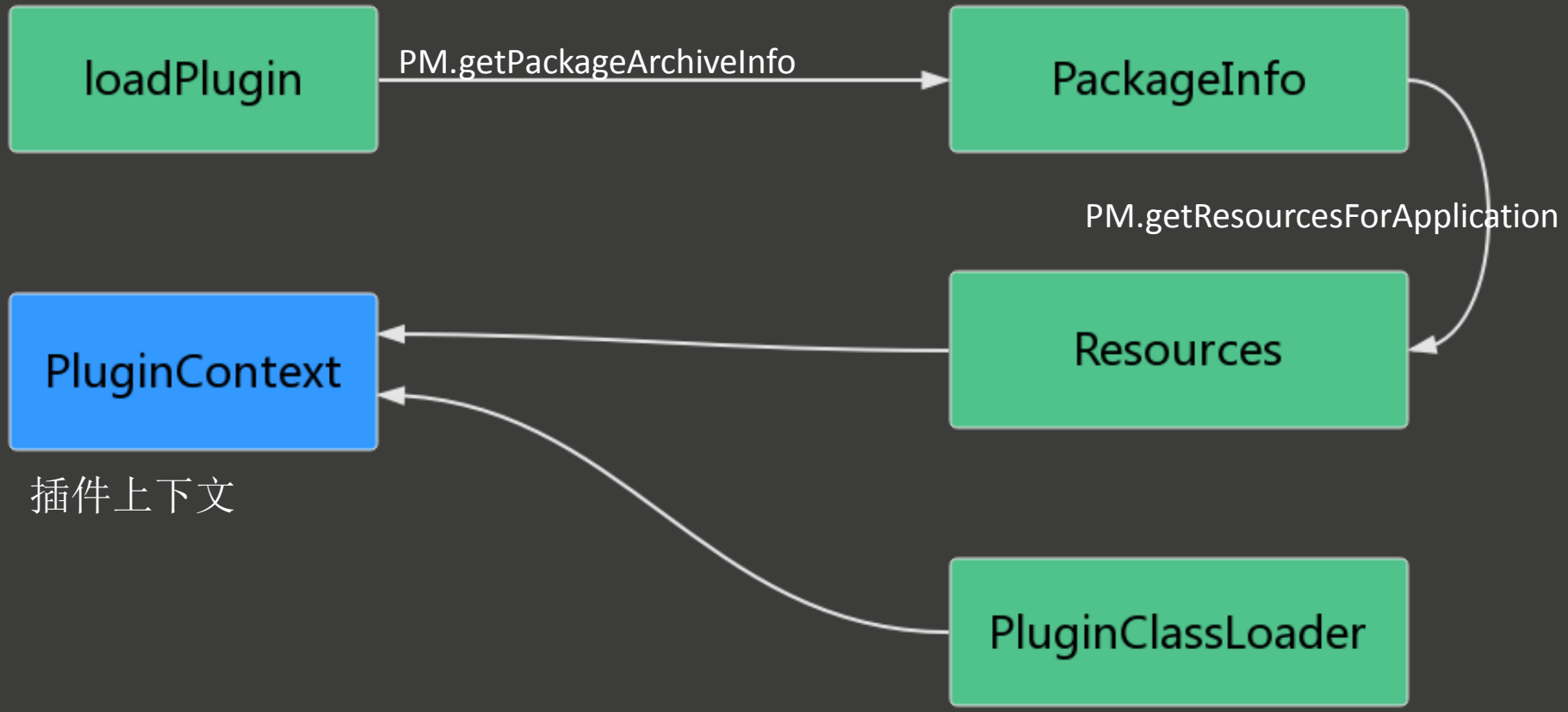
1.插件Context创建流程

2.插件Activity的Context替换

3.插件Resources处理

4.定制插件Activity的Layout流程

3.1 插件上下文创建

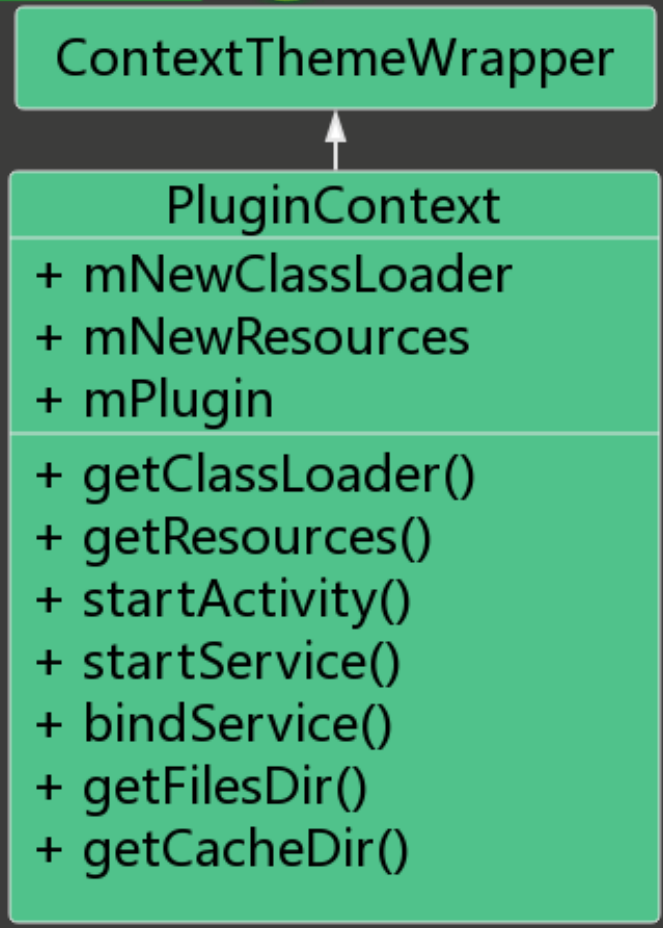


插件上下文

3.1 插件上下文创建

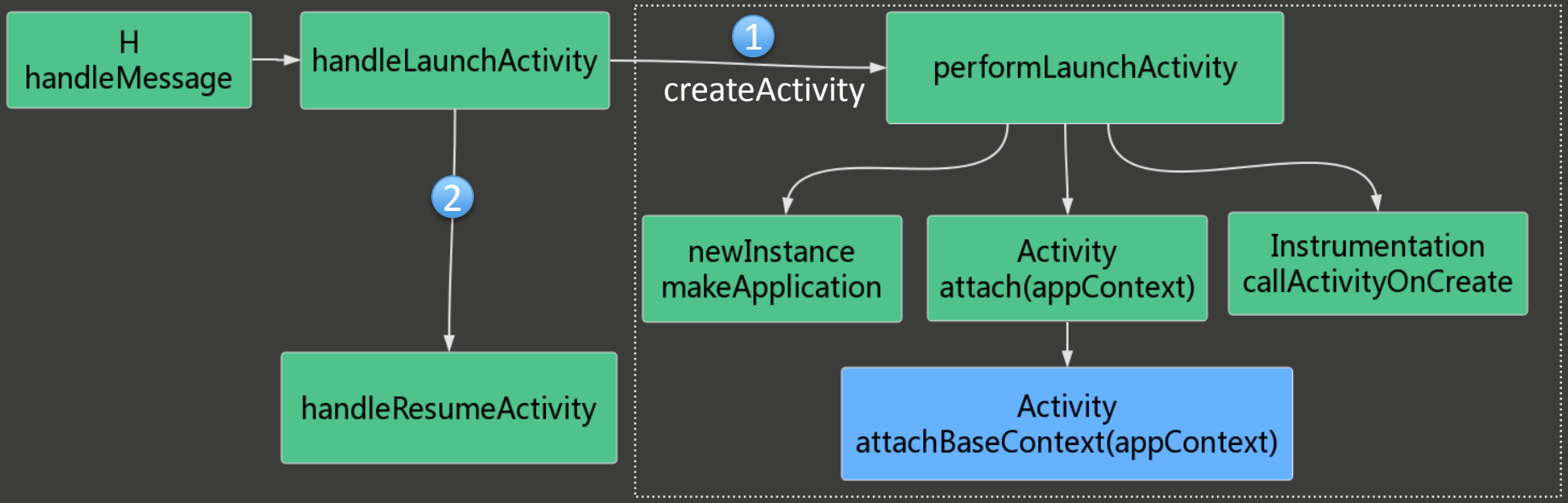
1.插件上下文：PluginContext

2.加载插件时，创建并传递给插件，插件中四大组件都使用



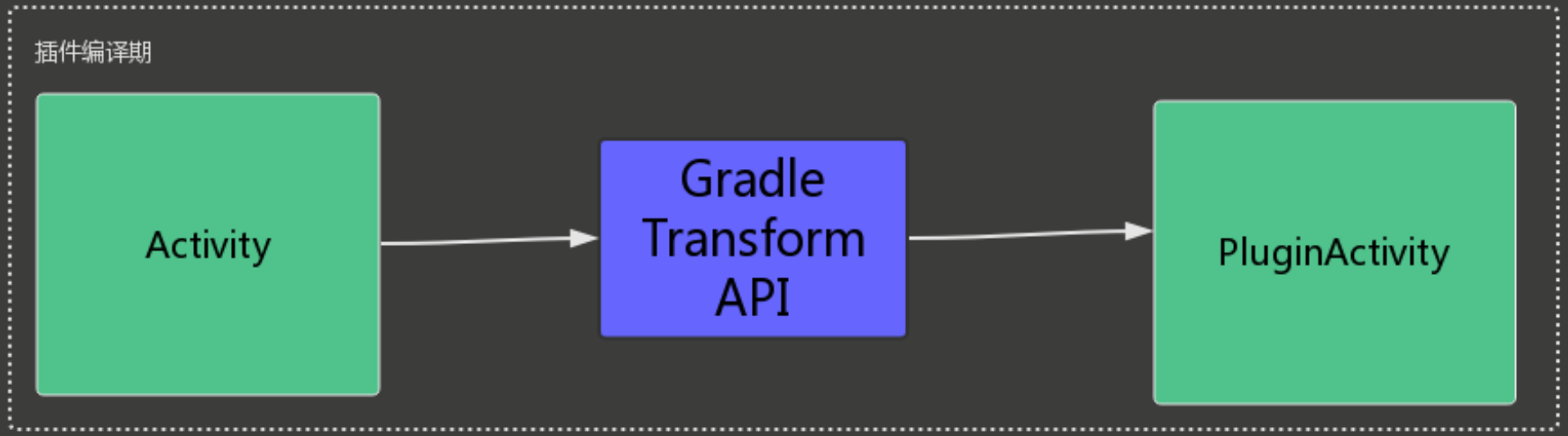
3.2 插件Activity上下文替换

Activity创建过程：



3.2 插件Activity上下文替换

```
@Override
protected void attachBaseContext(Context newBase) {
    newBase = RePluginInternal.createActivityContext(this, newBase);
    super.attachBaseContext(newBase);
}
```



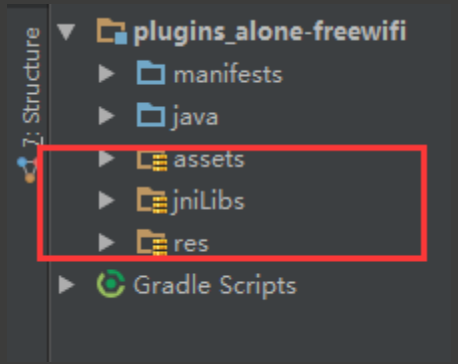
3.3 插件资源处理

插件资源分类：

1.assets

2.res

3.so



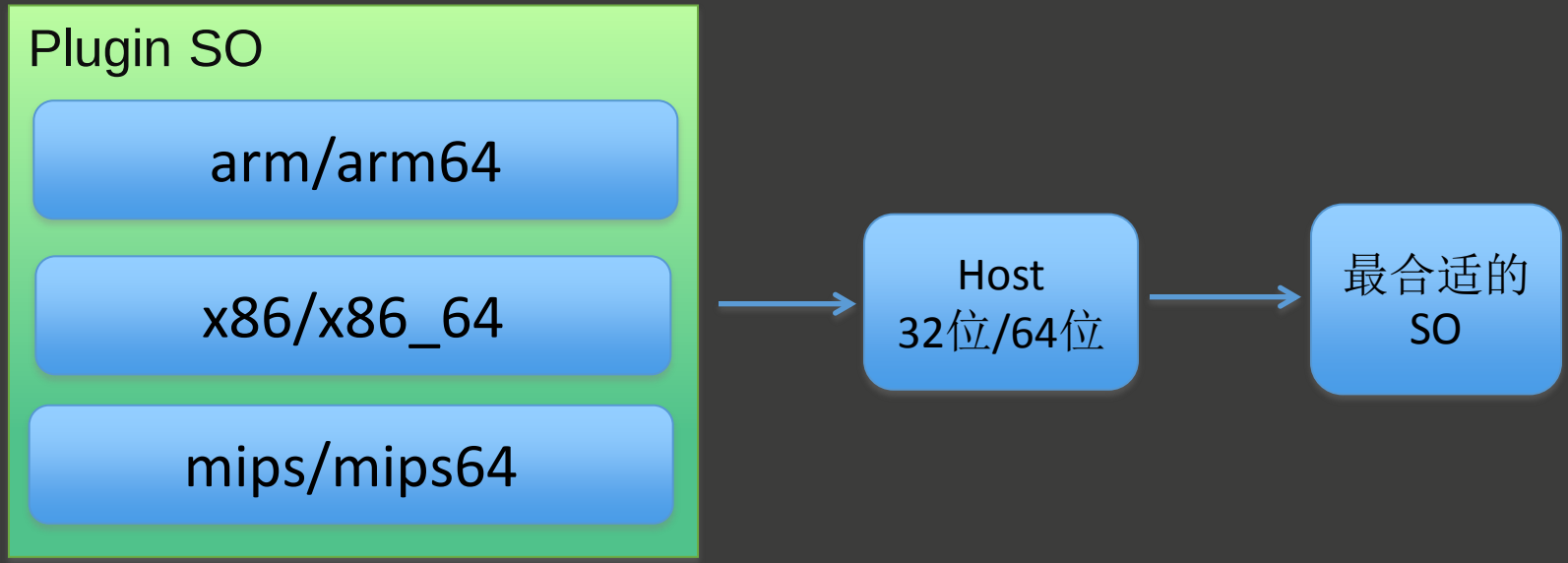
3.3 资源-assets和res

assets和res：

```
@Override
public Resources getResources() {
    if (mNewResources != null) {
        return mNewResources;
    }
    return super.getResources();
}

@Override
public AssetManager getAssets() {
    if (mNewResources != null) {
        return mNewResources.getAssets();
    }
    return super.getAssets();
}
```

3.3 资源-so文件

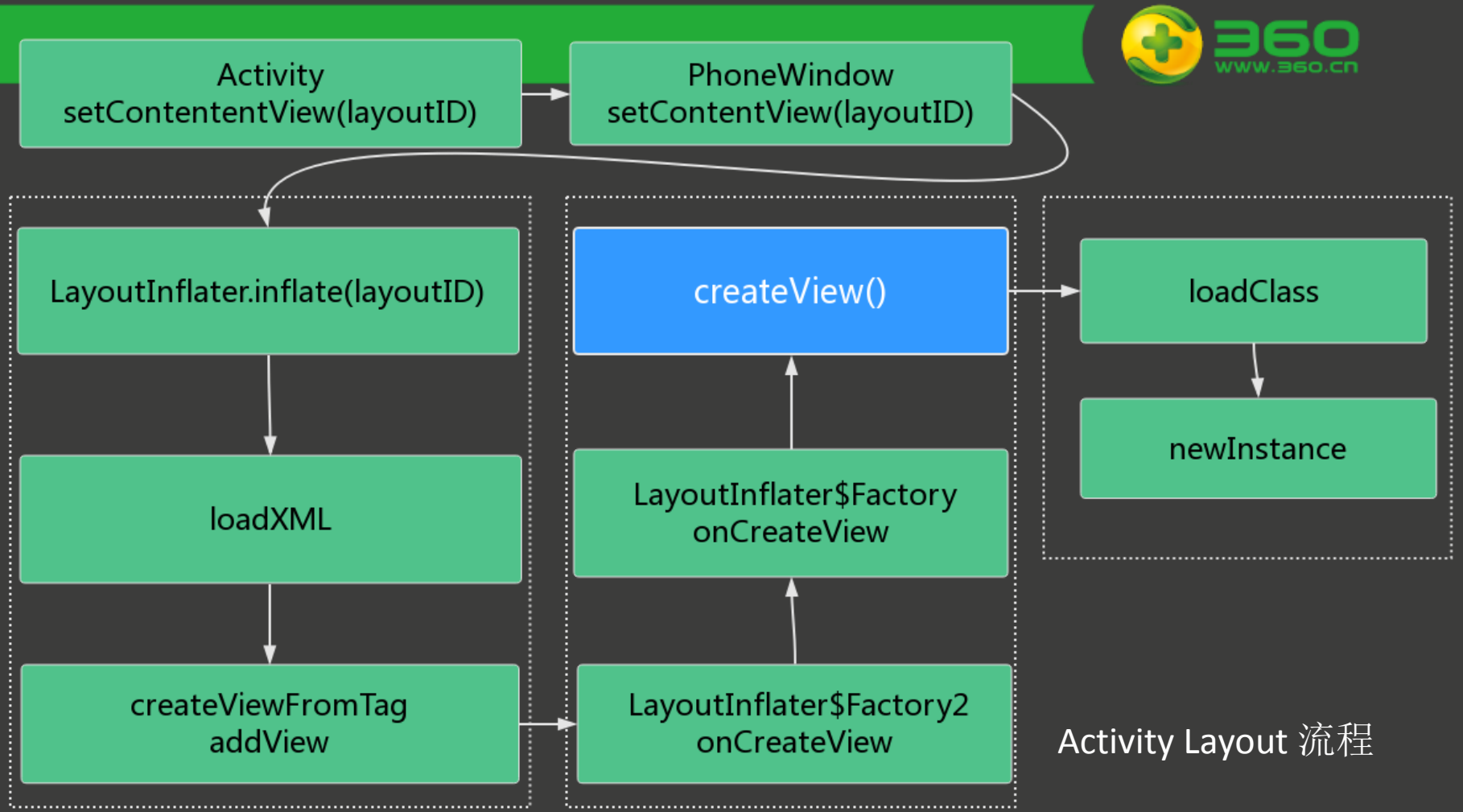

Plugin SO
arm/arm64
x86/x86_64
mips/mips64
Host
32位/64位
最合适的
SO

3.4 定制Activity Layout流程

1. 替换Activity的Layout资源

2. 替换Layout的自定义View控件

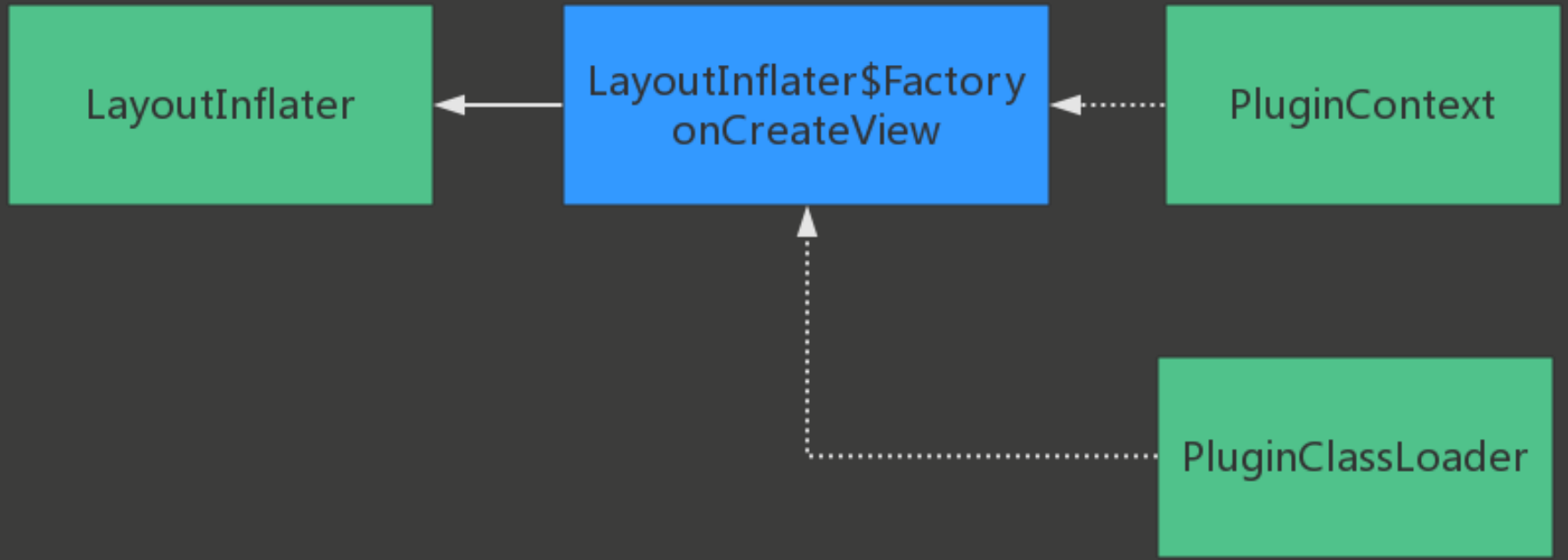
```
<com.qihoo360.mobilesafe.ui.common.layout.CommonTitleBar
    android:id="@+id/home_header_view"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:text="@string/app_name" />
```



Activity Layout 流程

3.4 定制Activity Layout流程

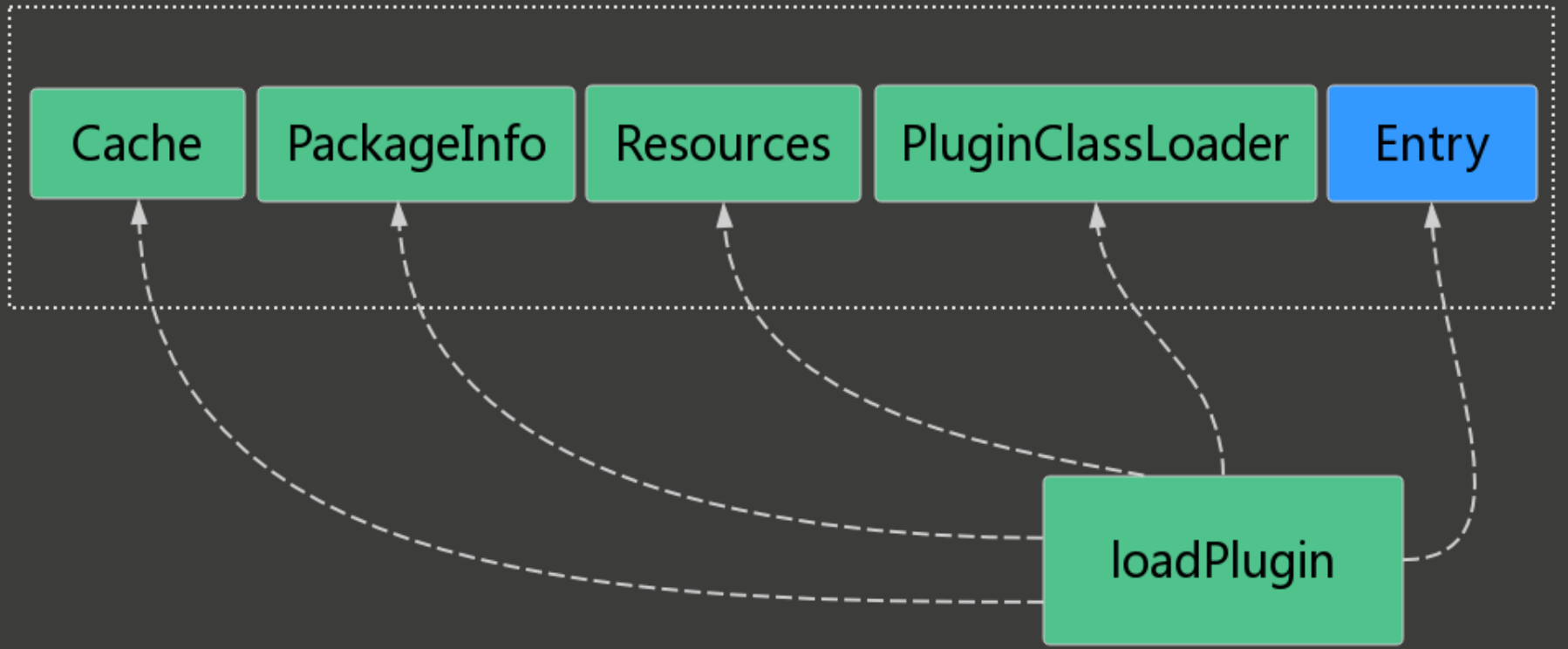
定制 LayoutInflater Factory：



4.插件加载

4. 插件加载

加载理念：按需加载



5.内部通信框架

内容

1.同步Binder

2.同进程，跨插件通信

3.跨进程，跨插件通信

4.LocalBroadcastManager使用

5.1 异步Binder


```
public class CoreService extends Service {
    @Override
    public IBinder onBind(Intent intent) {
        return XXXXX;
    }
}
```

```
bindService(intent, conn, BIND_AUTO_CREATE);

private boolean isBound = false;
private ServiceConnection conn = new ServiceConnection() {
    @Override
    public void onServiceConnected(ComponentName name, IBinder binder) {
        isBound = true;
    }

    @Override
    public void onServiceDisconnected(ComponentName name) {
        isBound = false;
    }
};
```

5.1 同步Binder 框架

同步Binder服务框架

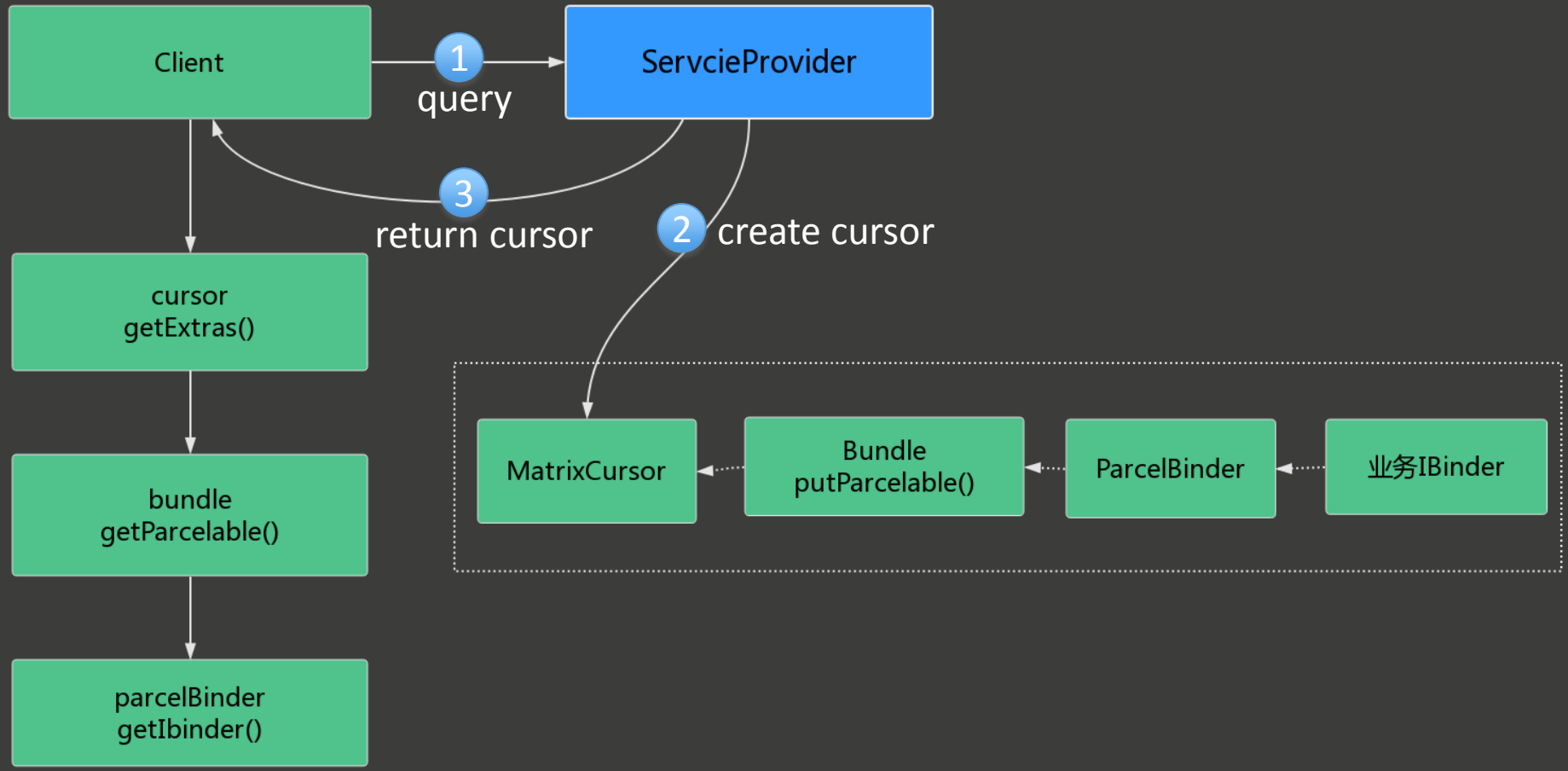
目标

可以同步的从远程进程中取一个Binder。

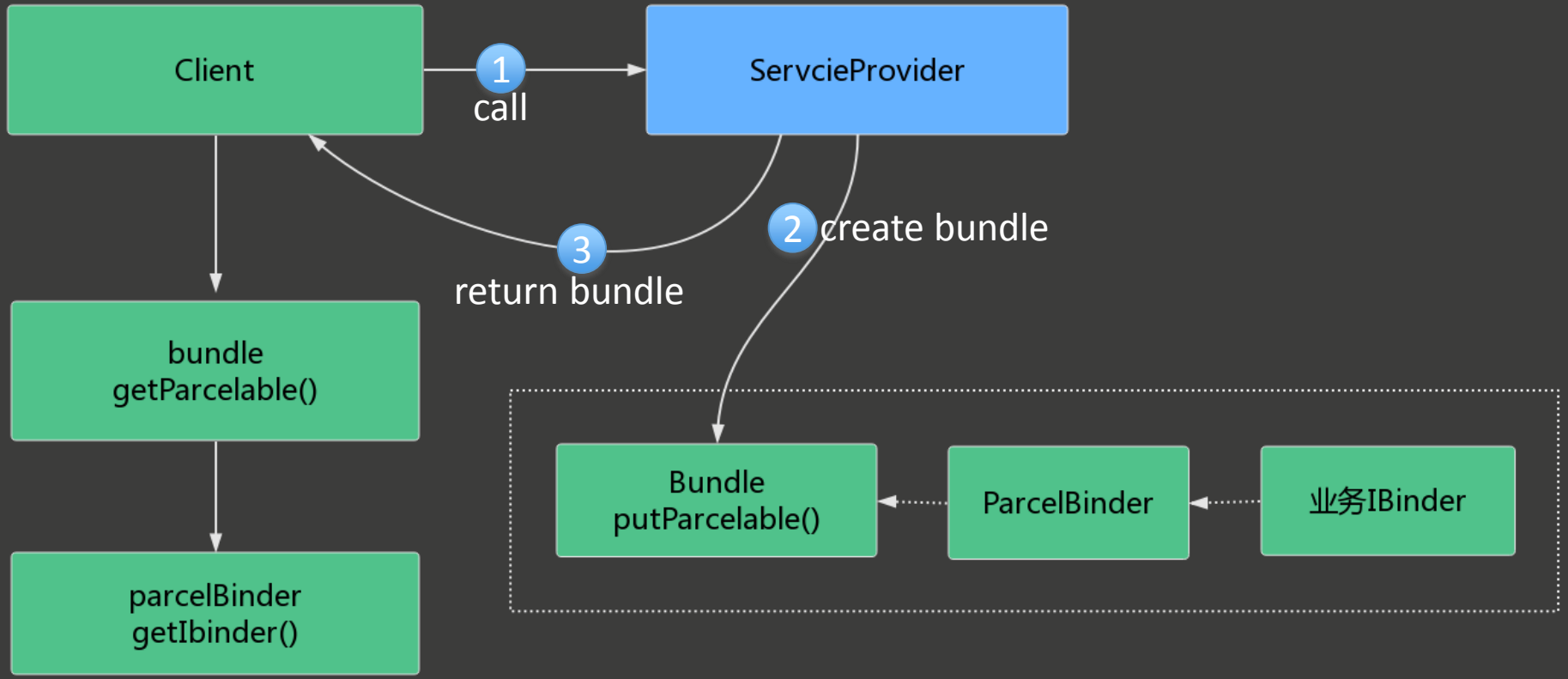
原理

利用 ContentProvider 跨进程特性，以及ContentProvider 的query()/call()接口同步特性，可以实现跨进程同步传递Binder。

5.1 同步Binder传输——query



5.1 同步Binder传输——call

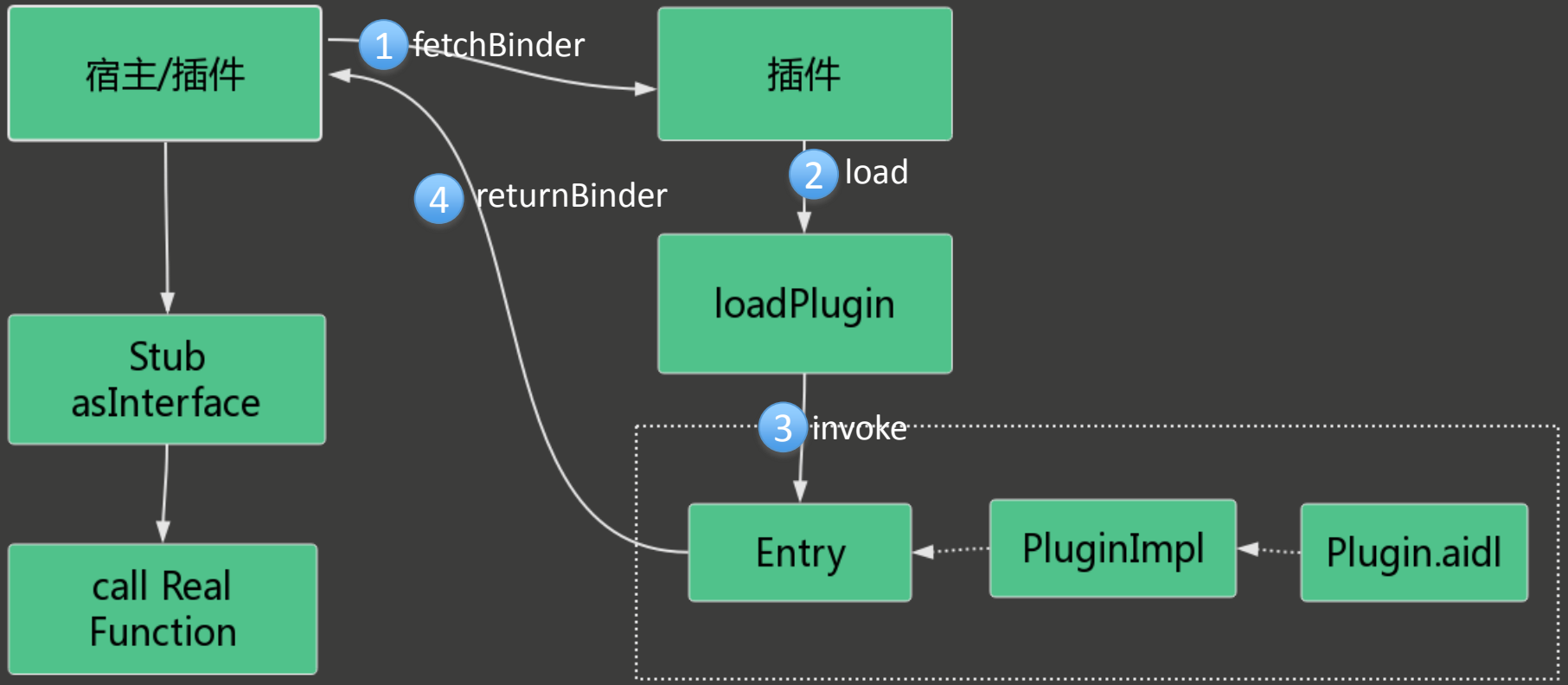


5.2 同进程跨插件通信

调用接口：

IBinder fetchBinder(String pluginName, String module);

5.2 同进程跨插件通信

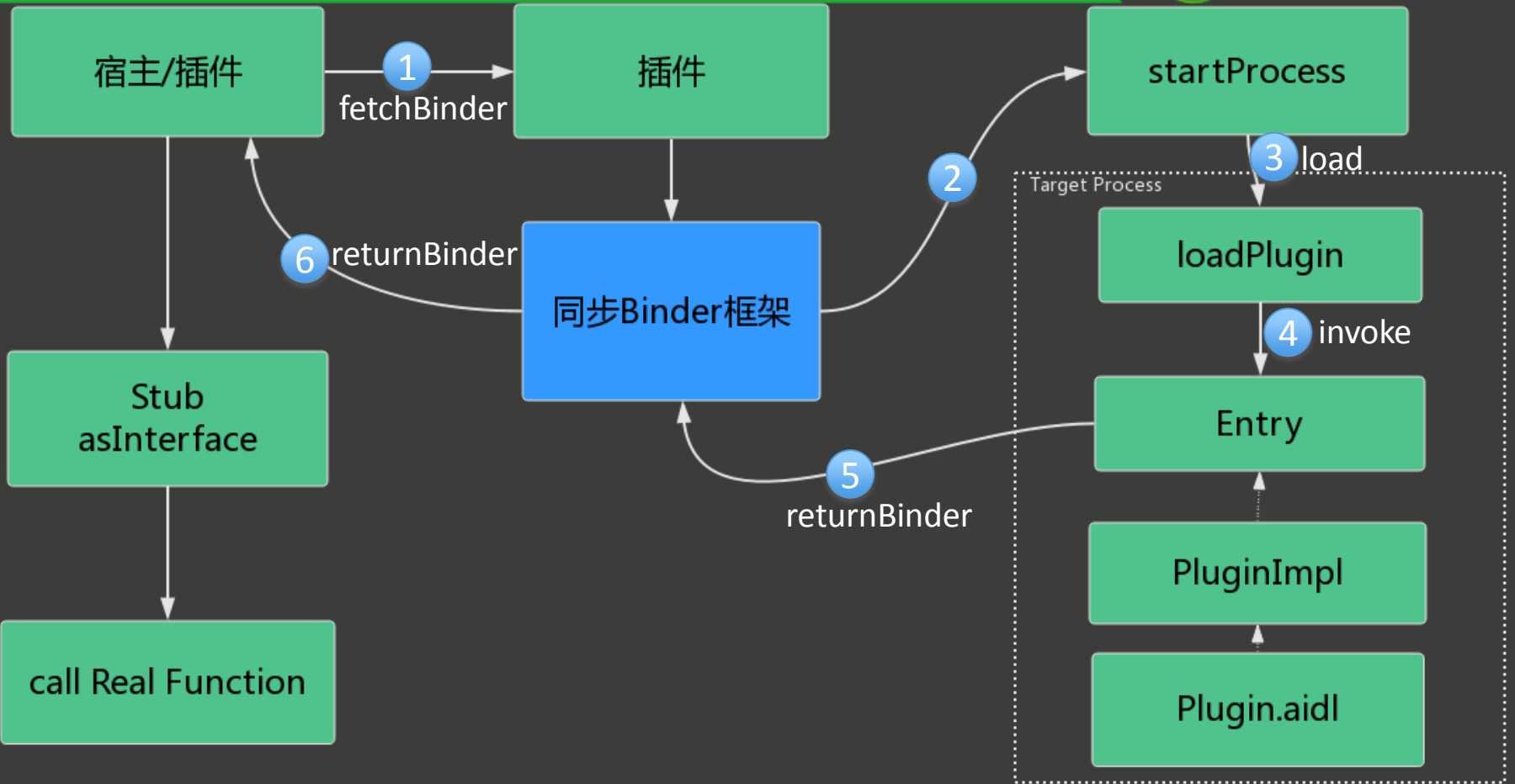

宿主/插件
1 fetchBinder
插件
2 load
4 returnBinder
loadPlugin
Stub
asInterface
3 invoke
call Real
Function
Entry
PluginImpl
Plugin.aidl

5.3 跨进程跨插件通信

调用接口：

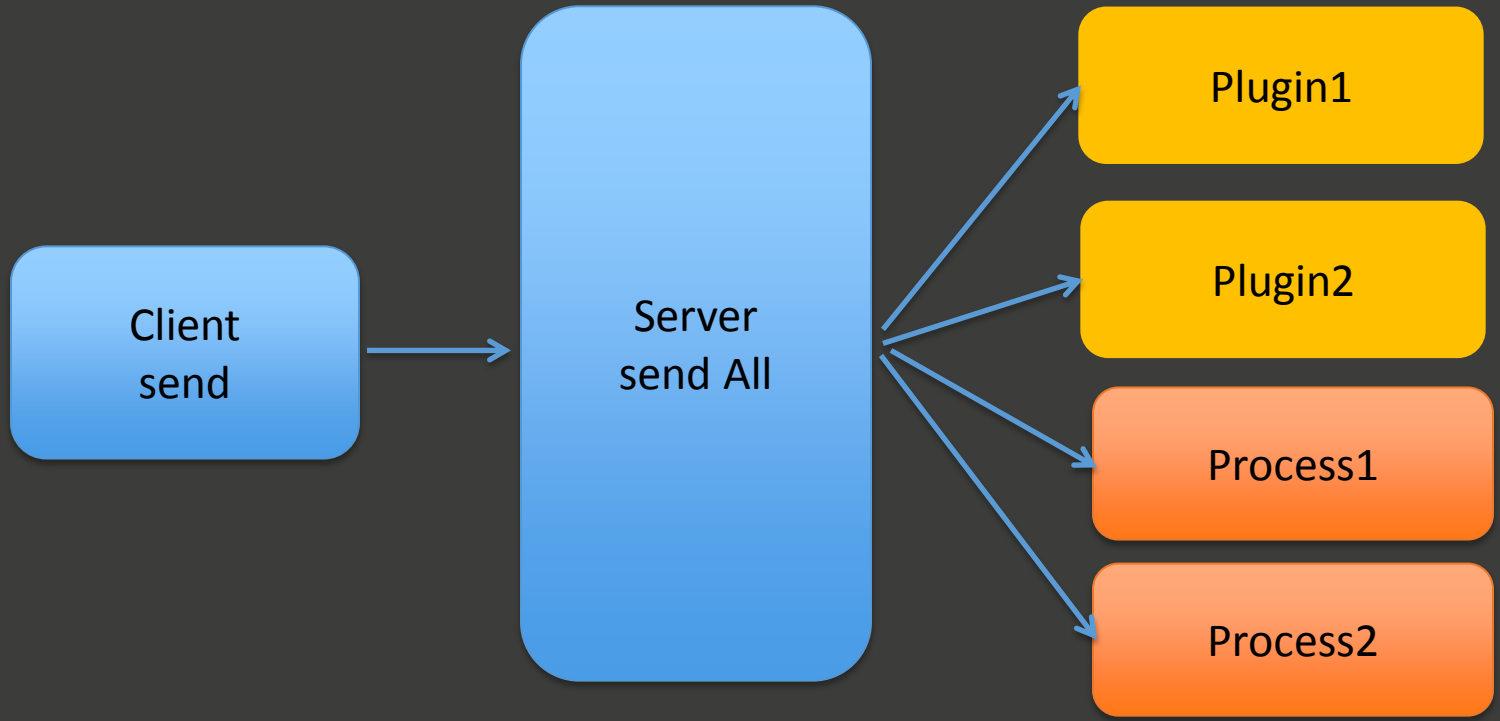
IBinder fetchBinder(String pluginName, String module, int process);

5.3 跨进程跨插件通信


宿主/插件
1
fetchBinder
插件
startProcess
3
load
Target Process
2
6
returnBinder
同步Binder框架
loadPlugin
4
invoke
Stub
asInterface
Entry
5
returnBinder
PluginImpl
call Real Function
Plugin.aidl

5.4 LocalBroadcastManager

实现跨进程，跨插件发消息



6.Activity插件化流程

6 内容

1.Activity坑位生成

2.启动插件Activity流程

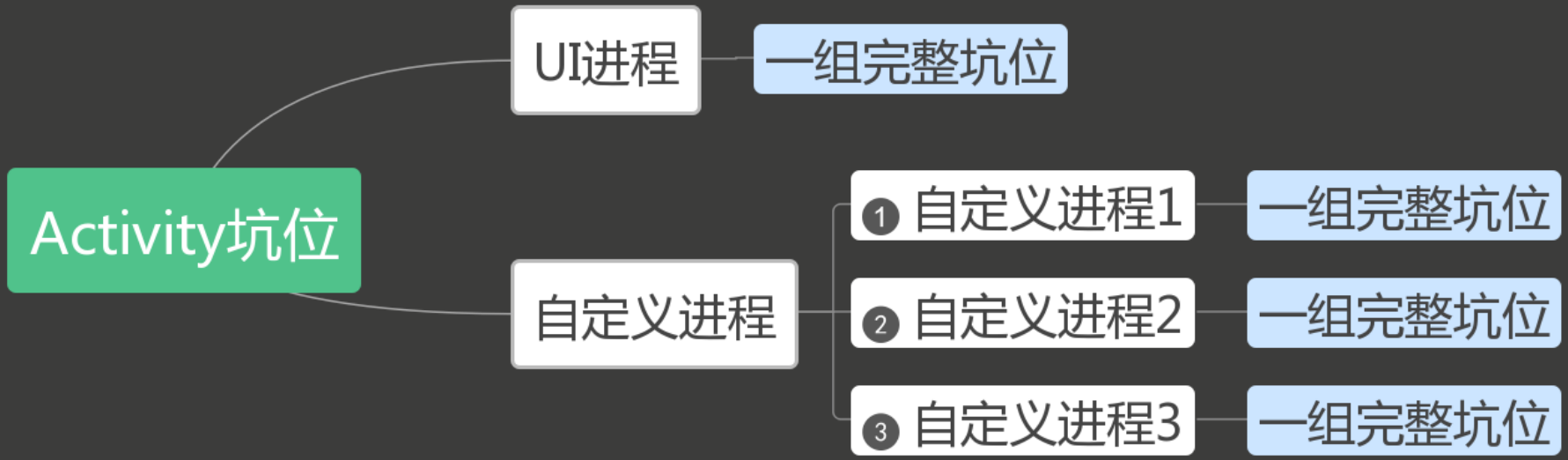
6.1 Activity坑位生成

一种坑位示例：

```
<activity
    android:theme="@android:style/Theme.NoTitleBar"
    android:name="com.qihoo360.mobilesafe.loader.a.ActivityP2TA3STNTS1"
    android:exported="false"
    android:process=":p2"
    android:taskAffinity=":t3"
    android:launchMode="singleTask"
    android:screenOrientation="portrait"
    android:configChanges="keyboard|keyboardHidden|orientation|screenSize" />
```

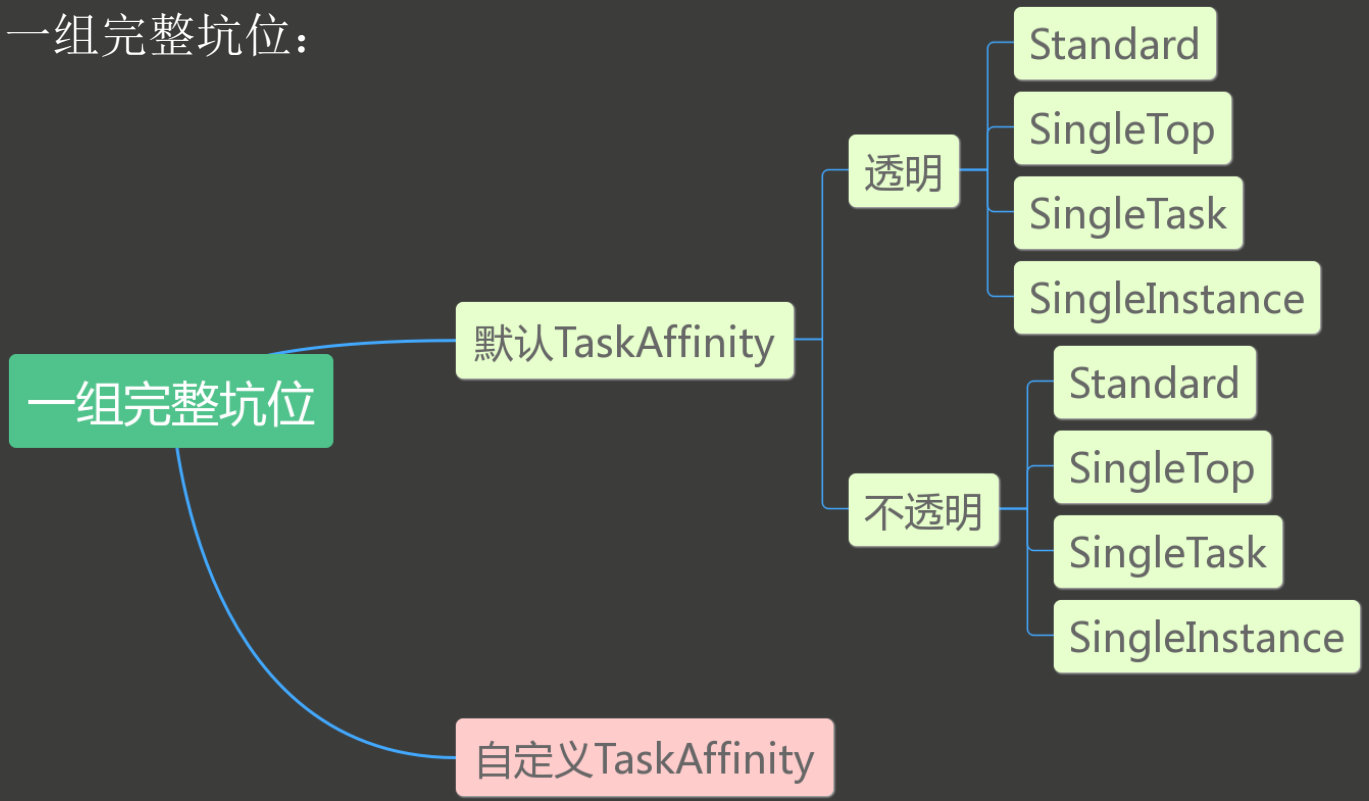
6.1 Activity坑位生成

坑位分组：

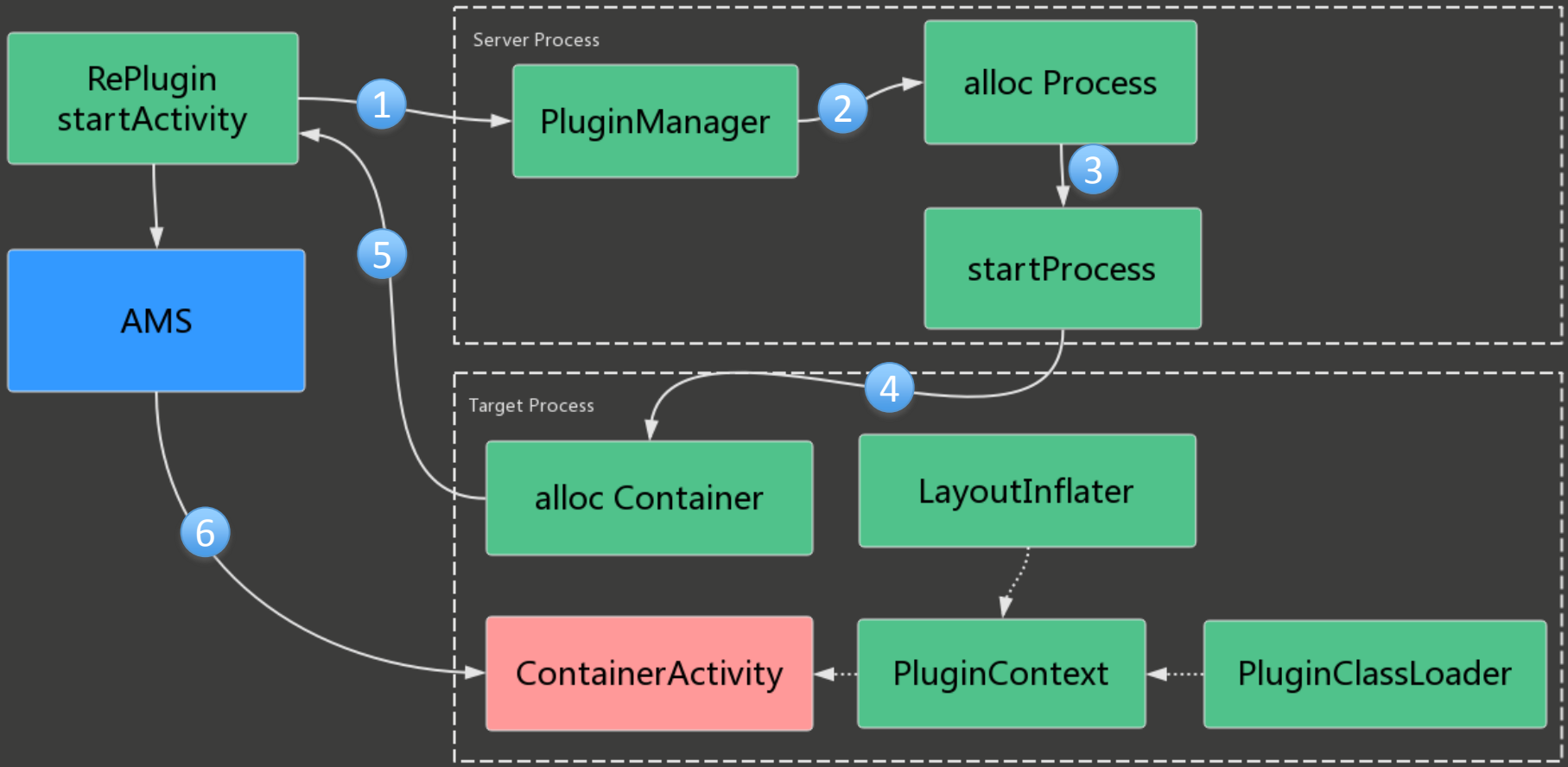


6.1 Activity坑位生成

一组完整坑位：



6.2 startActivity流程



7.最佳实践

7.1 内部通信

1. 建议使用RePlugin原生通信体系；
2. 也支持引入第三方，如：EventBus、HermesEventBus；

7.2 宿主/插件使用插件UI

Activity：

直接使用

Fragment：

1）如果在XML中使用，需要拦截系统对于宿主中XXFragment至插件；

2）如果在代码中使用，直接获取插件ClassLoader加载插件Fragment；

View：

使用插件上下文，RePlugin.fetchView()；

7.3 插件使用宿主代码

1.宿主中需要共享的代码，封装为JAR；

2.插件provided引用；

3.打开“如果插件里没有，就从宿主ClassLoader中找”开关；

7.4 提取业务插件/基础插件

业务插件：

1.将变动频繁的业务逻辑，抽象至单独插件，独立更新；

2.使用中转插件来管理新业务；

基础插件：

将基础功能（如下载、网络、图片、数据统计、缓存、 用户中心、分享、支付、WebView等），封装为插件，独立升级，同时对外暴露Binder接口，供宿主和其他插件使用；

7.5 动态/静态分配插件中进程

1.交给框架，自动分配进程

2.在插件AndroidManifest.xml中配置静态映射表

```
<meta-data
   android:name="process_map"
   android:value="[
   {'from':'com.qihoo360.sample:push1','to':'$p0'}
   ]"/>
```

7.6 加快插件加载速度

1.ART虚拟机，加载插件，必须经过dex2oat过程；

2.插件加载时，学习Android N的混合编译，“解释模式”加载，空闲时“全编译”；

技术交流（干货）：奇卓社（360移动技术微信公众号）