

Weishu's Notes

Android插件化原理解析——Hook机制之动态代理

发表于 2016-01-28 | [54条评论](#) | 5358次阅读

使用代理机制进行API Hook进而达到方法增强是框架的常用手段，比如J2EE框架Spring通过动态代理优雅地实现了AOP编程，极大地提升了Web开发效率；同样，插件框架也广泛使用了代理机制来增强系统API从而达到插件化的目的。本文将带你了解基于动态代理的Hook机制。

阅读本文之前，可以先clone一份 [understand-plugin-framework](#)，参考此项目的 `dynamic-proxy-hook` 模块。另外，插件框架原理解析系列文章见[索引](#)。

代理是什么

为什么需要代理呢？其实这个代理与日常生活中的“代理”，“中介”差不多；比如你想海淘买东西，总不可能亲自飞到国外去购物吧，这时候我们使用第三方海淘服务比如惠惠购物助手等；同样拿购物为例，有时候第三方购物会有折扣比如当初的米折网，这时候我们可以少花点钱；当然有时候这个“代理”比较坑，坑我们的钱，坑我们的货。

从这个例子可以看出来，代理可以实现**方法增强**，比如常用的日志,缓存等；也可以实现方法拦截，通过代理方法修改原方法的参数和返回值，从而实现某种不可告人的目的~接下来我们用代码解释一下。

静态代理

静态代理，是最原始的代理方式；假设我们有一个购物的接口，如下：

```
1 public interface Shopping {  
2     Object[] doShopping(long money);  
3 }
```

它有一个原始的实现，我们可以理解为亲自，直接去商店购物：

```
1 public class ShoppingImpl implements Shopping {  
2     @Override  
3     public Object[] doShopping(long money) {  
4         System.out.println("逛淘宝 ,逛商场,买买买!!");  
5         System.out.println(String.format("花了%s块钱", money));  
6         return new Object[] { "鞋子", "衣服", "零食" };
```

```
7     }  
8 }
```

好了，现在我们自己没时间但是需要买东西，于是我们就找了个代理帮我们买：

```
1 public class ProxyShopping implements Shopping {  
2  
3     Shopping base;  
4  
5     ProxyShopping(Shopping base) {  
6         this.base = base;  
7     }  
8  
9     @Override  
10    public Object[] doShopping(long money) {  
11  
12        // 先黑点钱(修改输入参数)  
13        long readCost = (long) (money * 0.5);  
14  
15        System.out.println(String.format("花了%s块钱", readCost));  
16  
17        // 帮忙买东西  
18        Object[] things = base.doShopping(readCost);  
19  
20        // 偷梁换柱(修改返回值)  
21        if (things != null && things.length > 1) {  
22            things[0] = "被掉包的东西!!";  
23        }  
24  
25        return things;  
26    }  
}
```

很不幸，我们找的这个代理有点坑，坑了我们的钱还坑了我们的货；先忍忍。

动态代理

传统的静态代理模式需要为每一个需要代理的类写一个代理类，如果需要代理的类有几百个那不是要累死？为了更优雅地实现代理模式，JDK提供了动态代理方式，可以简单理解为JVM可以在运行时帮我们动态生成一系列的代理类，这样我们就不需要手写每一个静态的代理类了。依然以购物为例，用动态代理实现如下：

```
1 public static void main(String[] args) {  
2     Shopping women = new ShoppingImpl();  
3     // 正常购物  
4     System.out.println(Arrays.toString(women.doShopping(100)));  
5     // 招代理  
6     women = (Shopping) Proxy.newProxyInstance(Shopping.class.getClassLoader(),
```

```
7         women.getClass().getInterfaces(), new ShoppingHandler(women));
8
9     System.out.println(Arrays.toString(women.doShopping(100)));
10 }
```

动态代理主要处理 `InvocationHandler` 和 `Proxy` 类；完整代码可以见[github](#)

代理Hook

我们知道代理有比原始对象更强大的能力，比如飞到国外买东西，比如坑钱坑货；那么很自然，如果我们自己创建代理对象，然后把原始对象替换为我们的代理对象，那么就可以在这个代理对象为所欲为了；修改参数，替换返回值，我们称之为Hook。

下面我们Hook掉 `startActivity` 这个方法，使得每次调用这个方法之前输出一条日志；（当然，这个输入日志有点点弱，只是为了展示原理；只要你想，你想可以替换参数，拦截这个 `startActivity` 过程，使得调用它导致启动某个别的Activity，指鹿为马！）

首先我们得找到被Hook的对象，我称之为Hook点；什么样的对象比较好Hook呢？自然是**容易找到的对象**。什么样的对象容易找到？**静态变量和单例**；在一个进程之内，静态变量和单例变量是相对不容易发生变化的，因此非常容易定位，而普通的对象则要么无法标志，要么容易改变。我们根据这个原则找到所谓的Hook点。

然后我们分析一下 `startActivity` 的调用链，找出合适的Hook点。我们知道对于 `Context.startActivity`（`Activity.startActivity`的调用链与之不同），由于 `Context` 的实现实际上是 `ContextImpl`；我们看 `ContextImpl` 类的 `startActivity` 方法：

```
1 @Override
2 public void startActivity(Intent intent, Bundle options) {
3     warnIfCallingFromSystemProcess();
4     if ((intent.getFlags() & Intent.FLAG_ACTIVITY_NEW_TASK) == 0) {
5         throw new AndroidRuntimeException(
6             "Calling startActivity() from outside of an Activity "
7             + " context requires the FLAG_ACTIVITY_NEW_TASK flag."
8             + " Is this really what you want?");
9     }
10    mMainThread.getInstrumentation().execStartActivity(
11        getOuterContext(), mMainThread.getApplicationThread(), null,
12        (Activity)null, intent, -1, options);
13 }
```

这里，实际上使用了 `ActivityThread` 类的 `mInstrumentation` 成员的 `execStartActivity` 方法；注意到，`ActivityThread` 实际上是主线程，而主线程一个进程只有一个，因此这里是一个良好的Hook点。

接下来就是想要Hook掉我们的主线程对象，也就是把这个主线程对象里面的 `mInstrumentation` 给

替换成我们修改过的代理对象；要替换主线程对象里面的字段，首先我们得拿到主线程对象的引用，如何获取呢？`ActivityThread` 类里面有一个静态方法 `currentActivityThread` 可以帮助我们拿到这个对象类；但是 `ActivityThread` 是一个隐藏类，我们需要用反射去获取，代码如下：

```
1 // 先获取到当前的ActivityThread对象
2 Class<?> activityThreadClass = Class.forName("android.app.ActivityThread");
3 Method currentActivityThreadMethod = activityThreadClass.getDeclaredMethod("currentActivityThread");
4 currentActivityThreadMethod.setAccessible(true);
5 Object currentActivityThread = currentActivityThreadMethod.invoke(null);
```

拿到这个 `currentActivityThread` 之后，我们需要修改它的 `mInstrumentation` 这个字段为我们的代理对象，我们先实现这个代理对象，由于JDK动态代理只支持接口，而这个 `Instrumentation` 是一个类，没办法，我们只有手动写静态代理类，覆盖掉原始的方法即可。（`cglib` 可以做到基于类的动态代理，这里先不介绍）

```
1 public class EvilInstrumentation extends Instrumentation {
2
3     private static final String TAG = "EvilInstrumentation";
4
5     // ActivityThread中原始的对象，保存起来
6     Instrumentation mBase;
7
8     public EvilInstrumentation(Instrumentation base) {
9         mBase = base;
10    }
11
12    public ActivityResult execStartActivity(
13        Context who, IBinder contextThread, IBinder token, Activity target,
14        Intent intent, int requestCode, Bundle options) {
15
16        // Hook之前，XXX到此一游！
17        Log.d(TAG, "\n执行了startActivity, 参数如下: \n" + "who = [" + who + "], " +
18            "\ncontextThread = [" + contextThread + "], \ntoken = [" + token + "], " +
19            "\ntarget = [" + target + "], \nintent = [" + intent +
20            "], \nrequestCode = [" + requestCode + "], \noptions = [" + options + "]")
21
22        // 开始调用原始的方法，调不调用随你，但是不调用的话，所有的startActivity都失效了。
23        // 由于这个方法是隐藏的，因此需要使用反射调用；首先找到这个方法
24        try {
25            Method execStartActivity = Instrumentation.class.getDeclaredMethod(
26                "execStartActivity",
27                Context.class, IBinder.class, IBinder.class, Activity.class,
28                Intent.class, int.class, Bundle.class);
29            execStartActivity.setAccessible(true);
30            return (ActivityResult) execStartActivity.invoke(mBase, who,
31                contextThread, token, target, intent, requestCode, options);
```

```

32         } catch (Exception e) {
33             // 某该死的rom修改了 需要手动适配
34             throw new RuntimeException("do not support!!! pls adapt it");
35         }
36     }
37 }

```

Ok，有了代理对象，我们要做的就是偷梁换柱！代码比较简单，采用反射直接修改：

```

1 public static void attachContext() throws Exception{
2     // 先获取到当前的ActivityThread对象
3     Class<?> activityThreadClass = Class.forName("android.app.ActivityThread");
4     Method currentActivityThreadMethod = activityThreadClass.getDeclaredMethod("currentAct
5     currentActivityThreadMethod.setAccessible(true);
6     Object currentActivityThread = currentActivityThreadMethod.invoke(null);
7
8     // 拿到原始的 mInstrumentation字段
9     Field mInstrumentationField = activityThreadClass.getDeclaredField("mInstrumentation")
10    mInstrumentationField.setAccessible(true);
11    Instrumentation mInstrumentation = (Instrumentation) mInstrumentationField.get(current
12
13    // 创建代理对象
14    Instrumentation evilInstrumentation = new EvilInstrumentation(mInstrumentation);
15
16    // 偷梁换柱
17    mInstrumentationField.set(currentActivityThread, evilInstrumentation);
18 }

```

好了，我们启动一个Activity测试一下，结果如下：

```

02-22 00:19:39.515 9207-9207/com.weishu.upf.dynamic_proxy_hook.app2 D/EvilInstrumentation: 执行了startActivity, 参数如下:
who = [android.app.Application@7654b0b],
contextThread = [android.app.ActivityThread$ApplicationThread@8807de8],
token = [null],
target = [null],
intent = [Intent { act=android.intent.action.VIEW dat=http://www.baidu.com/... flg=0x10000000 }],
requestCode = [-1],
options = [null]

```

可见，Hook确实成功了！这就是使用代理进行Hook的原理——偷梁换柱。整个Hook过程简要总结如下：

1. 寻找Hook点，原则是静态变量或者单例对象，尽量Hook public的对象和方法，非public不保证每个版本都一样，需要适配。
2. 选择合适的代理方式，如果是接口可以用动态代理；如果是类可以手动写代理也可以使用cglib。
3. 偷梁换柱——用代理对象替换原始对象

完整代码参照：[understand-plugin-framework](http://weishu.me/2016/01/28/understand-plugin-framework-proxy-hook/)；里面留有一个作业：我们目前仅Hook了 Context

类的 `startActivity` 方法，但是 `Activity` 类却使用了自己的 `mInstrumentation`；你可以尝试 Hook掉Activity类的 `startActivity` 方法。

喜欢就点个赞吧~持续更新，请关注github项目 [understand-plugin-framework](#)和我的 [博客](#)!

#android #droidplugin #plugin framework

提升markdown的中文输入效率

Android插件化原理解析——概要

免费分享，随意打赏 ^ ^

54 条评论 3 条新浪微博



风语

写的真好啊，期待您的下一篇

1月29日 回复 顶 转发



可能失踪

内容深刻，表达易懂！期待接下来的作品！

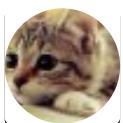
2月4日 回复 顶 转发



大兔男

兄弟，我喜欢你的风格

2月17日 回复 顶 转发



维木

回复 大兔男: 😊 谢谢支持~

2月18日 回复 顶 转发

**hobartEagle**

写的太好了，我正在研究360的droidplugin,里面好多东西不明白，你这样一写，我就明白很多了，期待你后面继续努力吧，帮助我们这些弱小者👍

2月23日 回复 顶 转发

**沉睡者**

讲解的很详细啊，期待中.....

2月26日 回复 顶 转发

**我**

给力,期待下一期

2月29日 回复 顶 转发

**陆镇生**

通俗易懂

2月29日 回复 顶 转发

**oKay**

大神来说说，同是实现热更新，动态加载和react native 有啥区别

2月29日 回复 顶 转发

**陈大明**

留个脚印证明我学过

2月29日 回复 顶 转发

**林边雨边**

现在的水平只能膜拜了

3月2日 回复 顶 转发

**午夜**

写的很好，虽然一次看不太明白，但是可以慢慢消化，支持大神。

3月3日 回复 顶 转发

**寒枫**

留个脚印证明我来过，嘻嘻

3月11日 回复 顶 转发

**Sanjay_F**

很好！

反馈个小事：

在动态代理 的第二行结尾--》 “可以简单理解为在JVM可以在运”，打错了。
另外那个打印语句也有几个符号缺失。

3月25日 回复 顶 转发



维术

回复 Sanjay_F: 多谢提醒，已修复！

3月28日 回复 顶 转发



王焱

为何6.0上不起效果呢？

4月3日 回复 顶 转发



维术

回复 王焱: 请问是有异常吗？有没有更详细的信息？

4月5日 回复 顶 转发



Damon

很赞，对DroidPlugin所需要了解的知识点系统的总结了。

4月5日 回复 顶 转发



朱淞

回复 维术: 请问下，activity的怎么hook，我看源码是Activity内有个mInstrumentation，而且是attachBaseContext之后才赋值的，在attachBaseContext即使替换了这个mInstrumentation，之后又会被覆盖掉了

4月5日 回复 顶 转发



维术

回复 朱淞: 简单的话，在onCreate里面就行了。

4月5日 回复 顶 转发



朱淞

回复 维术: 奥奥 谢谢

4月5日 回复 顶 转发



朱淞

回复 维术: 我的想法是参考你文章里写的方式，然后把activity里的那个mInstrumentation替换掉，请问，你留在attachBaseContext方法里进行hook，是想怎么做的呀？

4月5日 回复 顶 转发



维术

回复 朱淞: 因为这里没有你指出的Activity中的『顺序』问题，Hook越早越好；本来应该在Application的attachBaseContext中做的；但是懒得重新写个Application类，因此在这里做了。实际上Activity类的startActivity Hook不是在onCreate中进行的，而是在ActivityManagerNative中，这个时序是最早的；关于这一点，请查阅我后续的文章。

4月5日 回复 顶 转发

维术

回复 维术: 关于『为什么Hook越早越好』可以类比奈何桥孟婆汤；本来所有对象框架都需要hook掉的，



如果如果hook得太晚了免不了有漏网之鱼；就像有人木有喝孟婆汤就投胎了一样，这会很麻烦。正如孟婆汤需要在奈何桥上喝而不是在六道轮回中。

4月5日 回复 顶 转发



朱淞

回复 维术: 奥奥 好的 谢谢 正在看

4月5日 回复 顶 转发



袁也

作业中在attach里面hook，我还以为要在那替换Instrumentation，结果无效，后来发现那时候还没初始化这个，初始化完了就被系统的自己替换了，然后在onClick监听中替换了就行了。

但是你这边作业留的是，『提前跳转』，我以为是想替换之后直接跳转界面，但是好像那些参数都没有啊

4月7日 回复 顶 转发



袁也

喔不对，是『支持Activity直接跳转请在这里Hook』

4月7日 回复 顶 转发



维术

回复 袁也: 抱歉，已经修复！

4月7日 回复 顶 转发



维术

回复 袁也: 能自己看代码解决也是极好的 😁

4月7日 回复 顶 转发



郑永欣

回复 维术: 感谢楼主分享，我在attachBaseContext方法里面进行的hook，可以完美运行。不过我看了activity源码，它对mInstrumentation的赋值确实是在attachBaseContext方法之后，这应该怎么解释呢？

4月7日 回复 顶 转发



维术

回复 郑永欣: 你再确认一下，什么是完美运行；还有调用的是哪一个startActivity

4月7日 回复 顶 转发



袁也

回复 郑永欣: ==这个就是我上面的情况，其实你在attach方法里hook的，并没有效果，直接被它系统赋值给替换掉了

4月7日 回复 顶 转发



袁也

回复 维术: 哈哈，留作业的形式真是好，有时候看的半懂不懂的，有东西练练能更加强，谢啦

4月7日 回复 顶 转发



何以诚

前辈 我试着run一下这个demo，发现并没有运行成功，首先报的就是android.app.ActivityThread找不到这个类，当然问题不大，我改了下类加载器就好了，之后的sCurrentActivityThread这个域也无法找到，我很郁闷，其它隐藏域都能打印出来就唯独这个，不过还好，有个公共接口留出来了，直接调用一下就行。真是百般挫折啊。。。贴出代码：

```
// 先获取到当前的ActivityThread对象
Class<?> activityThreadClass = null;
try {
    activityThreadClass = Class.forName("android.app.ActivityThread", false, getClassLoader());
    Method method = activityThreadClass.getDeclaredMethod("currentActivityThread");
    Object currentActivityThread = method.invoke(null);

    Field field = activityThreadClass.getDeclaredField("mInstrumentation");
    field.setAccessible(true);
    Instrumentation instrumentation = (Instrumentation) field.get(currentActivityThread);
    MyInstrumentation myInstrumentation = new MyInstrumentation(instrumentation);
    field.set(currentActivityThread, myInstrumentation);
} catch (ClassNotFoundException | IllegalAccessException | NoSuchFieldException e) {
    e.printStackTrace();
} catch (NoSuchMethodException e) {
    e.printStackTrace();
} catch (InvocationTargetException e) {
    e.printStackTrace();
}
```

希望前辈不要嫌弃 也让后面阅读的人能够指点其中不足之处

4月8日 回复 顶 转发



凡卓宋

学习中。多谢！

4月11日 回复 顶 转发



维术

回复 何以诚: 这是个好问题，你得更新一下github的内容；我的测试机是Nexus 运行的Android 6.0系统，所以之前是对着源码看到直接读取sCurrentActivityThread这个字段的；后来这个bug已经修正了；让读者感到困惑，我深感抱歉：(

4月13日 回复 顶 转发



何以诚

回复 维术: 😲 前辈多虑了~

4月14日 回复 顶 转发



顾竹君

你好， 我尝试着去Hook Activity中的Instrumentation但是我发现无法去获取Activity中原来的Instrumentation实例, 代码是这样的你帮忙分析下, 我哪里没有考虑到
...

```
Class<?> currentActivity = Class.forName("android.app.Activity");
```

```
Field instrumentationField = currentActivity.getDeclaredField("mInstrumentation");
instrumentationField.setAccessible(true);
Instrumentation mInstrumentation = (Instrumentation) instrumentationField.get(currentActivity);

Instrumentation customInstrumentation = new ActivityInstrumentation();
instrumentationField.set(currentActivity, customInstrumentation);
...
```

4月15日 回复 顶 转发



维术

回复 顾竹君: 看代码来说，是没有任何问题的；你运行的是什么手机什么Rom，有的rom把这个字段改了，因此你这种hook方法只是在原生系统理论上生效。你可以反射看看当前Activity都有哪些字段。

4月15日 回复 顶 转发



懵逼小书童

真诚感谢楼主的文章，受益匪浅，谢谢！

4月19日 回复 顶 转发



维术

回复 懵逼小书童: 谢谢支持~ 🤗

4月19日 回复 顶 转发



826478309

lz大大，我看博客代码里反射mInstrumentation对象的时候，用的是getField方法，而在4.x版本，mInstrumentation对象是protected属性，所以会导致NoSuchFieldException。建议改成getDeclaredField方法。

4月20日 回复 顶 转发



维术

回复 826478309: 代码以github为准，上面的版本已经修复了；文中的代码已经改正，多谢指出！

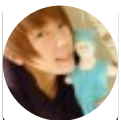
4月20日 回复 顶 转发



826478309

我是写了一个BaseActivity，在这个类的onCreate方法里做的hook，然后作为其他Activity的基类。看了楼上哥们的处理方案，原来是要在ActivityManagerNative里，我再试试这个。

4月20日 回复 顶 转发



CodingWang



4月23日 回复 顶 转发



Felix_Zhang00



4月29日 回复 顶 转发

- 

木讷逐莫

好厉害！可以转载吗

5月3日 回复 顶 转发
- 

维木

回复 木讷逐莫: 可以转载，注明来源即可 🤔

5月3日 回复 顶 转发
- 

geniusmart

厉害，文风也很棒！

5月4日 回复 顶 转发
- 


geniusmart

厉害，文风也很棒

5月4日 回复 顶 转发
- 1

2

社交帐号登录: [微信](#) [微博](#) [QQ](#) [人人](#) [更多»](#)



说点什么吧...

发布