链滴

Activity 的启动过程拦截 Intent 参数

作者：lypoint

原文链接：https://ld246.com/article/1495095510016

来源网站：链滴

```
import  android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.util.Log;

import java.lang.reflect.Field;
import java.lang.reflect.InvocationHandler;
import java.lang.reflect.Method;
import java.lang.reflect.Proxy;

public class HookUtil {

    private Class proxyActivity;

 private Context context;

 public HookUtil(Context context) {
        try {
            this.proxyActivity = Class.forName("class");
  } catch (ClassNotFoundException e) {
            e.printStackTrace();
  }
        this.context = context;
  }

    public void hookSystemHandler() {
        try {

            Class activityThreadClass = Class.forName("android.app.ActivityThread");
  Method currentActivityThreadMethod = activityThreadClass.getDeclaredMethod("currentAct
vityThread");
  currentActivityThreadMethod.setAccessible(true);
  //获取主线程对象
  Object activityThread = currentActivityThreadMethod.invoke(null);
  //获取mH字段
  Field mH = activityThreadClass.getDeclaredField("mH");
  mH.setAccessible(true);
  //获取Handler
  Handler handler = (Handler) mH.get(activityThread);
  //获取原始的mCallBack字段
  Field mCallBack = Handler.class.getDeclaredField("mCallback");
  mCallBack.setAccessible(true);
  //这里设置了我们自己实现了接口的CallBack对象
  mCallBack.set(handler, new ActivityThreadHandlerCallback(handler));

  } catch (Exception e) {
            e.printStackTrace();
  }
    }

    public void hookAms() {
```

```
        //一路反射，直到拿到IActivityManager的对象
 try {
            Class ActivityManagerNativeClss = Class.forName("android.app.ActivityManagerNativ
");
 Field defaultFiled = ActivityManagerNativeClss.getDeclaredField("gDefault");
 defaultFiled.setAccessible(true);
 Object defaultValue = defaultFiled.get(null);
 //反射SingleTon
 Class SingletonClass = Class.forName("android.util.Singleton");
 Field mInstance = SingletonClass.getDeclaredField("mInstance");
 mInstance.setAccessible(true);
 //到这里已经拿到ActivityManager对象
 Object iActivityManagerObject = mInstance.get(defaultValue);

 //开始动态代理，用代理对象替换掉真实的ActivityManager，瞒天过海
 Class IActivityManagerIntercept = Class.forName("android.app.IActivityManager");

 AmsInvocationHandler handler = new AmsInvocationHandler(iActivityManagerObject);

 Object proxy = Proxy.newProxyInstance(Thread.currentThread().getContextClassLoader(), ne
Class[]{IActivityManagerIntercept}, handler);

 //现在替换掉这个对象
 mInstance.set(defaultValue, proxy);

 } catch (Exception e) {
            e.printStackTrace();
 }
   }

    private class ActivityThreadHandlerCallback implements Handler.Callback {

        private Handler handler;

 private ActivityThreadHandlerCallback(Handler handler) {
            this.handler = handler;
 }

        @Override
 public boolean handleMessage(Message msg) {
            Log.i("HookAmsUtil", "handleMessage");
 //替换之前的Intent
 if (msg.what == 100) {
                Log.i("HookAmsUtil", "lauchActivity");
 handleLauchActivity(msg);
 }

            handler.handleMessage(msg);
 return true;  }

        private void handleLauchActivity(Message msg) {
            Object obj = msg.obj;//ActivityClientRecord
 try {
```

```
            Field intentField = obj.getClass().getDeclaredField("intent");
 intentField.setAccessible(true);
 Intent proxyInent = (Intent) intentField.get(obj);
 Intent realIntent = proxyInent.getParcelableExtra("oldIntent");
if (realIntent != null) {
                proxyInent.setComponent(realIntent.getComponent());
 }
        } catch (Exception e) {
            Log.i("HookAmsUtil", "lauchActivity falied");
 }

    }
  }

   public class AmsInvocationHandler implements InvocationHandler {

       private Object iActivityManagerObject;

 public AmsInvocationHandler(Object iActivityManagerObject) {
        this.iActivityManagerObject = iActivityManagerObject;
 }

       @Override
 public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {

        Log.i("HookUtil", method.getName());
 if ("startActivity".contains(method.getName())) {
            Log.e("HookUtil", "Activity已经开始启动");
 //换掉
 Intent intent = null;
int index = 0;
for (int i = 0; i < args.length; i++) {
                Object arg = args[i];
 if (arg instanceof Intent) {
                    //说明找到了startActivity的Intent参数
 intent = (Intent) args[i];
 //这个意图是不能被启动的，因为Acitivity没有在清单文件中注册
 index = i;
 if (intent != null) {
                    Bundle extras = intent.getExtras();
 if (extras != null && extras.keySet() != null) {
                        for (String key : extras.keySet()) {
                            Log.d("HookUtil", "intent.extras.key=" + key + ",value=" + extras.get(ke
));
 }
                    }
                    Log.d("HookUtil", extras + "");
 }
            }
        }

        //伪造一个代理的Intent，代理Intent启动的是proxyActivity
 if (context != null && proxyActivity != null) {
            Intent proxyIntent = new Intent();
```

```
  ComponentName componentName = new ComponentName(context, proxyActivity);
  proxyIntent.setComponent(componentName);
  proxyIntent.putExtra("oldIntent", intent);
  args[index] = proxyIntent;
  }
          }

          return method.invoke(iActivityManagerObject, args);
  }
    }
}
```

在Application中的onCreate函数中插桩

```
HookUtil util = new HookUtil(getContext());
util.hookSystemHandler();
util.hookAms();
```

原文地址

Android插件化系列第（一）篇---Hook技术之Activity的启动过程拦截