1. Hook 钩子

对函数进行挂钩，可以影响整个函数的执行，挂上钩子之后，注入的代码可以接管整个函数，修改函数的参数或者返回值，甚至修改函数的行为等等。

2. Cydia 的 Demo 示例（兼容性问题，所以使用不普遍）

函数 Hook 的方法如下：

找到要挂钩的类，然后找到要挂钩的函数，最后对目标函数进行挂钩重写。

步骤：

(1)导入 lib 文件 substrate-api.jar

(2)AndroidManifest.xml 文件修改

```
<application>
    <meta-data android:name="com.saurik.substrate.main"
android:value="f8.CydiaMain"/>
</application>
<uses-permission android:name="cydia.permission.SUBSTRATE"/>
```

(3)入口类编写

```
import com.saurik.substrate.MS;
public class CydiaMain {
    static void initialize() {
      MS.hookClassLoad("android.content.res.Resources", new MS.ClassLoadHook() {
       Method getColor = resources.getMethod("getColor", Integer.TYPE);
       final MS.MethodPointer old = new MS.MethodPointer();
       MS.hookMethod(resources, getColor, new MS.MethodHook() {
          public Object invoked(Object resources, Object... args)
                throws Throwable
          {
           int color = (Integer) old.invoke(resources, args);
           return color & ~0x0000ff00 | 0x00ff0000;
          }
       }, old);
      }
    }
}
```

```
public class CydiaEntry {
    static void initialize() {
        MS.hookClassLoad("android.content.res.Resources", new MS.ClassLoadHook() {
            @Override
            public void classLoaded(final Class<?> resouces) {
                Method getColor;
                try {
                    getColor = resouces.getDeclaredMethod("getColor", Integer.TYPE);
                } catch (Exception e) {
                    getColor = null;
                }

                if (getColor != null) {
                    final MS.MethodPointer old = new MS.MethodPointer();
                    MS.hookMethod(resouces, getColor, new MS.MethodHook() {
                        @Override
                        public Object invoked(Object res, Object... args) throws Throwable {
                            return (int)old.invoke(res, args) & ~0x0000ff00 | 0x00ff0000;
                        }
                    }, old);
                }
            }
        });
    }
}
```

只需要知道类名就能钩上，而且是属于系统全局属性的钩子，基本不会被检测到，这是 cydia 的特点之一。

3. Java 反射

Java 能够通过反射方法获取类和它的成员，反射相当于提供一些函数，在不知道原始类定义的情况下，修改类中相关成员的属性和值等。

方法如下:

所有类都是继承于 Object 的,所以都可以使用 Object 的方法,也可以强制转换为 Object。所以，当遇到无法表示出的对象时，直接使用 Object 即可。

获取对象的类：

```
Object obj = "123";
Class clazz = obj.getClass();
```

获取类中的方法：

```
Method[] mPubMethods = clazz.getMethods();      //获取公有可直接调用的方法
Method[] mDeclareMethods = clazz.getDeclaredMethods();  //获取类中声明的所有方法
Field[] mPubFields = clazz.getFields();         //获取public 的Field
Field[] mDeclareFields = clazz.getDeclaredFields();     //获取声明的所有Field
```

使用对应的不是以 s 后缀的函数可以获取特定的函数或 field

方法调用

```
method.invoke(obj, arg)
```

域操作

```
field.set(obj, "1");
field.get(obj);
```

访问权限设置，域和方法都一样，控制是否可以直接访问，相当于 public 属性

```
field.isAccessible();
field.setAccessible(true);
```

其余的函数，可以获取函数的名称，还有其他种种信息

```
field.getName();
field.toString();
Class.forName("android.view.Menu");         //寻找类，必须是一个 classloader 下的才能使用
```