链滴

手把手教你实现热更新功能，带你了解 Arthas 热更新背后的原理

作者：9526xu

原文链接：https://ld246.com/article/1574040836447

来源网站：链滴

一、前言

一天下午正在摸鱼的时候，测试小姐姐走了过来求助，说是需要改动测试环境 mock 应用。但是这个用一时半会又找不到源代码存在何处。但是测试小姐姐的活还是一定要帮，突然想起了 Arthas 可以更新应用代码，按照网上的步骤，反编译应用代码，加上需要改动的逻辑，最后热更新成功。对此，试小姐姐很满意，并表示下次会少提 Bug。

嘿嘿，以前一直对热更新背后原理很好奇，借着这个机会，研究一下热更新的原理。

二、Arthas 热更新

我们先来看下 Arthas 是如何热更新的。

详情参考:阿里巴巴Arthas实践--jad/mc/redefine线上热更新一条龙

假设我们现在有一个 HelloService 类，逻辑如下，现在我们使用 Arthas 热更新代码，让其输出 hello arthas。

```
public class HelloService {

    public static void main(String[] args) throws InterruptedException {

        while (true){
            TimeUnit.SECONDS.sleep(1);
            hello();
        }
    }

    public static void hello(){
        System.out.println("hello world");
    }
```

```
}
```

2.1、jad 反编译代码

首先运行 jad 命令反编译 class 文件获取源代码,运行命令如下：。

```
jad --source-only com.andyxh.HelloService > /tmp/HelloService.java
```

2.2、修改反编译之后的代码

拿到源代码之后，使用 VIM 等文本编辑工具编辑源代码，加入需要改动的逻辑。

2.3、查找 ClassLoader

然后使用 sc 命令查找加载修改类的 ClassLoader，运行命令如下:

```
$ sc -d  com.andyxh.HelloService | grep classLoaderHash
 classLoaderHash   4f8e5cde
```

这里运行之后将会得到 ClassLoader 哈希值。

2.4、 mc 内存编译源代码

使用 mc 命令编译上一步修改保存的源代码，生成最终 class 文件。

```
$ mc -c 4f8e5cde  /tmp/HelloService.java  -d /tmp
Memory compiler output:
/tmp/com/andyxh/HelloService.class
Affect(row-cnt:1) cost in 463 ms.
```

2.5、redefine 热更新代码

运行 redefine 命令：

```
$ redefine /tmp/com/andyxh/HelloService.class
redefine success, size: 1
```

热更新成功之后，程序输出结果如下：

一般情况下，我们本地将会有源代码，上面的步骤我们可以进一步省略，我们可以先在自己 IDE 上改代码，编译生成 class 文件。这样我们只需要运行 redefine 命令即可。也就是说实际上起到作用只是 edefine 。

三、 Instrumentation 与 attach 机制

Arthas 热更新功能看起来很神奇，实际上离不开 JDK 一些 API，分别为 instrument API 与 attach API。

3.1 Instrumentation

Java Instrumentation 是 JDK5 之后提供接口。使用这组接口，我们可以获取到正在运行 JVM 相关息，使用这些信息我们构建相关监控程序检测 JVM。另外， 最重要我们可以**替换**和**修改**类的，这样实现了热更新。

Instrumentation 存在两种使用方式，一种为 pre-main 方式，这种方式需要在虚拟机参数指定 Instrmentation 程序，然后程序启动之前将会完成修改或替换类。使用方式如下:

```
java -javaagent:jar Instrumentation_jar -jar xxx.jar
```

有没有觉得这种启动方式很熟悉，仔细观察一下 IDEA 运行输出窗口。

另外很多应用监控工具，如：zipkin、pinpoint、skywalking。

这种方式只能在应用启动之前生效，存在一定的局限性。

JDK6 针对这种情况作出了改进，增加 agent-main 方式。我们可以在应用启动之后，再运行 Instrumentation 程序。启动之后，只有连接上相应的应用，我们才能做出相应改动，这里我们就需要使用 Java 提供 attach API。

3.2 Attach API

Attach API 位于 tools.jar 包，可以用来连接目标 JVM。Attach API 非常简单，内部只有两个主要类，VirtualMachine 与 VirtualMachineDescriptor。

VirtualMachine 代表一个 JVM 实例， 使用它提供 attach 方法，我们就可以连接上目标 JVM。

```
VirtualMachine vm = VirtualMachine.attach(pid);
```

VirtualMachineDescriptor 则是一个描述虚拟机的容器类，通过该实例我们可以获取到 JVM PID(进ID),该实例主要通过 VirtualMachine#list 方法获取。

```
    for (VirtualMachineDescriptor descriptor : VirtualMachine.list()){

        System.out.println(descriptor.id());
    }
```

介绍完热更新涉及的相关原理，接下去使用上面 API 实现热更新功能。

四、实现热更新功能

这里我们使用 Instrumentation agent-main 方式。

4.1、实现 agent-main

首先需要编写一个类，包含以下两个方法:

```
public static void agentmain (String agentArgs, Instrumentation inst);         [1]
public static void agentmain (String agentArgs);            [2]
```

上面的方法只需要实现一个即可。若两个都实现， [1] 优先级大于 [2]，将会被优先执行。

接着读取外部传入 class 文件，调用 Instrumentation#redefineClasses，这个方法将会使用新 class 替换当前正在运行的 class，这样我们就完成了类的修改。

```
public class AgentMain {
    /**
     *
     * @param agentArgs 外部传入的参数，类似于 main 函数 args
     * @param inst
     */
    public static void agentmain(String agentArgs, Instrumentation inst) {
        // 从 agentArgs 获取外部参数
        System.out.println("开始热更新代码");
        // 这里将会传入 class 文件路径
        String path = agentArgs;
        try {
            // 读取 class 文件字节码
            RandomAccessFile f = new RandomAccessFile(path, "r");
            final byte[] bytes = new byte[(int) f.length()];
            f.readFully(bytes);
            // 使用 asm 框架获取类名
            final String clazzName = readClassName(bytes);

            // inst.getAllLoadedClasses 方法将会获取所有已加载的 class
            for (Class clazz : inst.getAllLoadedClasses()) {
                // 匹配需要替换 class
                if (clazz.getName().equals(clazzName)) {
                    ClassDefinition definition = new ClassDefinition(clazz, bytes);
                    // 使用指定的 class 替换当前系统正在使用 class
                    inst.redefineClasses(definition);
                }
            }

        } catch (UnmodifiableClassException | IOException | ClassNotFoundException e) {
            System.out.println("热更新数据失败");
        }


    }

    /**
     * 使用 asm 读取类名
     *
     * @param bytes
     * @return
     */
    private static String readClassName(final byte[] bytes) {
        return new ClassReader(bytes).getClassName().replace("/", ".");
    }
}
```

完成代码之后，我们还需要往 jar 包 manifest 写入以下属性。

```
## 指定 agent-main 全名
Agent-Class: com.andyxh.AgentMain
```

```
## 设置权限，默认为 false，没有权限替换 class
Can-Redefine-Classes: true
```

我们使用 maven-assembly-plugin，将上面的属性写入文件中。

```
<plugin>
    <artifactId>maven-assembly-plugin</artifactId>
    <version>3.1.0</version>
    <configuration>
        <!--指定最后产生 jar 名字-->
        <finalName>hotswap-jdk</finalName>
        <appendAssemblyId>false</appendAssemblyId>
        <descriptorRefs>
            <!--将工程依赖 jar 一块打包-->
            <descriptorRef>jar-with-dependencies</descriptorRef>
        </descriptorRefs>
        <archive>
            <manifestEntries>
                <!--指定 class 名字-->
                <Agent-Class>
                    com.andyxh.AgentMain
                </Agent-Class>
                <Can-Redefine-Classes>
                    true
                </Can-Redefine-Classes>
            </manifestEntries>
            <manifest>
                <!--指定 mian 类名字，下面将会使用到-->
                <mainClass>com.andyxh.JvmAttachMain</mainClass>
            </manifest>
        </archive>
    </configuration>
    <executions>
        <execution>
            <id>make-assembly</id> <!-- this is used for inheritance merges -->
            <phase>package</phase> <!-- bind to the packaging phase -->
            <goals>
                <goal>single</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

到这里我们就完成热更新主要代码，接着使用 Attach API，连接目标虚拟机，触发热更新的代码。

```
public class JvmAttachMain {
    public static void main(String[] args) throws IOException, AttachNotSupportedException, A
entLoadException, AgentInitializationException {
        // 输入参数，第一个参数为需要 Attach jvm pid 第二参数为 class 路径
        if(args==null||args.length<2){
            System.out.println("请输入必要参数，第一个参数为 pid，第二参数为 class 绝对路径");
            return;
        }
        String pid=args[0];
```

```
        String classPath=args[1];
        System.out.println("当前需要热更新 jvm pid 为 "+pid);
        System.out.println("更换 class 绝对路径为 "+classPath);
        // 获取当前 jar 路径
        URL jarUrl=JvmAttachMain.class.getProtectionDomain().getCodeSource().getLocation();
        String jarPath=jarUrl.getPath();

        System.out.println("当前热更新工具 jar 路径为 "+jarPath);
        VirtualMachine vm = VirtualMachine.attach(pid);//7997是待绑定的jvm进程的pid号
        // 运行最终 AgentMain 中方法
        vm.loadAgent(jarPath, classPath);
    }
}
```

在这个启动类，我们最终调用 VirtualMachine#loadAgent，JVM 将会使用上面 AgentMain 方法用传入 class 文件替换正在运行 class。

4.2、运行

这里我们继续开头使用的例子，不过这里加入一个方法获取 JVM 运行进程 ID。

```
public class HelloService {

    public static void main(String[] args) throws InterruptedException {
        System.out.println(getPid());
        while (true){
            TimeUnit.SECONDS.sleep(1);
            hello();
        }
    }

    public static void hello(){
        System.out.println("hello world");
    }

    /**
     * 获取当前运行 JVM PID
     * @return
     */
    private static String getPid() {
        // get name representing the running Java virtual machine.
        String name = ManagementFactory.getRuntimeMXBean().getName();
        System.out.println(name);
        // get pid
        return name.split("@")[0];
    }

}
```

首先运行 HelloService，获取当前 PID,接着复制 HelloService 代码到另一个工程，修改 hello 方法出 hello agent，重新编译生成新的 class 文件。

最后在命令行运行生成的 jar 包。

```
$ java -jar hotswap-jdk.jar  33268  [redacted]GitHub/hotswap-example/target/HelloService.class
当前需要热更新 jvm pid 为 33268
更换 class 绝对路径为 /Users/andy.xu/Documents/GitHub/hotswap-example/target/HelloService.class
当前热更新工具 jar 路径为 /Users/andy.xu/Documents/GitHub/hotswap-example/target/hotswap-jdk.jar
```

HelloService 输出效果如下所示：

```
33268@andy.local
33268
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
开始热更新代码
hello agent
hello agent
hello agent
hello agent
hello agent
```

源代码地址：https://github.com/9526xu/hotswap-example

4.3、调试技巧

普通的应用我们可以在 IDE 直接使用 Debug 模式调试程序，但是上面的程序无法直接使用 Debug 刚开始运行的程序碰到很多问题，无奈之下，只能选择最原始的办法，打印错误日志。后来查看 artha 的文档，发现上面一篇文章介绍使用 IDEA Remote Debug 模式调试程序。

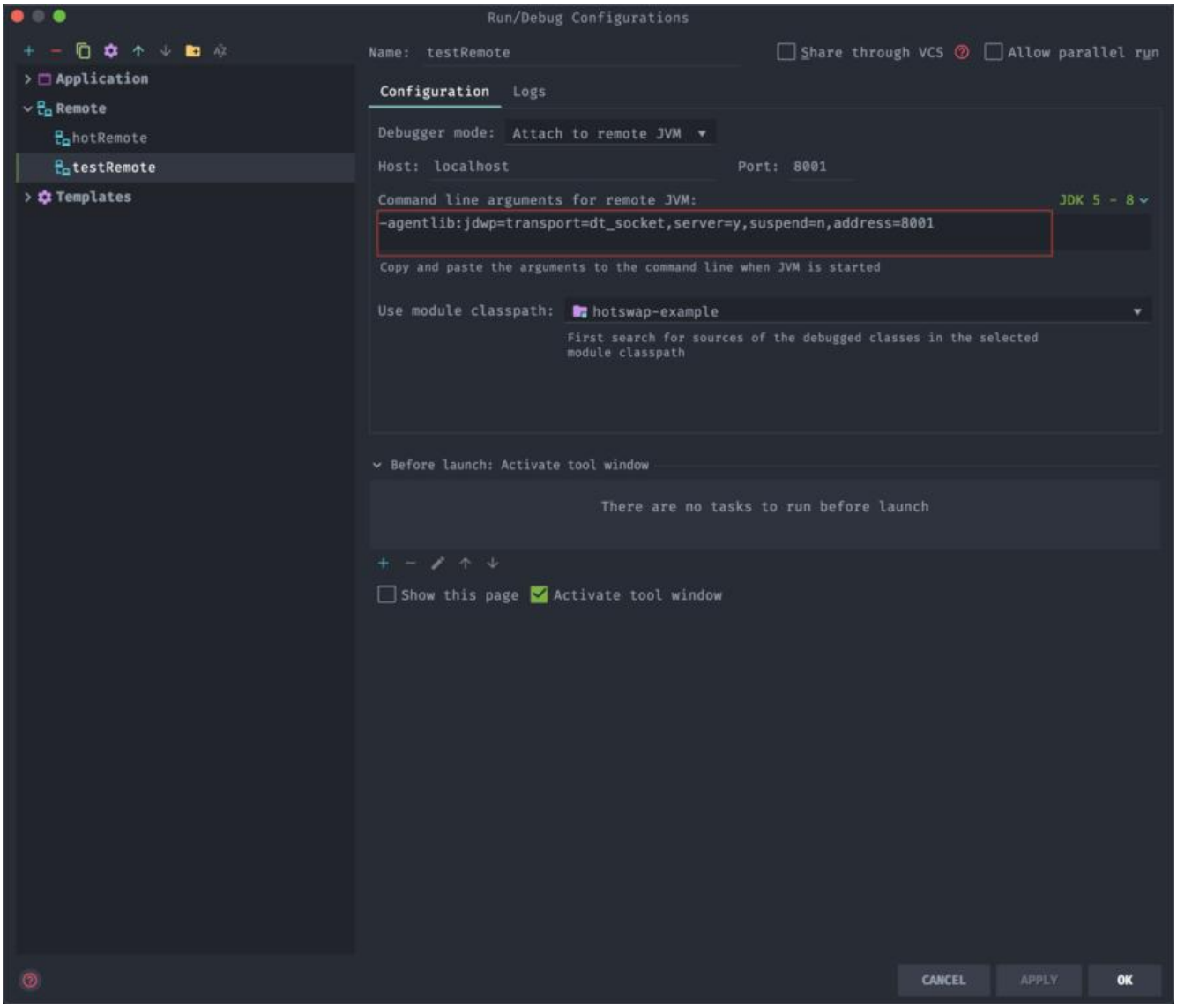
首先我们需要在 HelloService JVM 参数加入以下参数：

-Xrunjdwp:transport=dt_socket,server=y,address=8001

此时程序将会被阻塞，直到远程调试程序连接上 8001 端口，输出如下：

然后在 Agent-main 这个工程增加一个 remote 调试。

原文链接：手把手教你实现热更新功能，带你了解 Arthas 热更新背后的原理



图中参数如下:

-agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=8001

在 Agent-main 工程打上断点，运行远程调试， HelloService 程序将会被启动。

最后在命令行窗口运行 Agent-main 程序，远程调试将会暂停到相应断点处，接下来调试就跟普通 Debug 模式一样，不再叙述。

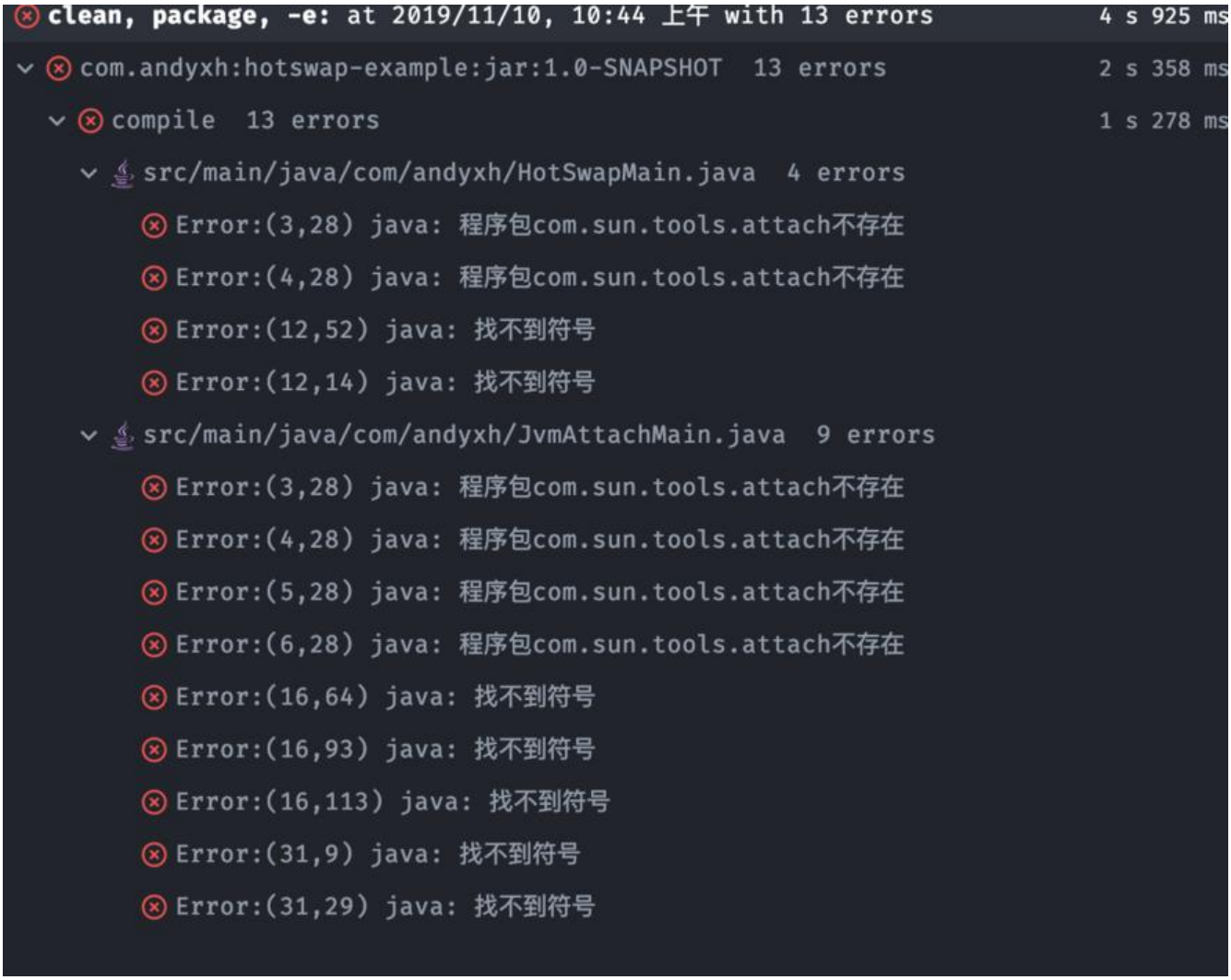
4.4、相关问题

由于 Attach API 位于 tools.jar 中，而在 JDK8 之前 tools.jar 与我们常用JDK jar 包并不在同一个位，所以编译与运行过程可能找不到该 jar 包，从而导致报错。

如果 maven 编译与运行都使用 JDK9 之后，不用担心下面问题。

maven 编译问题

maven 编译过程可能发生如下错误。



解决办法为在 pom 下加入 tools.jar 。

```
<dependency>
  <groupId>jdk.tools</groupId>
  <artifactId>jdk.tools</artifactId>
  <scope>system</scope>
  <version>1.6</version>
  <systemPath>${java.home}/../lib/tools.jar</systemPath>
</dependency>
```

或者使用下面依赖。

```
<dependency>
  <groupId>com.github.olivergondza</groupId>
  <artifactId>maven-jdk-tools-wrapper</artifactId>
  <version>0.1</version>
  <scope>provided</scope>
  <optional>true</optional>
</dependency>
```

程序运行过程 tools.jar 找不到

运行程序时抛出 java.lang.NoClassDefFoundError，主要原因还是系统未找到 tools.jar 导致。

```
$ java -jar hotswap-jdk.jar 3903  /tmp/HelloService.class
Error: A JNI error has occurred, please check your installation and try again
Exception in thread "main" java.lang.NoClassDefFoundError: com/sun/tools/attach/AttachNotSupportedException
        at java.lang.Class.getDeclaredMethods0(Native Method)
        at java.lang.Class.privateGetDeclaredMethods(Class.java:2701)
        at java.lang.Class.privateGetMethodRecursive(Class.java:3048)
        at java.lang.Class.getMethod0(Class.java:3018)
        at java.lang.Class.getMethod(Class.java:1784)
        at sun.launcher.LauncherHelper.validateMainClass(LauncherHelper.java:544)
        at sun.launcher.LauncherHelper.checkAndLoadMain(LauncherHelper.java:526)
Caused by: java.lang.ClassNotFoundException: com.sun.tools.attach.AttachNotSupportedException
        at java.net.URLClassLoader.findClass(URLClassLoader.java:381)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:424)
        at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:331)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:357)
        ... 7 more
```

在运行参数加入 -Xbootclasspath/a:${java_home}/lib/tools.jar，完整运行命令如下：

```
$ java  -Xbootclasspath/a:$JAVA_HOME/lib/tools.jar  -jar hotswap-jdk.jar 3903  /tmp/HelloService.class
当前需要热更新 jvm pid 为 3903
更换 class 绝对路径为 /tmp/HelloService.class
当前热更新工具 jar 路径为 /Users/andy.xu/Documents/GitHub/hotswap-example/target/hotswap-jdk.jar
```

4.5、热更新存在一些限制

并不是所有改动热更新都将会成功，当前使用 Instrumentation#redefineClasses 还是存在一些限制我们仅只能修改方法内部逻辑，属性值等，不能添加，删除方法或字段，也不能更改方法的签名或继关系。

五、彩蛋

写完热更新代码，收到一封系统邮件提示 xxx bug 待修复。恩，说好的少提 Bug 呢 o(╥﹏╥)o。

六、帮助

1.深入探索 Java 热部署

2.Instrumentation 新功能