## 安装

（一）安装沙箱

1、下载最新版本的JVM-SANDBOX：wget http://ompc.oss-cn-hangzhou.aliyuncs.com/jvm-sandbox/release/sandbox-stable-bin.zip

2、解压：unzip sandbox-stable-bin.zip

3、安装：./install-local.sh -p /export/opt/app/jvm-sandbox (该步骤可省略)

安装成功：

VERSION=1.3.3

PATH=/export/opt/app/jvm-sandbox/sandbox

install sandbox successful.

（二）启动沙箱（挂载目标应用）

1、attach方式启动：./sandbox.sh -p pid

a、可以在不重启目标JVM的情况下完成沙箱的植入。启动成功，沙箱已经顺利植入了目标JVM中，并且打开了HTTP端口55756，完成系统模块sandbox-mgr-module.jar 的加载。

2、agent方式启动

a、沙箱工作在应用代码加载之前，或者一次性渲染大量的类、加载大量的模块，此时如果用ATTACH方式加载，可能会引起目标JVM的卡顿或停顿（GC），这就需要启用到AGENT的启动方式。

b、JVM启动参数中增加上：-javaagent:/export/opt/app/jvm-sandbox/sandbox/lib/sandbox-agent.jar

例：nohup java -javaagent:/export/opt/app/jvm-sandbox/sandbox/lib/sandbox-agent.jar -jar sunxj-deploy-1.0-SNAPSHOT.jar >/export/opt/app/runningApp/nohup.log 2>&1 &

c、这样沙箱将会伴随着JVM启动而主动启动并加载对应的沙箱模块。

（三）卸载沙箱

./sandbox.sh -p pid -S

./sandbox.sh -p 33342 -S

三、沙箱配置

为了更好地使用JVM-Sandbox和管理模块，配置上做了一些约定和规范。

（一）沙箱结构介绍

1、./sandbox/bin/sandbox.sh：沙箱的客户端脚本，用于启动、管理沙箱

2、./sandbox/cfg/：沙箱配置文件存放目录

a、./sandbox/cfg/version：存放沙箱容器的版本号

b、./sandbox/cfg/sandbox.properties：存放沙箱容器的配置信息，配置文件只会在沙箱容器启动的时候加载一次。

c、./sandbox/cfg/sandbox-logback.xml：日志配置，日志文件默认写入到${HOME}/logs/sandbox/sandbox.log文件中

3、./sandbox/lib/：沙箱主程序的库包目录

sandbox-agent.jar：沙箱启动代理

sandbox-core.jar：沙箱内核

sandbox-spy.jar：沙箱间谍库，用于提供插桩埋点的间谍类

4、运行时文件：沙箱启动后将会创建一个隐藏文件${HOME}/.sandbox.token，这个文件将完成目标JVM进程和沙箱客户端进程一些信息的交互.

5、沙箱拥有两个加载模块的目录，用途各自不一

a、./sandbox/module/：沙箱系统模块目录，由配置项system\_module进行定义。用于存放沙箱通用的管理模块，比如用于沙箱模块管理功能的module-mgr模块，未来的模块运行质量监控模块、安全校验模块也都将存放在此处，跟随沙箱的发布而分发。系统模块不受刷新(-f)、\*\*强制刷新(-F)功能的影响，只有容器重置(-R)\*\*能让沙箱重新加载系统模块目录下的所有模块。

b、${HOME}/.sandbox-module/：沙箱用户模块目录，由sandbox.properties的配置项user\_module进行定义，默认为${HOME}/.sandbox-module/。一般用于存放用户自研的模块。自研的模块经常要面临频繁的版本升级工作，当需要进行模块动态热插拔替换的时候，可以通过\*\*刷新(-f)或强制刷新(-F)\*\*来完成重新加载。

c、./sandbox/sandbox-module ：沙箱用户模块目录

（二）沙箱模块

1、所有的沙箱模块都可以被设计成为热插拔

2、一个JAR包下可以申明多个模块，模块需要符合Java SPI规范，要求

a、必须拥有publish的无参构造函数

b、必须实现com.alibaba.jvm.sandbox.api.Module接口

c、必须完成META-INF/services/com.alibaba.jvm.sandbox.api.Module文件中的注册（Java SPI规范要求）

d、也可以通过依赖sandbox-module-starter来简化以上操作。

<parent>

<groupId>com.alibaba.jvm.sandbox</groupId>

<artifactId>sandbox-module-starter</artifactId>

<version>1.2.0</version>

</parent>

3、同一个JAR包所声明的所有模块共享同一个ModuleJarClassLoader

4、模块一共有四种状态

a、加载：模块被沙箱正确加载，沙箱将会允许模块进行命令相应、代码插桩等动作

b、卸载：沙箱不会再看到该模块，之前给该模块分配的所有资源都将会被回收，包括模块已经侦听事件的类都将会被移除掉监听插桩，干净利落不留后遗症

c、激活：模块加载成功后默认是冻结状态，需要代码主动进行激活。模块只有在激活状态下才能监听到沙箱事件

d、冻结：模块进入到冻结状态之后，之前侦听的所有沙箱事件都将被屏蔽。需要注意的是，冻结的模块不会退回事件侦听的代码插桩，只有delete()、wathcing()或者模块被卸载的时候插桩代码才会被清理。

（三）配置文件解释

1、./cfg/sandbox.properties。如图：

配置文件只会在沙箱第一次启动的时候加载，刷新(-f)、\*\*强制刷新(-F)和重置(-R)都不会让配置文件重新生效。如果希望配置文件重新生效，需要关闭(-S)\*\*容器，重新再次加载



四、命令说明

1、-p：指定目标JVM进程号

./sandbox.sh -p 10043

2、-l：列出目标JVM沙箱中已经加载的模块

./sandbox.sh -p 10043 -l

3、-F：强制刷新用户模块。首先卸载掉所有已加载的用户模块，然后再重新进行加载。当任何一个模块加载失败时，忽略该模块，继续加载其他可加载的模块

4、-f：刷新用户模块。与强制刷新用户模块不同的地方是，普通刷新会遍历用户模块下所有发生改变的模块文件，当且仅对发生变化的文件进行重新加载操作。

5、-R：沙箱模块重置。沙箱模块重置的时候将会强制刷新所有的模块，包括用户模块和系统模块。但sandbox.properties不会被重新加载

./sandbox.sh -p 10043 -R

6、-u：卸载指定模块。卸载指定模块，支持通配符表达式子。卸载模块不会区分系统模块和用户模块。例如：./sandbox.sh -p 4321 -u 'debug-module'

7、-a：激活模块。模块激活后才能收到沙箱事件

8、-A：冻结模块。模块冻结后将感知不到任何沙箱事件，但对应的代码插桩还在。

9、-m：查看模块详细信息。模块名需要精确匹配，不支持通配符。如图：



10、-d：模块自定义命令

挂载到对应进程：./sandbox.sh -p pid -d 'my-sandbox-module/addLog' 【my-sandbox-module/addLog】

@Information(id = "my-sandbox-module")// 模块名,在指定挂载进程后通过-d指定模块,配合@Command注解来唯一确定方法【标记在类上】

@Command("addLog")// 模块命令名【标记在方法上】

通过指令查看是否挂载成功：./sandbox.sh -p pid -l

11、-S: 卸载沙箱

./sandbox.sh -p 10043 -S

## sandbox-debug-module

沙箱分发包中自带了实用工具的例子./example/sandbox-debug-module.jar，代码在沙箱的sandbox-debug-module模块，也是非常不错的实用工具和学习资料。

| **例子** | **例子说明** |
| --- | --- |
| [DebugWatchModule.java](https://github.com/alibaba/jvm-sandbox/blob/master/sandbox-debug-module/src/main/java/com/alibaba/jvm/sandbox/module/debug/DebugWatchModule.java) | 模仿GREYS的watch命令 |
| [DebugTraceModule.java](https://github.com/alibaba/jvm-sandbox/blob/master/sandbox-debug-module/src/main/java/com/alibaba/jvm/sandbox/module/debug/DebugTraceModule.java) | 模仿GREYES的trace命令 |
| [DebugRalphModule.java](https://github.com/alibaba/jvm-sandbox/blob/master/sandbox-debug-module/src/main/java/com/alibaba/jvm/sandbox/module/debug/DebugRalphModule.java) | 无敌破坏王，故障注入（延时、熔断、并发限流、TPS限流） |
| [LogExceptionModule.java](https://github.com/alibaba/jvm-sandbox/blob/master/sandbox-debug-module/src/main/java/com/alibaba/jvm/sandbox/module/debug/LogExceptionModule.java) | 记录下你的应用都发生了哪些异常 $HOME/logs/sandbox/debug/exception-monitor.log |
| [LogServletAccessModule.java](https://github.com/alibaba/jvm-sandbox/blob/master/sandbox-debug-module/src/main/java/com/alibaba/jvm/sandbox/module/debug/LogServletAccessModule.java) | 记录下你的应用的HTTP服务请求 $HOME/logs/sandbox/debug/servlet-access.log |

并发控制

-d 'debug-ralph/c-limit?class=<CLASS>&method=<METHOD>&c=<CONCURRENT>'

速率控制

-d 'debug-ralph/r-limit?class=<CLASS>&method=<METHOD>&c=<RATE>'

注入异常

-d 'debug-ralph/wreck?class=<CLASS>&method=<METHOD>&type=<EXCEPTION-TYPE>'

注入延时

-d 'debug-ralph/delay?class=<CLASS>&method=<METHOD>&delay=<DELAY(ms)>'