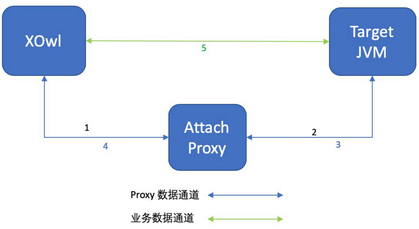
涉及功能

- Java Agent、JVMTI Agent 改造
- MXBean 相关功能使用

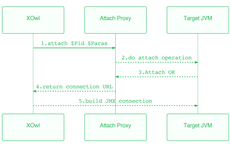
架构

概览

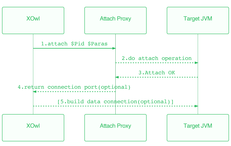


时序

Build MXBean Connection



Load Agent Or Build Connection



XOwl 与 Attach Proxy 通信协议设计

二者之间的通信全部采用同步阻塞的方式进行，换句话说，XOwl 执行调用 Proxy 执行某个操作时，会一直等到 Proxy 执行完毕并返回。

XOwl to Attach Proxy

原则上，Attach Proxy 只是透传从 XOwl 受到的命令并执行，为了确保扩展性，特设计相关的通信协议，方便后续的版本维护和功能扩展。

	version	type	pid	data_len	data
类型	int	int	int	int	byte[]
备注	协议版本	操作类型	目标进程 id	数据长度	数据

type 设计

为了简化设计，对 type 做了个特殊设计，当 type 为 -1 时，表示 Attach Proxy 在执行某项操作时失败了，失败的原因保持在 data 字段。其他 type 都表示某个具体的操作执行成功。

type	备注
1	MXBean Connection
2	Load Java Agent
3	Load CPU Agent
4	Load Thread/Memory/Common Agent

Attach Proxy to XOwl

由于 Attach Proxy 要向 Xowl 汇报执行情况，所以添加了一个返回值字段，其他字段均相同

	version	ret_value	type	pid	data_len	data
类型	int	int	int	int	int	byte[]
备注	返回值	协议版本	操作类型	目标进程 id	数据长度	数据

ret_value 设计

返回值用于告知 XOwl 执行某个命令的具体结果，目前暂定如下，后续有需求再扩展

ret_value	备注
-1	Execution Error, the specific error may unknow
0	Execution OK
1	Execution Error, process is not exist
2	Execution Error, operation is timeOut

XOwl 端 AttachManager 设计

获取目标进程可执行文件的全路径

目前针对 Linux 和 AIX 获取目标进程可执行文件全路径的方式有所不同，所以需要采取不同实现

Linux

可以通过使用 Java 来读取文件"/proc/$pid/exe"的链接来获取全路径，代码如下

```
private static String getExeFileOnUnixOnLinux(String pid) throws IOException {
        final String exe = File.separator + "proc" + File.separator + pid + File.separator + "exe";
        File exeFile = new File(exe);
        if (exeFile.exists()) {
            return exeFile.getCanonicalPath();
        } else {
            throw new RuntimeException("Can not find files " + exe);
        }
    }
```

AIX

AIX 的实现有些复杂，直接从 Java 层面来获取好像还办不到，因为 AIX 系统并没有和 Linux 下的"/proc/$pid/exe"文件，需要写个 Native 本地方法才能获取到，Native 的代码大概如下：

```
#include <assert>
#include <iostream>
#include <string>
#include <unistd.h>
#include <procinfo.h>
#include <stdio.h>
int main(int argc, char** argv) {
    struct procentry64 processInfo;
    const pid_t thisPid = getpid();
    pid_t retPid = pid_t();
    char argsBuffer[1024];
    char destBuffer[1024];
    memset (argsBuffer, 0, sizeof(argsBuffer));
    sprintf(argsBuffer, 1024, "/proc/%d/cwd",thisPid);
    printf("cwd: %s \n", argsBuffer);
    readlink(argsBuffer,destBuffer,1024);
    printf("cwd: %s \n", destBuffer);

    while (getprocs64(&processInfo, sizeof (processInfo), 0, 0, &retPid, 1) > 0) {
        if (static_cast<pid_t>(processInfo.pi_pid) != thisPid){
        continue;
        }

        int retValue = getargs (&processInfo, sizeof(processInfo),
                                argsBuffer[1], sizeof(argsBuffer));
        assert (retValue == 0);
        break;
    }

    const std::string exeName = argsBuffer;
    assert (exeName.size() < sizeof(argsBuffer));
    // 设置把 destbuffer 和 exeName 合并起来那为全路径，但是这里可以优化
    // 如果 exeName 返回的本来就是全路径，那么 destbuffer 就可以不用拼接了
    // 如果 exeName 返回的是相对路径，那么不需要把 destbuffer 加在其前面，以组成全路径
    std::cout << "exeName = " << destBuffer << exeName << std::endl;
    return 0;
}
```

组装相关参数，通过协议发送到 Attach Proxy

这个功能主要是把要实行的程序、命令行参数通过协议组装，同时调用 attach proxy。
代码略。。。

等待 Proxy 执行完毕，获取返回值

因为 XOwl 调用 Proxy 是同步的，所以可以同步获取 Proxy 的返回值，然后根据返回协议来反序列化相关数据，呈现给 XOwl 调用方

Attach Proxy 端设计

功能

1. 解析由 Xowl 的 AttachManager 发送的数据；
2. 根据命令内容执行不同的操作；
3. 返回执行结果给 AttachManager。

实现

1. CmdParser
   解析命令入参数据
2. CmdExecutor（接口）
   根据协议中的 type 字段，来实现特定的实现类进行相关的操作，然后回传特定协议的执行结果给 XOwl。