JVM Greys

Greys对正在执行的进程进行异常诊断，其可以用来定位线上或者开发中遇到的各种性能、功能等问题，包括：

* 在运行过程中查看方法入参、返回值，或者方法执行的堆栈时间
* 在Jar冲突时定位类加载来自哪个Jar包
* 打印log日志

Greys是一个C/S架构的程序，所以当Client访问到Server时，Server会维护一个session（会话），以及session的心跳、超时机制。事务（Tx）机制则是建立在session的基础上，所有的命令交互都会创建一个事务，并且产生对应的队列进行输出缓冲。

# 1.Greys的安装

1. 下载安装文件

*wget* [*http://ompc.oss.aliyuncs.com/greys/release/greys-1.7.6.4-bin.zip*](http://ompc.oss.aliyuncs.com/greys/release/greys-1.7.6.4-bin.zip)

1. 解压并安装

*tar -zxvf greys-1.7.6.4-bin.zip*

*cd greys*

*sh install-local.sh*

1. 对目标进程进行Attach Agent，目标进程为28300

*$ ./ga.sh 28300*

*greys attach to target(28300) success*

查看端口：

*netstat –anp|grep 28300*

*0 0.0.0.0:3658 0.0.0.0:\* LISTEN 28300/java*

不指定，默认使用3685。

1. 客户端操作服务进程

*$ ./greys.sh 28300*

*\_*

*\_\_\_\_ \_\_\_\_ \_\_\_\_\_ \_ \_ \_\_\_ \_\_\_\_\_ \_\_\_\_\_ \_\_\_\_ \_\_\_\_\_ \_| |\_ \_\_\_ \_\_\_\_ \_ \_*

*/ \_ |/ \_\_\_) \_\_\_ | | | |/\_\_\_|\_\_\_\_\_|\_\_\_\_ | \_ \(\_\_\_\_ (\_ \_) \_ \| \| | | |*

*( (\_| | | | \_\_\_\_| |\_| |\_\_\_ | / \_\_\_ | | | / \_\_\_ | | || |\_| | | | | |\_| |*

*\\_\_\_ |\_| |\_\_\_\_\_)\\_\_ (\_\_\_/ \\_\_\_\_\_|\_| |\_\\_\_\_\_\_| \\_\_)\_\_\_/|\_|\_|\_|\\_\_ |*

*(\_\_\_\_\_| (\_\_\_\_/ (\_\_\_\_/*

*+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+*

*|v|e|r|s|i|o|n|:|1|.|7|.|6|.|4|*

*+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+*

*ga?>*

*ga?>*

目标Java进程ID（请确保执行当前执行命令的用户必须有足够的权限操作对应的Java进程） IP：目标服务器IP地址，当远程服务开启之后，其他人可以通过指定IP的形式加载到对应目标机器的Java进程中，从而实现远程协助。专门用于解决目标主机账号没有权限，但对方兄弟却非常需要你支援的时候。Greys允许多个用户同时访问，并且各自的命令不会相互干扰执行。 PORT：目标服务器端口号，设计端口号的初心则是希望解决同台机器上存在多个Java进程需要被Greys分析的情况，默认的端口号是3658，如果不做区分则会引起端口冲突。

1. 其他连接方式

* sudo支持

成熟的线上管理环境一般都不会直接开放JVM部署用户权限给你，而是通过sudo-list来控制和监控用户的越权操作。由于greys.sh脚本中会对当前用户的环境变量产生感知，所以需要加上-H参数

sudo -u admin -H ./greys.sh 12345

* TELNET的支持

Greys支持通过telnet来访问服务端，如果当你手头的机器没有安装Greys的客户端，你可以简单的通过telnet命令来进行访问。

telnet 10.232.12.113 3658

当然了，telnet命令和Greys自带的Console在使用友好度上还是有一定的差距，不过解决应急之需没有问题。

1. 使用，事务伴随着命令的生命周期而存在，命令分两种：

* **立即返回**

立即返回的命令定义是：敲下命令后Server端立即返回最终结果，后续无持续反馈信息，释放Client对输入的锁定，重新开放让用户输入信息，比如version、sc、sm等。

* **等待中止**

等待中止的命令则是需要用户主动输入Ctrl+D完成的命令中止操作。命令执行后无法立即返回最终结果，而是不断的将中间产生的输出源源不断的输出到客户端中，这种命令比如stack、monitor等。

当session关闭时，所有挂在session的事务也会立即被关闭。

7）greys的命令清单

|  |  |
| --- | --- |
| 命令 | 描述 |
| help | 查看命令的帮助文档，每个命令和参数都有很详细的说明 |
| sc | 查看JVM已加载的类信息 |
| sm | 查看已加载的方法信息 |
| monitor | 方法执行监控 |
| trace | 渲染方法内部调用路径，并输出方法路径上的每个节点上耗时 |
| ptrace | 强化版的trace命令，通过指定渲染路径，并可记录下路径中所有方法的入参、返回值；与tt命令联动 |
| watch | 方法执行数据观测 |
| tt | 方法执行数据的时空隧道，记录下指定方法每次调用的入参和返回信息，并能对这些不同的时间下调用进行观测 |
| stack | 输出当前方法被调用的调用路径 |
| js | 支持使用JavaScript脚本，支持CommonJS部分规范模块化 |
| version | 输出当前目标Java进程加载的Greys版本号 |
| quit | 退出greys客户端 |
| shutdown | 关闭greys服务端 |
| reset | 重置增强类，将被greys增强过的类全部还原 |
| session | 查看当前会话 |
| jvm | 查看当前JVM的信息 |

# 2.Greys的使用

## 2.1 命令使用概念

下面是Greys命令中使用的参数概念：

1. 表达式，表达式分两种：条件表达式和观察表达式

* 条件表达式，使用表达式表达TRUE或FALSE的场景，trace、stack、tt、watch命令都增加了条件表达式支持（使用greys内置的表达式解析引擎，识别OGNL语法）。

|  |  |  |
| --- | --- | --- |
| 条件表达式 | 预测结果 | 解析结果说明 |
| 1==1 | TRUE | 条件表达式为真 |
| true | TRUE | 条件表达式为真 |
| @@@ | FALSE | 非法的条件表达式 |
| params==null | FALSE | 条件表达式为假 |
| false | FALSE | 条件表达式为假 |
| 1!=1 | FALSE | 条件表达式为假 |

1. 观察表达式，用在使用表达式表达输出内容的场景，尤其watch和tt命令中，观察表达式非常重要。下面是观察表达式使用的例子：

字符串拼接：*clazz.name+”.”+method.name*

数字运算：*class.name.length()+method.name.length()*

1. 表达式核心变量，无论是匹配表达式、观察表达式，他们的核心判断变量都是围绕一个greys中通用通知对象Advice进行，Advise代码结构如下：

*public class Advice {*

*private final ClassLoader loader;*

*private final Class<?> clazz;*

*private final GaMethod method;*

*private final Object target;*

*private final Object[] params;*

*private final Object returnObj;*

*private final Throwable throwExp;*

*private final boolean isBefore;*

*private final boolean isThrow;*

*private final boolean isReturn;*

*// getter/setter*

*}*

变量的含义如下表：

|  |  |
| --- | --- |
| 变量名 | 描述 |
| loader | 调用类所在的ClassLoader |
| clazz | 调用类的Class引用 |
| method | 调用方法反射引用 |
| target | 本次调用类的实例 |
| params | 调用参数列表，这是一个数组，如果方法是无参方法则为空数组 |
| returnObj | 本地调用返回的对象，当前仅当isReturn==true成立时候有效，表明方法调用以正常返回的方式结束，当前方法无返回值则为null |
| throwExp | 调用抛出的异常，当前仅当isThrow==true成立时有效，表明方法调用是以抛出异常的方式结束 |
| isBefore | 辅助判断标记，当前的通知节点有可能是在方法一开始就通知，此时isBefore==true成立，同时isThrow==false和isReturn==false，因为在方法刚开始时，还无法确定方法调用将会如何结束。 |
| isThrow | 辅助判断标记，当前的方法调用以抛异常的形式结束 |
| isReturn | 辅助判断标记，当前的方法调用以正常返回的形式结束 |

所有变量都可以在表达式中直接使用，如果表达式中包含了不符合语法或者引入不在表格中的变量：

* 对条件表达式、检索表达式而言，则一律当成false来处理
* 对观察表达式而言，则放弃当前方法调用的处理（不输出）

1. 模式匹配，一些命令需要对类、方法进行模式匹配，目前仅支持\*和?两个通配符，正则表达式需要显示指定-E参数激活。举例如下：

* sc命令的正则表达式匹配

*sc com.apache.commons.lang.StringUtils*

*sc \*lang.StringUtils*

* 若想继续使用正则表达式匹配，需要显示-E参数激活

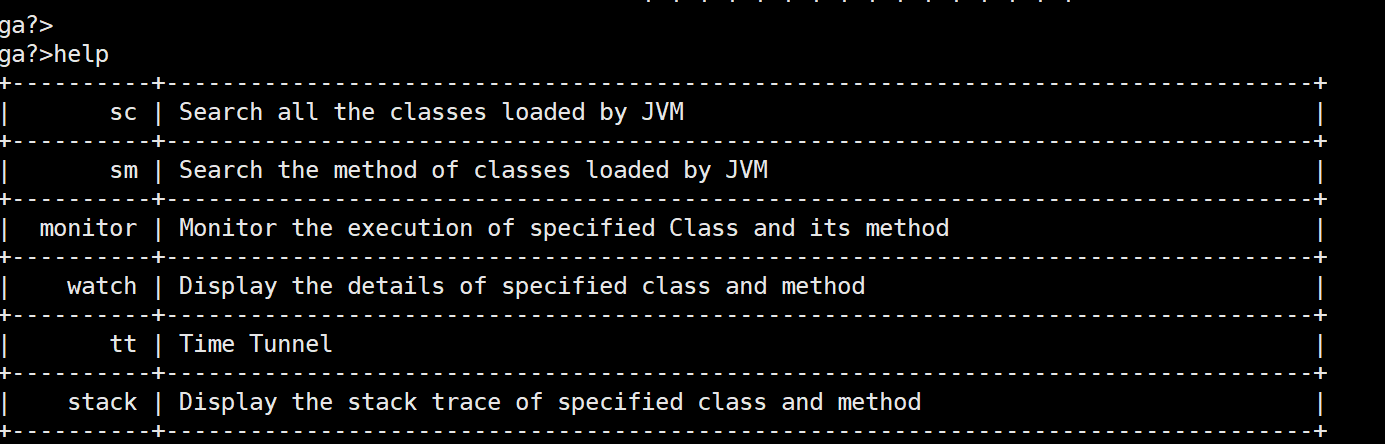
*sc -E com\.apache\.commons\.lang\.StringUtils*

*sc -E com\..\*StringUtils*

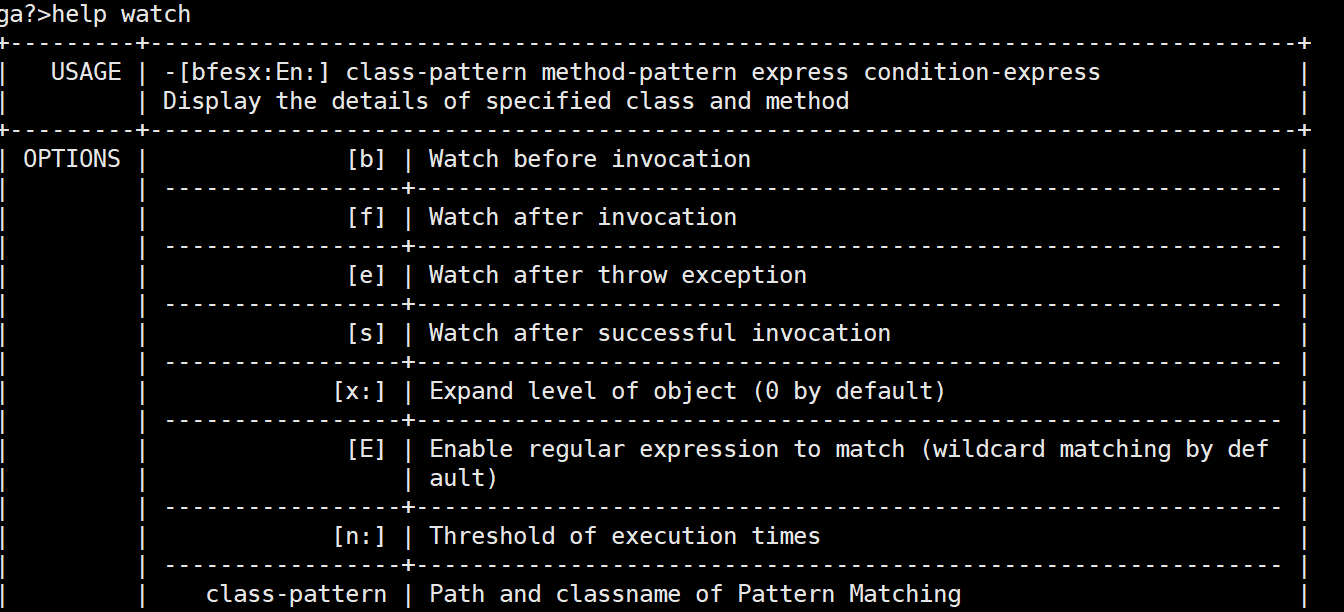
所有需要模式匹配命令都支持参数-E，分别是：sc、sm、stack、monitor、watch、tt、trace

## 2.2 help

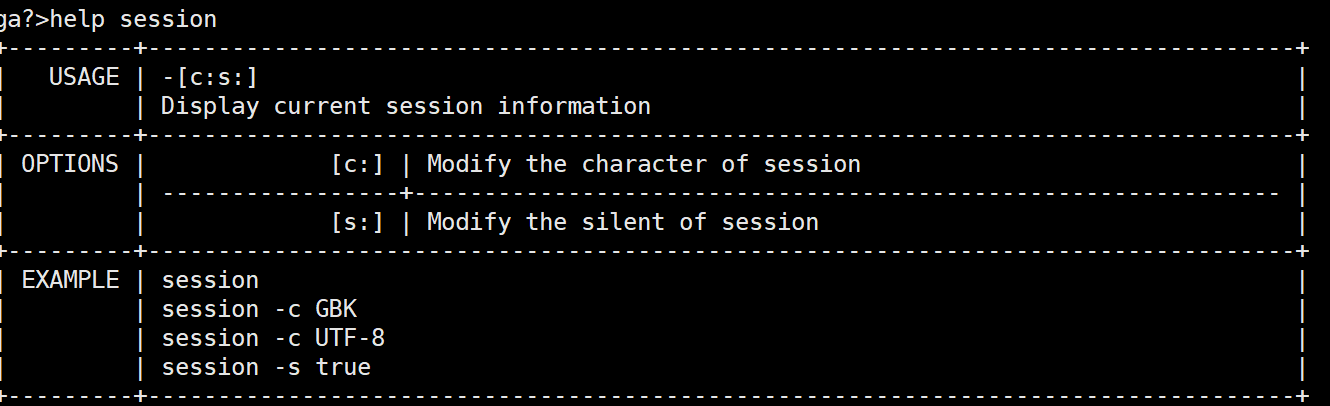
进入Greys的Console后，所有命令都可以通过help获取帮助，Greys会返回所有命令的大概用途介绍，如下图所示：



help命令同时也支持对其他命令的一个解释说明，比如help watch命令，greys将会返回watch命令的所有参数解释、用法介绍等详细信息。



帮助文档分成Usage,Options，Example三个区域，分别对应用途说明、参数类别和实际的例子，如下：

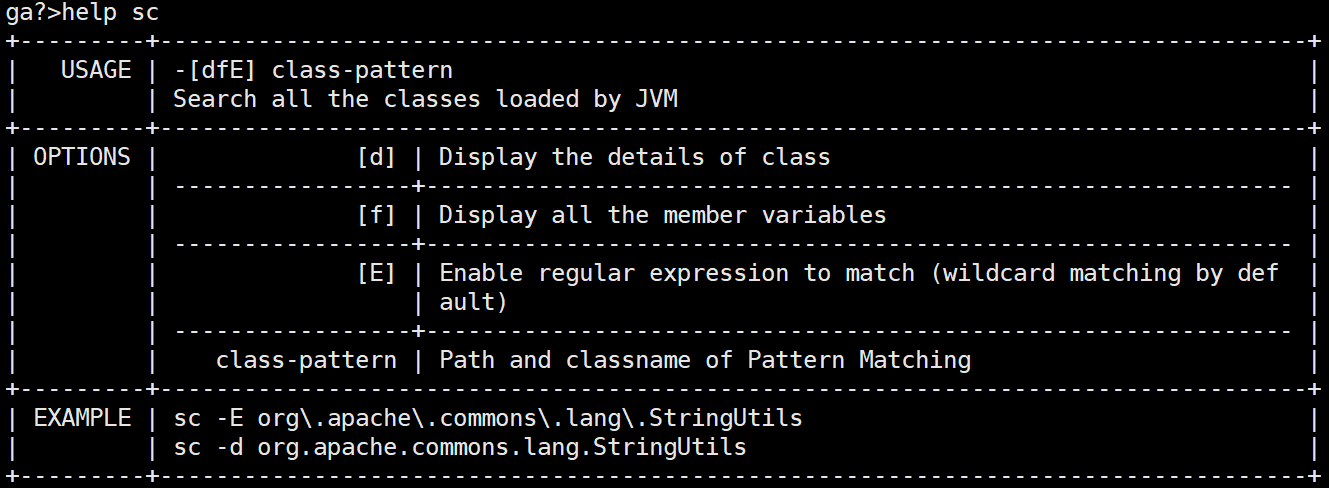


参数选项说明:

* []中的参数为选填项，比如[d]，意思是该命令可接受一个名词为d的选填参数，且不用参数值
* [:]中的参数则为选填，但是有值得参数，比如[c:]
* class-pattern/method-pattern，这两个参数为隐性参数，即在输入的时候不需要特意声明参数。class-pattern为类路径.类名词的表达式匹配，method-pattern则为方法名的表达式匹配。

## 2.3 sc

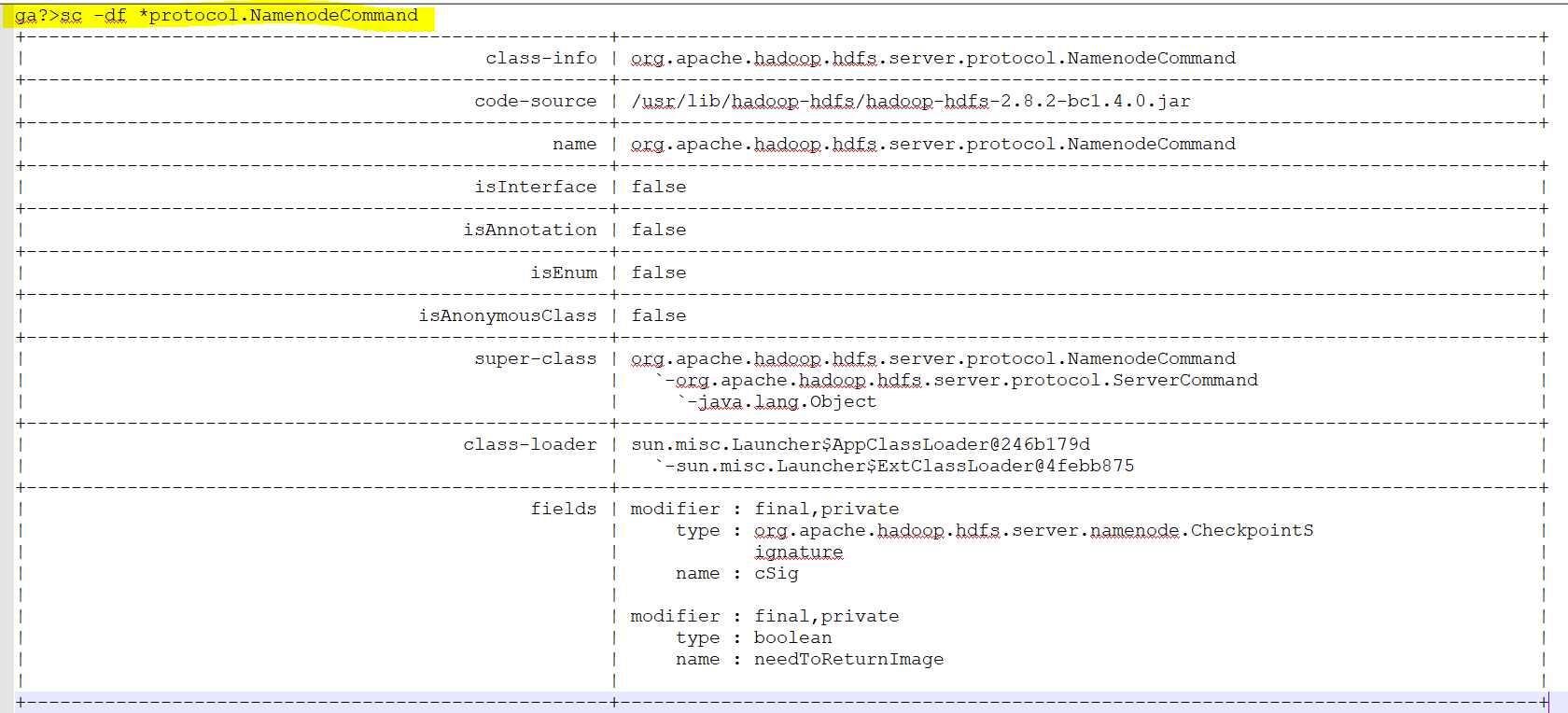
sc是Search-Class简写，搜索出所有已经加载到JVM中Class信息



1. 参数说明

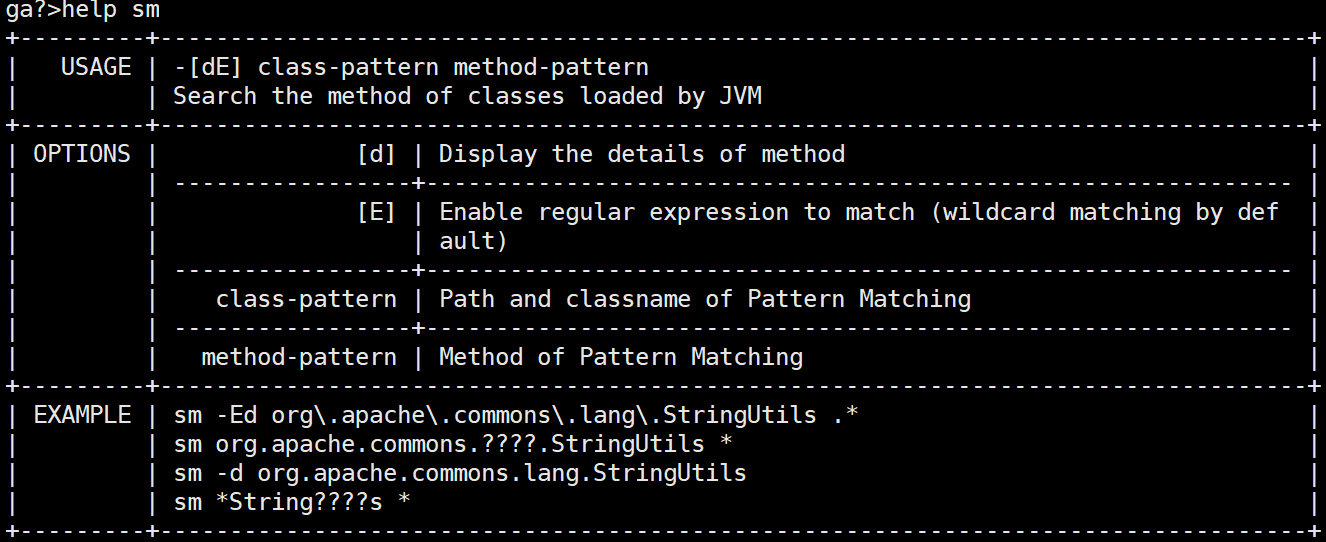
|  |  |
| --- | --- |
| 参数名称 | 参数说明 |
| class-pattern | 类名表达式匹配 |
| method-pattern | 方法名表达式匹配 |
| [d] | 输出当前类的详细信息，包含这个类所加载的原始文件来源、类的声明、加载的e2ClassLoader等详细信息，如果一个类被多个ClassLoader所加载，则会出现多次 |
| [f] | 输出当前类的成员变量信息 |
| [E] | 支持正则表达式匹配 |

2）使用举例



## 2.4 sm

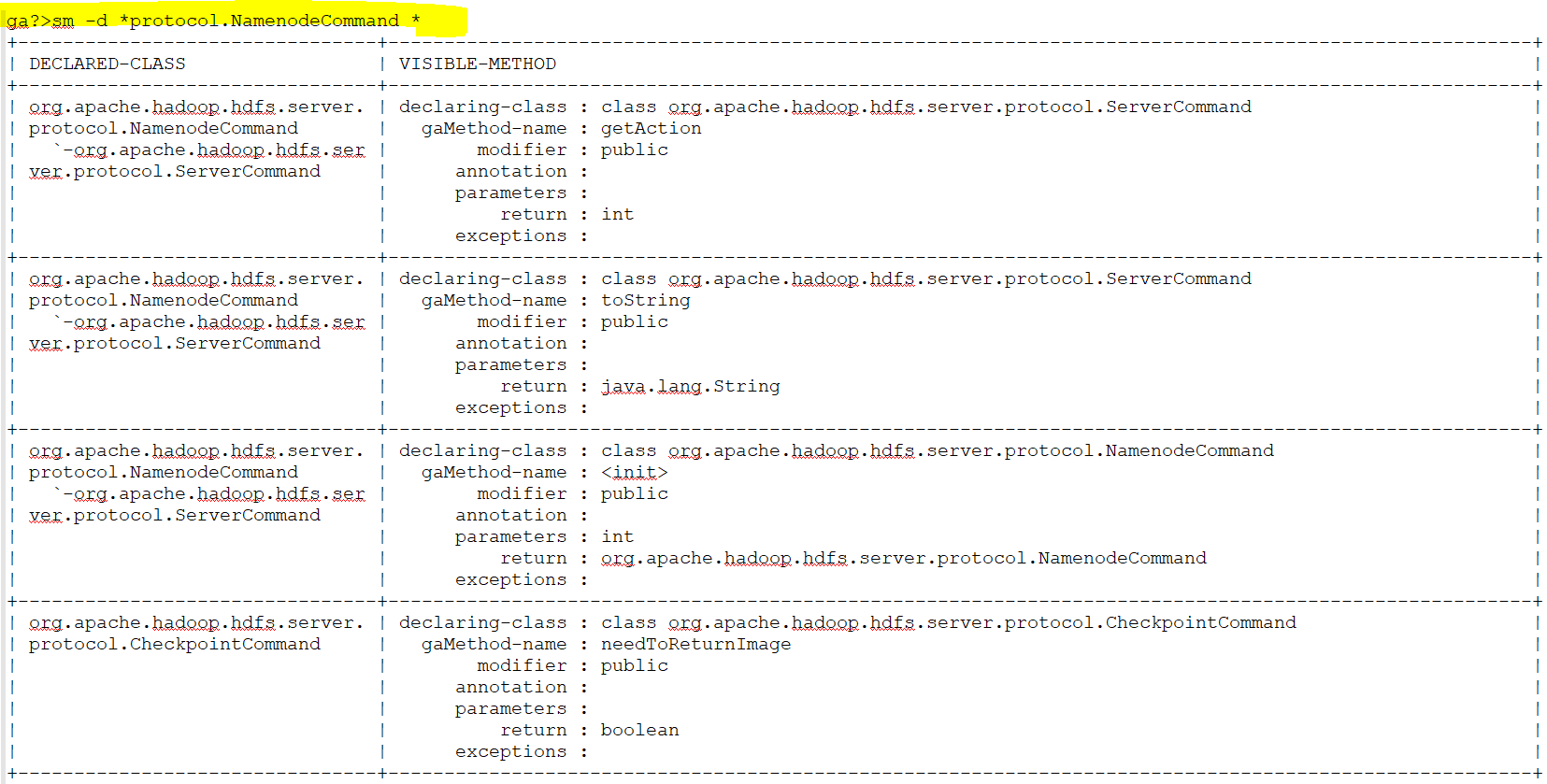
Search-Method的简写，这个命令能搜索所有已经加载了Class信息的方法信息



1. 参数说明

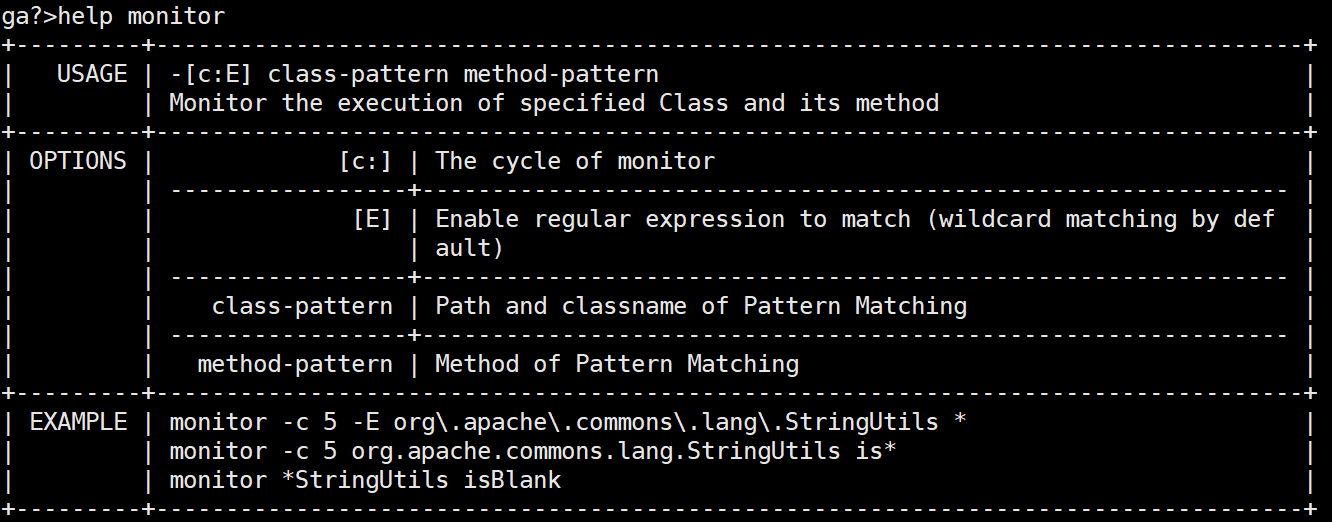
|  |  |
| --- | --- |
| 参数名称 | 参数说明 |
| class-pattern | 类名表达式匹配 |
| method-pattern | 方法名表达式匹配 |
| [d] | 展示每个方法的详细信息 |
| [E] | 支持正则表达式匹配 |

2）使用举例



## 2.5 monitor

对匹配class-pattern/method-pattern的类或者方法的调用进行监控，monitor命令是非实时返回命令，实时返回命令是输入之后立即返回，而非实时返回的命令则是不断等待目标Java进程返回信息，直到用户输入Ctrl+D为止。服务端是以任务的形式在后台跑任务，植入的代码随着任务的终止而被中断执行，所有任务关闭后，不会对原有性能产生太大的影响，而且原则上，任何Greys的命令也不会引起任何原有业务逻辑的改变。



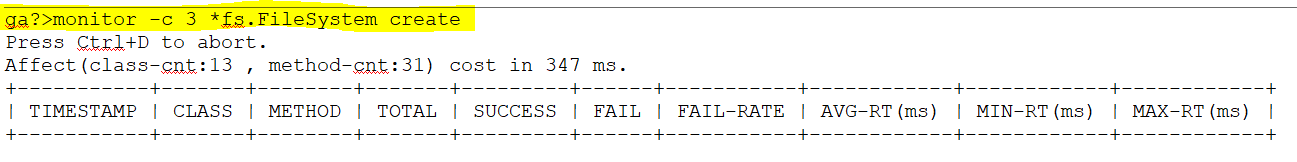
1. 监控维度说明

|  |  |
| --- | --- |
| 监控项 | 说明 |
| timestamp | 时间戳 |
| class | Java类 |
| method | 方法（构造方法、普通方法） |
| total | 调用次数 |
| success | 成功次数 |
| fail | 失败次数 |
| rt | 平均RT |
| fail-rate | 失败率 |

2）参数说明

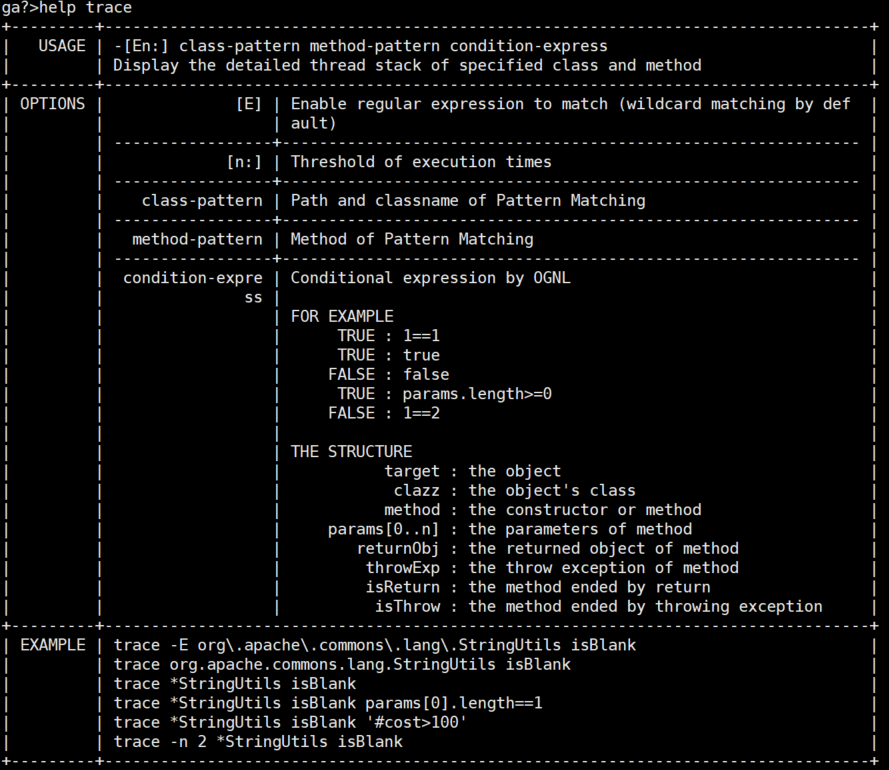
|  |  |
| --- | --- |
| 参数名称 | 参数说明 |
| [c:] | 统计周期，默认值为120秒 |

3）使用参考



## 2.6 trace

主动搜索class-pattern/method-pattern所渲染的方法调用路径，渲染和统计整个调用链路上所有性能开销和追踪调用链路

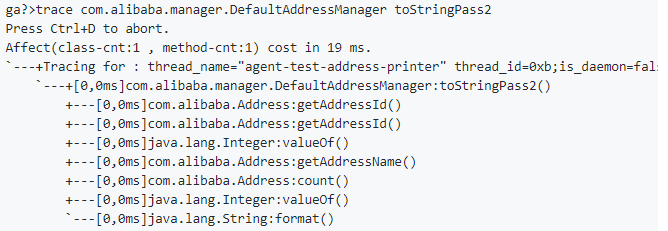


1）参数说明

|  |  |
| --- | --- |
| 参数名称 | 参数说明 |
| class-pattern | 类名表达式匹配 |
| method-pattern | 方法名表达式匹配 |
| condition-express | 条件表达式 |
| [n:] | 命令执行次数 |
| [E] | 支持正则表达式匹配 |

Trace能帮助开发者定位和发现RT高而导致的性能问题缺陷，但每一次只能跟踪一级方法的调用链路。

2）使用参考



[10,1ms]的含义，10所代表的含义是：当前节点的整体耗时；1的含义是：当前节点在当前步骤的耗时；两者之间用逗号分割，单位为毫秒。

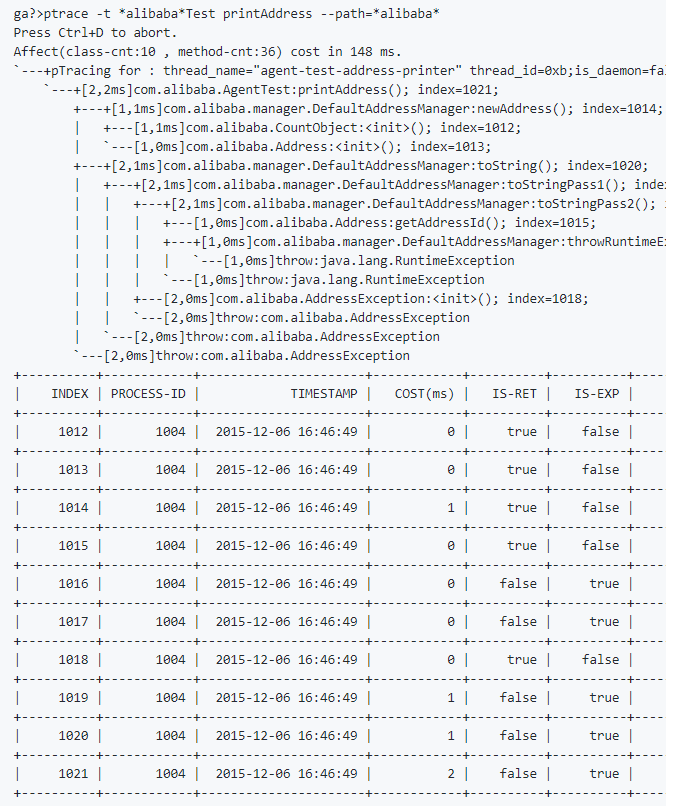
## 2.7 ptrace

Trace命令的强化版，通过指定渲染路径来完成对方法执行路径的渲染过程，命令能主动搜索tracing-path-pattern所渲染的路径，渲染和统计整个调用链路上所有性能开销和追踪调用链路。

1）参数说明

|  |  |
| --- | --- |
| 参数名称 | 参数说明 |
| class-pattern | 类名表达式匹配 |
| method-pattern | 方法名表达式匹配 |
| condition-express | 条件表达式 |
| [n:] | 命令执行次数 |
| [E] | 支持正则表达式匹配 |
| [t] | 记录渲染路径上所有方法的入参和返回值，记录下返回值可以与tt命令联动 |
| [path:] | 渲染路径表达式匹配，该参数可多次使用 |
| [Epath:] | 正则表达式渲染路径表达式匹配，该参数可多次使用 |

2）使用例子



## 2.8 watch

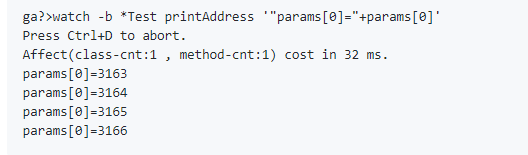
观察指定方法的调用情况，能观察的范围：返回值、抛出异常、入参，通过编写OGNL表达式进行对对应变量的查看

1）参数说明

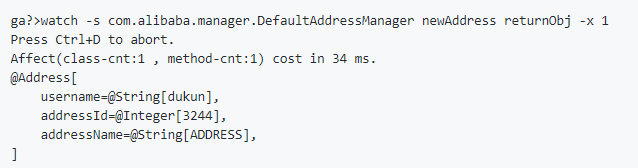
|  |  |
| --- | --- |
| 参数名称 | 参数说明 |
| class-pattern | 类名表达式匹配 |
| method-pattern | 方法名表达式匹配 |
| condition-express | 条件表达式 |
| express | 观察表达式 |
| [b] | 在方法调用之前观察 |
| [e] | 在方法异常之后观察 |
| [s] | 在方法返回之后观察 |
| [f] | 在方法结束之后观察 |

其中观察表达式主要有OGNL表达式组成，可以写成params[0]+”$”+target，观察的维度也比较多，主要体现在参数advice数据结构上。

1. 使用参考



有一个参数-x，决定输出结果的层次遍历输出对象，当加上这个参数以后，greys会将输出的对象按照层级进行展开，-x 1表明展开第一层级



## 2.9 tt

TimeTunnel的简写，记录当时方法调用所有入参和返回值、抛出的异常会对整个问题的思考和判断有帮助

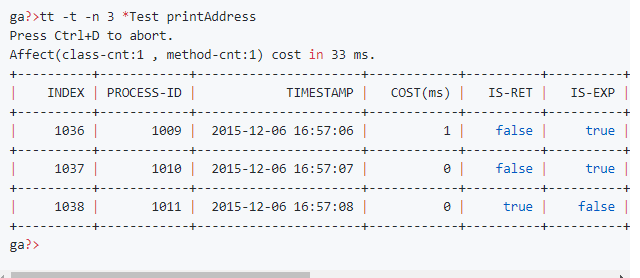
1. 命令参数

|  |  |
| --- | --- |
| 参数名称 | 参数说明 |
| -t | 这个方法记录类的方法的每次执行情况 |
| -n 3 | 当你执行一个调用量不高的方法时可能你还能有足够的时间用CTRL+D中断tt命令记录的过程，但如果遇到调用量非常大的方法，瞬间就能将你的JVM内存撑爆。此时你可以通过-n参数指定你需要记录的次数，当达到记录次数时greys会主动中断tt命令的记录过程，避免人工操作无法停止的情况 |
| -s | 筛序调用信息 |
| -i | 检索对应编号的详细信息 |

2）输出结果说明

|  |  |
| --- | --- |
| 参数名称 | 参数说明 |
| INDEX | 时间片段记录编号，每一个编号代表依次调用 |
| PROCESS-ID | 过程编号，认为同一线程的一次同步调用为一个过程 |
| TIMESTAMP | 方法执行的本机时间，记录这个时间片段所发生的本机时间 |
| COST(ms) | 方法执行耗时 |
| IS-RET | 是否以正常返回的形式结果 |
| IS-EXP | 方法是否以抛异常的形式结束 |
| OBJECT | 执行对象的hasCode |
| CLASS | 执行的类名 |
| METHOD | 执行的方法名 |

3）基本用法



4）高级用法1-条件表达式

使用条件表达式，解决特定参数、区分重载等方法

* 解决方法重载

*tt -t \*Test print params[0].length==1*

*tt -t \*Test print 'params[1].class == Integer.class'*

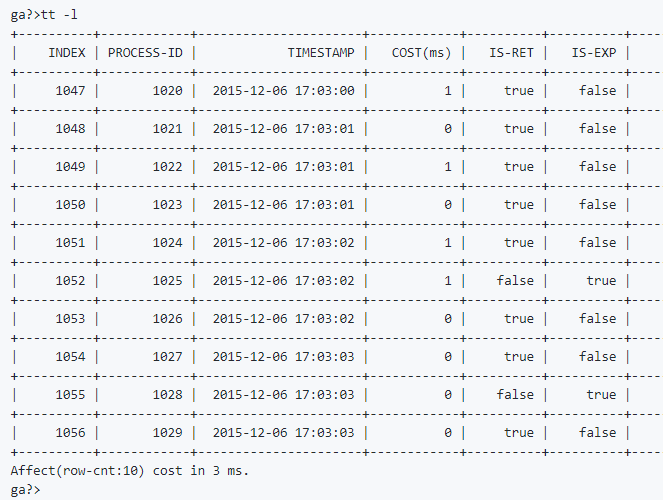
通过指定参数个数的形式解决不同的方法签名

* 解决指定参数

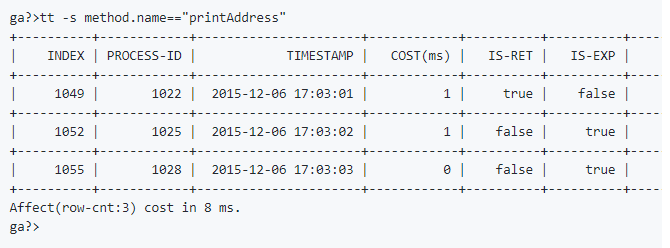
*tt -t \*Test print params[0].mobile=="13989838402"*

5）高级用法2-检索调用记录

当用tt记录多时间片段后，希望筛选需要的时间片段，对现有记录进行检索

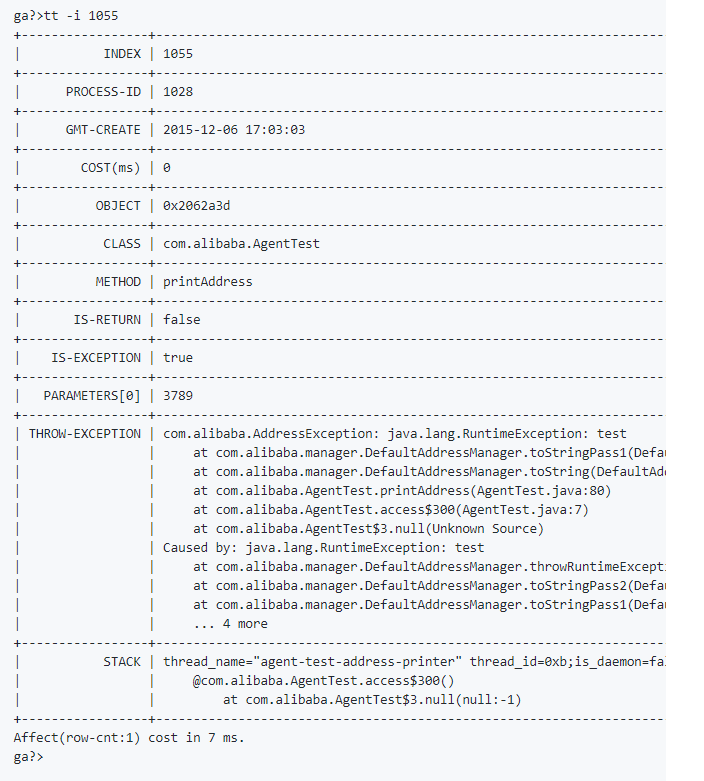


从上面记录筛选printAddress方法的调用信息



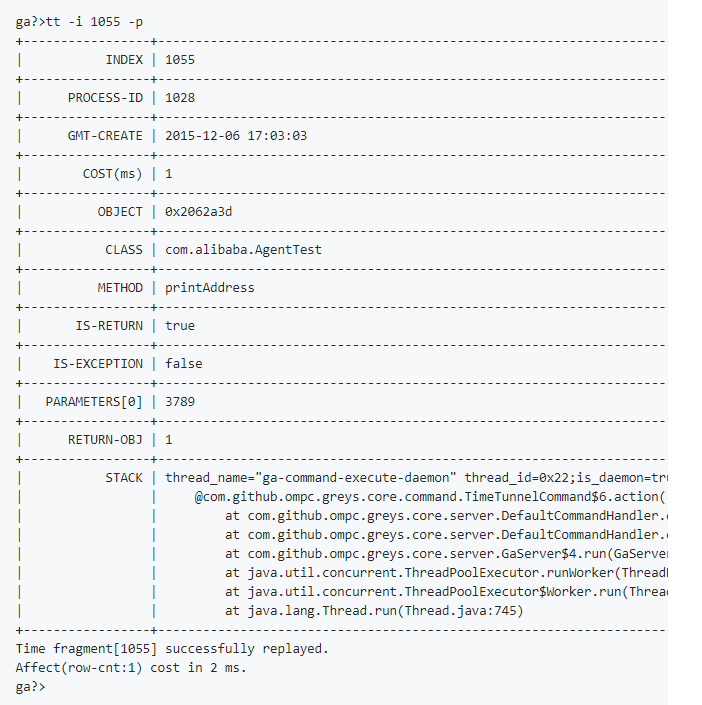
6）高级调用3-查看调用信息

对具体某个时间片信息而言，可以通过-i参数跟对应INDEX编号查看详细信息



7）高级调用4-重做一次调用(调用重放)

重新出发一次调用，tt命令保持了当时调用的所有现场信息，可以主动对一个INDEX编号的时间片自主发起一次调用。



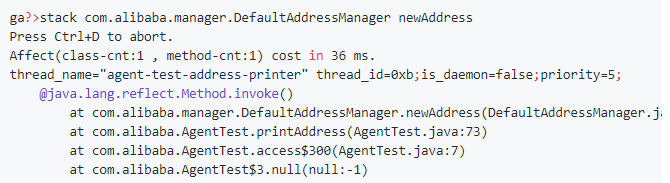
## 2.10 stack

获取某个方法的执行路径

1）参数说明

|  |  |
| --- | --- |
| 参数名称 | 参数说明 |
| class-pattern | 类名表达式匹配 |
| method-pattern | 方法名表达式匹配 |
| condition-express | 条件表达式 |
| [n:] | 命令执行次数 |
| [E] | 支持正则表达式匹配 |

2）使用例子



## 2.11 version

输出当前Greys的版本号

## 2.12 quit

将客户端关闭，不会将目标Java进程中的Greys的Server关闭

## 2.13 shutdown

命令执行完成两个事情：

* 关闭Greys在目标Java进程所加载的Socket服务，所占用的端口被释放
* 重置所有被Greys增强的类

## 2.14 reset

重置指定被Greys所增强的类

## 2.15 jvm

查看当前JVM信息，无参数

其中命令，如js命令，不再介绍

# 参考文献：

https://github.com/oldmanpushcart/greys-anatomy

https://github.com/oldmanpushcart/greys-anatomy/wiki/greys-pdf

https://www.cnblogs.com/zengkefu/p/5720854.html