调试排错 - Java应用在线调试Arthas

Arthas简介

*Arthas*是什么

Arthas 是Alibaba开源的Java诊断工具，深受开发者喜爱。

*Arthas*能解决什么问题

当你遇到以下类似问题而束手无策时，Arthas可以帮助你解决：

- 这个类从哪个 jar 包加载的? 为什么会报各种类相关的 Exception?
- 我改的代码为什么没有执行到? 难道是我没 commit? 分支搞错了?
- 遇到问题无法在线上 debug，难道只能通过加日志再重新发布吗?
- 线上遇到某个用户的数据处理有问题，但线上同样无法 debug，线下无法重现!
- 是否有一个全局视角来查看系统的运行状况?
- 有什么办法可以监控到JVM的实时运行状态?

Arthas支持JDK 6+，支持Linux/Mac/Windows，采用命令行交互模式，同时提供丰富的 Tab 自动补全功能，进一步方便进行问题的定位和诊断。

*Arthas*资源推荐

- 用户文档 (opens new window)
- 官方在线教程(推荐) (opens new window)
- 快速入门 (opens new window)
- 进阶使用 (opens new window)
- 命令列表 (opens new window)
- WebConsole (opens new window)
- Docker (opens new window)
- 用户案例 (opens new window)
- 常见问题 (opens new window)

*Arthas*基于了哪些工具上发展而来

- greys-anatomy (opens new window): Arthas代码基于Greys二次开发而来，非常感谢Greys之前所有的工作，以及Greys原作者对Arthas提出的意见和建议！
- termd (opens new window): Arthas的命令行实现基于termd开发，是一款优秀的命令行程序开发框架，感谢termd提供了优秀的框架。
- crash (opens new window): Arthas的文本渲染功能基于crash中的文本渲染功能开发，可以从这里 (opens new window)看到源码，感谢crash在这方面所做的优秀工作。
- cli (opens new window): Arthas的命令行界面基于vert.x提供的cli库进行开发，感谢vert.x在这方面做的优秀工作。
- compiler (opens new window) Arthas里的内存编绎器代码来源
- Apache Commons Net (opens new window) Arthas里的Telnet Client代码来源
- `JavaAgent`：运行在 main方法之前的拦截器，它内定的方法名叫 premain ，也就是说先执行 premain 方法然后再执行 main 方法
- `ASM`：一个通用的Java字节码操作和分析框架。它可以用于修改现有的类或直接以二进制形式动态生成类。ASM提供了一些常见的字节码转换和分析算法，可以从它们构建定制的复杂转换和代码分析工具。ASM提供了与其他Java字节码框架类似的功能，但是主要关注性能。因为它被设计和实现得尽可能小和快，所以非常适合在动态系统中使用(当然也可以以静态方式使用，例如在编译器中)

同类工具有哪些

- BTrace
- 美团 Java-debug-tool
- 去哪儿Bistoury: 一个集成了Arthas的项目 (opens new window)
- 一个使用MVEL脚本的fork (opens new window)

Arthas入门

Arthas 上手前

推荐先在线使用下arthas：官方在线教程(推荐) (opens new window)

Arthas 安装

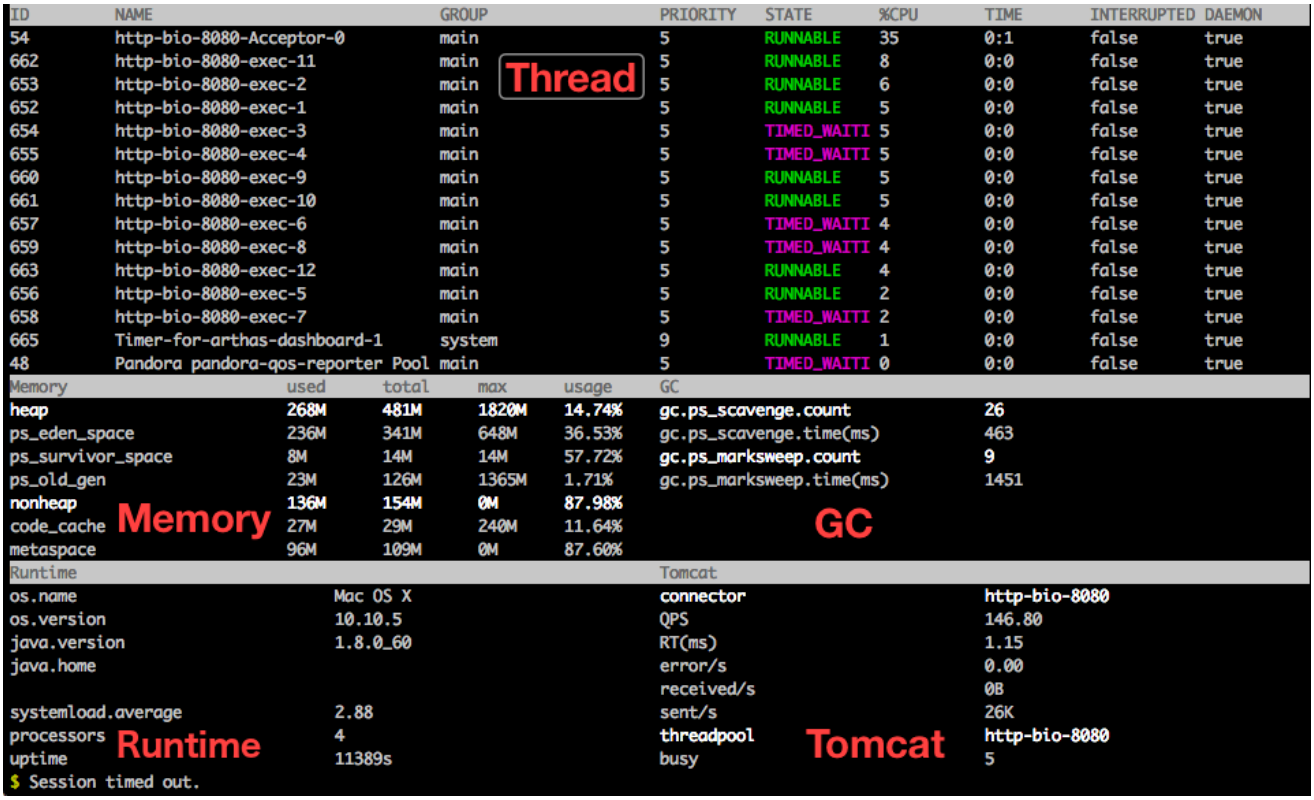
下载`arthas-boot.jar`，然后用`java -jar`的方式启动：

```
curl -O https://alibaba.github.io/arthas/arthas-boot.jar
java -jar arthas-boot.jar
```

Arthas 官方案例展示

Dashboard

- https://alibaba.github.io/arthas/dashboard



Thread

一目了然的了解系统的状态，哪些线程比较占cpu? 他们到底在做什么?

```
$ thread -n 3
"as-command-execute-daemon" Id=29 cpuUsage=75% RUNNABLE
    at sun.management.ThreadImpl.dumpThreads0(Native Method)
    at sun.management.ThreadImpl.getThreadInfo(ThreadImpl.java:440)
    at com.taobao.arthas.core.command.monitor200.ThreadCommand$1.action(ThreadCommand.java:58)
    at
com.taobao.arthas.core.command.handler.AbstractCommandHandler.execute(AbstractCommandHandler.jav
a:238)
    at
com.taobao.arthas.core.command.handler.DefaultCommandHandler.handleCommand(DefaultCommandHandler
.java:67)
    at com.taobao.arthas.core.server.ArthasServer$4.run(ArthasServer.java:276)
    at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1145)
    at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:615)
    at java.lang.Thread.run(Thread.java:745)

    Number of locked synchronizers = 1
    - java.util.concurrent.ThreadPoolExecutor$Worker@6cd0b6f8

"as-session-expire-daemon" Id=25 cpuUsage=24% TIMED_WAITING
    at java.lang.Thread.sleep(Native Method)
    at com.taobao.arthas.core.server.DefaultSessionManager$2.run(DefaultSessionManager.java:85)

"Reference Handler" Id=2 cpuUsage=0% WAITING on java.lang.ref.Reference$Lock@69ba0f27
    at java.lang.Object.wait(Native Method)
    -  waiting on java.lang.ref.Reference$Lock@69ba0f27
```

```
    at java.lang.Object.wait(Object.java:503)
    at java.lang.ref.Reference$ReferenceHandler.run(Reference.java:133)
```

jad

对类进行反编译:

```
$ jad javax.servlet.Servlet

ClassLoader:
+-java.net.URLClassLoader@6108b2d7
  +-sun.misc.Launcher$AppClassLoader@18b4aac2
    +-sun.misc.Launcher$ExtClassLoader@1ddf84b8

Location:
/Users/xxx/work/test/lib/servlet-api.jar

/*
 * Decompiled with CFR 0_122.
 */
package javax.servlet;

import java.io.IOException;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

public interface Servlet {
    public void init(ServletConfig var1) throws ServletException;

    public ServletConfig getServletConfig();

    public void service(ServletRequest var1, ServletResponse var2) throws ServletException,
IOException;

    public String getServletInfo();

    public void destroy();
}
```

mc

Memory Compiler/内存编译器，编译.java文件生成.class。

```
mc /tmp/Test.java
```

redefine

加载外部的.class文件，redefine jvm已加载的类。

```
redefine /tmp/Test.class
redefine -c 327a647b /tmp/Test.class /tmp/Test\$Inner.class
```

sc

查找JVM中已经加载的类

```
$ sc -d org.springframework.web.context.support.XmlWebApplicationContext
 class-info        org.springframework.web.context.support.XmlWebApplicationContext
 code-source       /Users/xxx/work/test/WEB-INF/lib/spring-web-3.2.11.RELEASE.jar
 name              org.springframework.web.context.support.XmlWebApplicationContext
 isInterface       false
 isAnnotation      false
 isEnum            false
 isAnonymousClass  false
 isArray           false
 isLocalClass      false
 isMemberClass     false
 isPrimitive       false
 isSynthetic       false
 simple-name       XmlWebApplicationContext
 modifier          public
 annotation
 interfaces
 super-class       +-
org.springframework.web.context.support.AbstractRefreshableWebApplicationContext
                     +-
org.springframework.context.support.AbstractRefreshableConfigApplicationContext
                       +-
org.springframework.context.support.AbstractRefreshableApplicationContext
                         +-org.springframework.context.support.AbstractApplicationContext
                           +-org.springframework.core.io.DefaultResourceLoader
                             +-java.lang.Object
 class-loader      +-org.apache.catalina.loader.ParallelWebappClassLoader
                     +-java.net.URLClassLoader@6108b2d7
                       +-sun.misc.Launcher$AppClassLoader@18b4aac2
                         +-sun.misc.Launcher$ExtClassLoader@1ddf84b8
 classLoaderHash   25131501
```

stack

查看方法 `test.arthas.TestStack#doGet` 的调用堆栈：

```
$ stack test.arthas.TestStack doGet
Press Ctrl+C to abort.
Affect(class-cnt:1 , method-cnt:1) cost in 286 ms.
ts=2018-09-18 10:11:45;thread_name=http-bio-8080-exec-
10;id=d9;is_daemon=true;priority=5;TCCL=org.apache.catalina.loader.ParallelWebappClassLoader@251
31501
    @test.arthas.TestStack.doGet()
        at javax.servlet.http.HttpServlet.service(HttpServlet.java:624)
        at javax.servlet.http.HttpServlet.service(HttpServlet.java:731)
        at
org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:303
)
        at
org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:208)
        at org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:52)
        at
org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:241
)
```

```
        at
org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:208)
        at
org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:241
)
        at
org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:208)
        at org.apache.catalina.core.StandardWrapperValve.invoke(StandardWrapperValve.java:220)
        at org.apache.catalina.core.StandardContextValve.invoke(StandardContextValve.java:110)
        ...
        at org.apache.catalina.core.StandardHostValve.invoke(StandardHostValve.java:169)
        at org.apache.catalina.valves.ErrorReportValve.invoke(ErrorReportValve.java:103)
        at org.apache.catalina.core.StandardEngineValve.invoke(StandardEngineValve.java:116)
        at org.apache.catalina.connector.CoyoteAdapter.service(CoyoteAdapter.java:451)
        at
org.apache.coyote.http11.AbstractHttp11Processor.process(AbstractHttp11Processor.java:1121)
        at
org.apache.coyote.AbstractProtocol$AbstractConnectionHandler.process(AbstractProtocol.java:637)
        at org.apache.tomcat.util.net.JIoEndpoint$SocketProcessor.run(JIoEndpoint.java:316)
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)
        at org.apache.tomcat.util.threads.TaskThread$WrappingRunnable.run(TaskThread.java:61)
        at java.lang.Thread.run(Thread.java:745)
```

Trace

观察方法执行的时候哪个子调用比较慢:

```
$ trace test.arthas.TestTraceServlet doGet
Press Ctrl+C to abort.
Affect(class-cnt:1 , method-cnt:1) cost in 164 ms.
`---ts=2018-09-18 10:21:24;thread_name=http-bio-8080-exec-4;id=cf;is_daemon=true;priority=5;TCCL=org.apache.catalina.load
er.ParallelWebappClassLoader@5903d6b6
    `---[811.251691ms] test.arthas.TestTraceServlet:doGet()
        +---[min=0.030753ms,max=0.354118ms,total=0.384871ms,count=2] java.lang.System:currentTimeMillis()
        +---[1.366375ms] test.arthas.TestTraceServlet:call1()
        +---[383.238375ms] test.arthas.TestTraceServlet:callhsfL1()
        +---[0.014185ms] test.arthas.TestPathTrace:<init>()
        +---[381.203972ms] test.arthas.TestPathTrace:callPathTrace()
        +---[10.408095ms] test.arthas.TestTraceServlet:call3()
        +---[2.720931ms] test.arthas.TestTraceServlet:call4()
        +---[1.150837ms] test.arthas.TestTraceServlet:call5()
        +---[10.156395ms] test.arthas.TestTraceServlet:call6()
        +---[8.333288ms] test.arthas.TestTraceServlet:call7()
        +---[3.304381ms] test.arthas.TestTraceServlet:call8()
        +---[6.456575ms] test.arthas.TestTraceServlet:call9()
        +---[0.172433ms] javax.servlet.http.HttpServletResponse:getWriter()
        +---[0.015543ms] java.lang.StringBuilder:<init>()
        +---[min=0.002805ms,max=0.04123ms,total=0.083268ms,count=3] java.lang.StringBuilder:append()
        +---[0.009705ms] java.lang.StringBuilder:toString()
        `---[0.018935ms] java.io.PrintWriter:write()
```

Watch

观察方法 `test.arthas.TestWatch#doGet` 执行的入参，仅当方法抛出异常时才输出。

```
$ watch test.arthas.TestWatch doGet {params[0], throwExp} -e
Press Ctrl+C to abort.
Affect(class-cnt:1 , method-cnt:1) cost in 65 ms.
ts=2018-09-18 10:26:28;result=@ArrayList[
    @RequestFacade[org.apache.catalina.connector.RequestFacade@79f922b2],
    @NullPointerException[java.lang.NullPointerException],
]
```

Monitor

监控某个特殊方法的调用统计数据，包括总调用次数，平均rt，成功率等信息，每隔5秒输出一次。

```
$ monitor -c 5 org.apache.dubbo.demo.provider.DemoServiceImpl sayHello
Press Ctrl+C to abort.
Affect(class-cnt:1 , method-cnt:1) cost in 109 ms.
 timestamp            class                                          method    total  success
fail  avg-rt(ms)  fail-rate
-----------------------------------------------------------------------------------------------
---------------------------
 2018-09-20 09:45:32  org.apache.dubbo.demo.provider.DemoServiceImpl  sayHello  5      5
 0     0.67        0.00%

 timestamp            class                                          method    total  success
fail  avg-rt(ms)  fail-rate
-----------------------------------------------------------------------------------------------
---------------------------
 2018-09-20 09:45:37  org.apache.dubbo.demo.provider.DemoServiceImpl  sayHello  5      5
 0     1.00        0.00%

 timestamp            class                                          method    total  success
fail  avg-rt(ms)  fail-rate
-----------------------------------------------------------------------------------------------
---------------------------
 2018-09-20 09:45:42  org.apache.dubbo.demo.provider.DemoServiceImpl  sayHello  5      5
 0     0.43        0.00%
```

Time Tunnel(tt)

记录方法调用信息，支持事后查看方法调用的参数，返回值，抛出的异常等信息，仿佛穿越时空隧道回到调用现场一般。

```
$ tt -t org.apache.dubbo.demo.provider.DemoServiceImpl sayHello
Press Ctrl+C to abort.
Affect(class-cnt:1 , method-cnt:1) cost in 75 ms.
 INDEX   TIMESTAMP            COST(ms)  IS-RET  IS-EXP   OBJECT         CLASS
       METHOD
-----------------------------------------------------------------------------------------------
-------------------------------------
 1000    2018-09-20 09:54:10  1.971195  true    false    0x55965cca     DemoServiceImpl
       sayHello
 1001    2018-09-20 09:54:11  0.215685  true    false    0x55965cca     DemoServiceImpl
       sayHello
 1002    2018-09-20 09:54:12  0.236303  true    false    0x55965cca     DemoServiceImpl
       sayHello
 1003    2018-09-20 09:54:13  0.159598  true    false    0x55965cca     DemoServiceImpl
       sayHello
 1004    2018-09-20 09:54:14  0.201982  true    false    0x55965cca     DemoServiceImpl
       sayHello
 1005    2018-09-20 09:54:15  0.214205  true    false    0x55965cca     DemoServiceImpl
       sayHello
 1006    2018-09-20 09:54:16  0.241863  true    false    0x55965cca     DemoServiceImpl
       sayHello
 1007    2018-09-20 09:54:17  0.305747  true    false    0x55965cca     DemoServiceImpl
       sayHello
 1008    2018-09-20 09:54:18  0.18468   true    false    0x55965cca     DemoServiceImpl
       sayHello
```

Classloader

了解当前系统中有多少类加载器，以及每个加载器加载的类数量，帮助您判断是否有类加载器泄露。

```
$ classloader
 name                                                numberOfInstances  loadedCountTotal
 BootstrapClassLoader                                1                  3346
 com.taobao.arthas.agent.ArthasClassloader           1                  1262
 java.net.URLClassLoader                             2                  1033
 org.apache.catalina.loader.ParallelWebappClassLoader 1                 628
 sun.reflect.DelegatingClassLoader                   166                166
 sun.misc.Launcher$AppClassLoader                    1                  31
 com.alibaba.fastjson.util.ASMClassLoader            6                  15
 sun.misc.Launcher$ExtClassLoader                    1                  7
 org.jvnet.hk2.internal.DelegatingClassLoader        2                  2
 sun.reflect.misc.MethodUtil                         1                  1
```

Web Console

- https://alibaba.github.io/arthas/web-console

Arthas 命令集

基础命令

- help——查看命令帮助信息
- cat (opens new window)——打印文件内容，和linux里的cat命令类似
- [grep]](https://github.com/alibaba/arthas/blob/master/site/src/site/sphinx/grep.md)——匹配查找，和linux里的grep命令类似
- pwd (opens new window)——返回当前的工作目录，和linux命令类似
- cls——清空当前屏幕区域
- session——查看当前会话的信息
- reset (opens new window)——重置增强类，将被 Arthas 增强过的类全部还原，Arthas 服务端关闭时会重置所有增强过的类
- version——输出当前目标 Java 进程所加载的 Arthas 版本号
- history——打印命令历史
- quit——退出当前 Arthas 客户端，其他 Arthas 客户端不受影响
- stop/shutdown——关闭 Arthas 服务端，所有 Arthas 客户端全部退出
- keymap (opens new window)——Arthas快捷键列表及自定义快捷键

jvm相关

- dashboard（opens new window)——当前系统的实时数据面板
- thread（opens new window)——查看当前 JVM 的线程堆栈信息
- jvm（opens new window)——查看当前 JVM 的信息
- sysprop（opens new window)——查看和修改JVM的系统属性
- sysenv（opens new window)——查看JVM的环境变量
- vmoption（opens new window)——查看和修改JVM里诊断相关的option
- logger（opens new window)——查看和修改logger
- getstatic（opens new window)——查看类的静态属性
- ognl（opens new window)——执行ognl表达式
- mbean（opens new window)——查看 Mbean 的信息
- heapdump（opens new window)——dump java heap, 类似jmap命令的heap dump功能

class/classloader相关

- sc（opens new window)——查看JVM已加载的类信息
- sm（opens new window)——查看已加载类的方法信息
- jad（opens new window)——反编译指定已加载类的源码
- mc（opens new window)——内存编绎器，内存编绎`.java`文件为`.class`文件
- redefine（opens new window)——加载外部的`.class`文件，redefine到JVM里
- dump（opens new window)——dump 已加载类的 byte code 到特定目录
- classloader（opens new window)——查看classloader的继承树，urls，类加载信息，使用classloader去getResource

monitor/watch/trace相关

> 请注意，这些命令，都通过字节码增强技术来实现的，会在指定类的方法中插入一些切面来实现数据统计和观测，因此在线上、预发使用时，请尽量明确需要观测的类、方法以及条件，诊断结束要执行 `shutdown` 或将增强过的类执行 `reset` 命令。

- monitor（opens new window)——方法执行监控
- watch（opens new window)——方法执行数据观测
- trace（opens new window)——方法内部调用路径，并输出方法路径上的每个节点上耗时
- stack（opens new window)——输出当前方法被调用的调用路径
- tt（opens new window)——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

options

- options（opens new window)——查看或设置Arthas全局开关

管道

Arthas支持使用管道对上述命令的结果进行进一步的处理，如`sm java.lang.String * | grep 'index'`

- grep——搜索满足条件的□结果
- plaintext——将□命令的结果去除ANSI颜色
- wc——按行统计输出结果

后台异步任务

当线上出现偶发的问题，比如需要watch某个条件，而这个条件一天可能才会出现一次时，异步后台任务就派上用场了，详情请参考这里（opens new window)

- 使用 > 将结果重写向到日志文件，使用 & 指定命令是后台运行，session断开不影响任务执行(生命周期默认为1天)

- jobs——列出所有job
- kill——强制终止任务
- fg——将暂停的任务拉到前台执行
- bg——将暂停的任务放到后台执行

Web Console

通过websocket连接Arthas。

- Web Console (opens new window)

用户数据回报

在3.1.4版本后，增加了用户数据回报功能，方便统一做安全或者历史数据统计。

在启动时，指定stat-url，就会回报执行的每一行命令，比如：`./as.sh --stat-url 'http://192.168.10.11:8080/api/stat'`

在tunnel server里有一个示例的回报代码，用户可以自己在服务器上实现。

StatController.java (opens new window)

其他特性

- 异步命令支持 (opens new window)
- 执行结果存日志 (opens new window)
- 批处理的支持 (opens new window)
- ognl表达式的用法说明 (opens new window)

Arthas场景实战

查看最繁忙的线程，以及是否有阻塞情况发生?

> 场景：我想看下查看最繁忙的线程，以及是否有阻塞情况发生? 常规查看线程，一般我们可以通过 top 等系统命令进行查看，但是那毕竟要很多个步骤，很麻烦。

```
thread -n 3 # 查看最繁忙的三个线程栈信息
thread  # 以直观的方式展现所有的线程情况
thread -b #找出当前阻塞其他线程的线程
```

确认某个类是否已被系统加载?

> 场景：我新写了一个类或者一个方法，我想知道新写的代码是否被部署了?

```
# 即可以找到需要的类全路径，如果存在的话
sc *MyServlet

# 查看这个某个类所有的方法
sm pdai.tech.servlet.TestMyServlet *

# 查看某个方法的信息，如果存在的话
sm pdai.tech.servlet.TestMyServlet testMethod
```

如何查看一个*class*类的源码信息?

> 场景：我新修改的内容在方法内部，而上一个步骤只能看到方法，这时候可以反编译看下源码

```
# 直接反编译出java 源代码，包含一此额外信息的
jad pdai.tech.servlet.TestMyServlet
```

重要：如何跟踪某个方法的返回值、入参.... ?

> 场景：我想看下我新加的方法在线运行的参数和返回值?

```
# 同时监控入参，返回值，及异常
watch pdai.tech.servlet.TestMyServlet testMethod "{params, returnObj, throwExp}" -e -x 2
```

具体看watch命令。

如何看方法调用栈的信息?

> 场景：我想看下某个方法的调用栈的信息?

```
stack pdai.tech.servlet.TestMyServlet testMethod
```

运行此命令之后需要即时触发方法才会有响应的信息打印在控制台上

重要：找到最耗时的方法调用?

> 场景：testMethod这个方法入口响应很慢，如何找到最耗时的子调用?

```
# 执行的时候每个子调用的运行时长，可以找到最耗时的子调用。
stack pdai.tech.servlet.TestMyServlet testMethod
```

运行此命令之后需要即时触发方法才会有响应的信息打印在控制台上，然后一层一层看子调用。

重要：如何临时更改代码运行?

> 场景：我找到了问题所在，能否线上直接修改测试，而不需要在本地改了代码后，重新打包部署，然后重启观察效果?

```
# 先反编译出class源码
jad --source-only com.example.demo.arthas.user.UserController > /tmp/UserController.java

# 然后使用外部工具编辑内容
mc /tmp/UserController.java -d /tmp  # 再编译成class

# 最后，重新载入定义的类，就可以实时验证你的猜测了
redefine /tmp/com/example/demo/arthas/user/UserController.class
```

如上，是直接更改线上代码的方式，但是一般好像是编译不成功的。所以，最好是本地ide编译成 class文件后，再上传替换为好!

总之，已经完全不用重启和发布了！这个功能真的很方便，比起重启带来的代价，真的是不可比的。比如，重启时可能导致负载重分配，选主等等问题，就不是你能控制的了。

我如何测试某个方法的性能问题?

> 场景：我想看下某个方法的性能

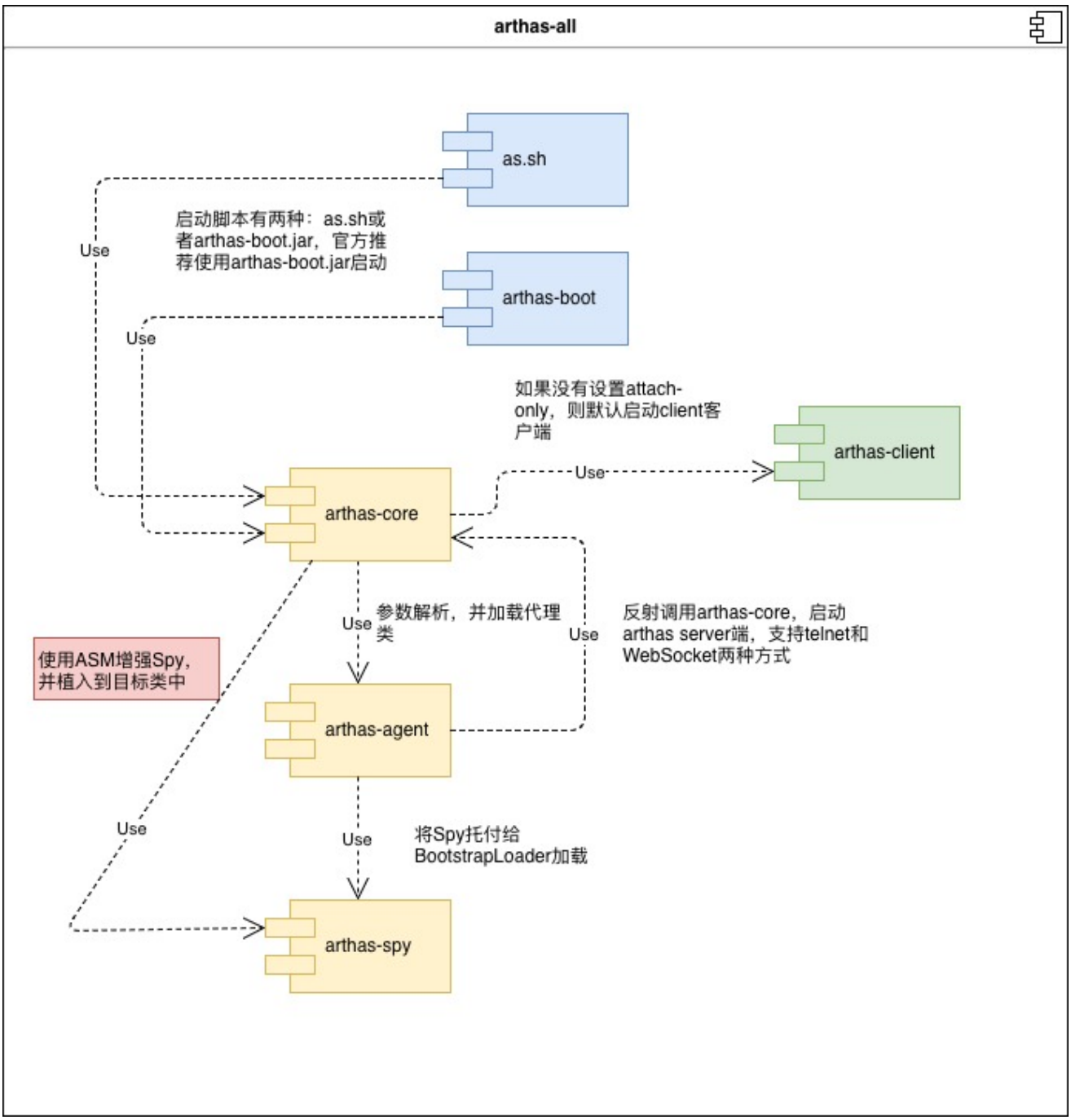
```
monitor -c 5 demo.MathGame primeFactors
```

更多

请参考: 官方Issue墙 (opens new window)

Arthas源码

整体宏观的模块调用图：



源码理解可以看这两篇文章:

- 什么是 Arthas(opens new window)
- Arthas阅读(opens new window)