监控 (http://39.106.3.150/tags/#1561876610222) 全链路 (http://39.106.3.150/tags/#1561876610220)

Java全链路监控技术

Posted by Palmer on 06-30，2019

BTrace

BTrace是基于Java语言的一个安全的、可提供动态追踪服务的工具。BTrace基于ASM、Java Attach Api、Instruments开发，为用户提供了很多注解。依靠这些注解，我们可以编写BTrace脚本（简单的Java代码）达到我们想要的效果，而不必深陷于ASM对字节码的操作中不可自拔。

看BTrace官方提供的一个简单例子：拦截所有java.io包中所有类中以read开头的方法，打印类名、方法名和参数名。当程序IO负载比较高的时候，就可以从输出的信息中看到是哪些类所引起，是不是很方便?

```
package com.sun.btrace.samples;

import com.sun.btrace.annotations.*;
import com.sun.btrace.AnyType;
import static com.sun.btrace.BTraceUtils.*;

/**
 * This sample demonstrates regular expression
 * probe matching and getting input arguments
 * as an array - so that any overload variant
 * can be traced in "one place". This example
 * traces any "readXX" method on any class in
 * java.io package. Probed class, method and arg
 * array is printed in the action.
 */
@BTrace public class ArgArray {
    @OnMethod(
        clazz="/java\\.io\\..*/",
        method="/read.*/"
    )
    public static void anyRead(@ProbeClassName String pcn, @ProbeMethodName String pm
        println(pcn);
        println(pmn);
        printArray(args);
    }
}
```

再来看另一个例子：每隔2秒打印截止到当前创建过的线程数。

```
package com.sun.btrace.samples;

import com.sun.btrace.annotations.*;
import static com.sun.btrace.BTraceUtils.*;
import com.sun.btrace.annotations.Export;

/**
 * This sample creates a jvmstat counter and
 * increments it everytime Thread.start() is
 * called. This thread count may be accessed
 * from outside the process. The @Export annotated
 * fields are mapped to jvmstat counters. The counter
 * name is "btrace." + <className> + "." + <fieldName>
 */
@BTrace public class ThreadCounter {

    // create a jvmstat counter using @Export
    @Export private static long count;

    @OnMethod(
        clazz="java.lang.Thread",
        method="start"
    )
    public static void onnewThread(@Self Thread t) {
        // updating counter is easy. Just assign to
        // the static field!
        count++;
    }

    @OnTimer(2000)
    public static void ontimer() {
        // we can access counter as "count" as well
        // as from jvmstat counter directly.
        println(count);
        // or equivalently ...
        println(Counters.perfLong("btrace.com.sun.btrace.samples.ThreadCounter.count"
    }
}
```

有了BTrace，文章开头的问题可以得到完美的解决。至于BTrace具体有哪些功能，脚本怎么写，这些Git上BTrace工程中有大量的说明和举例。

主要有下面几个模块：

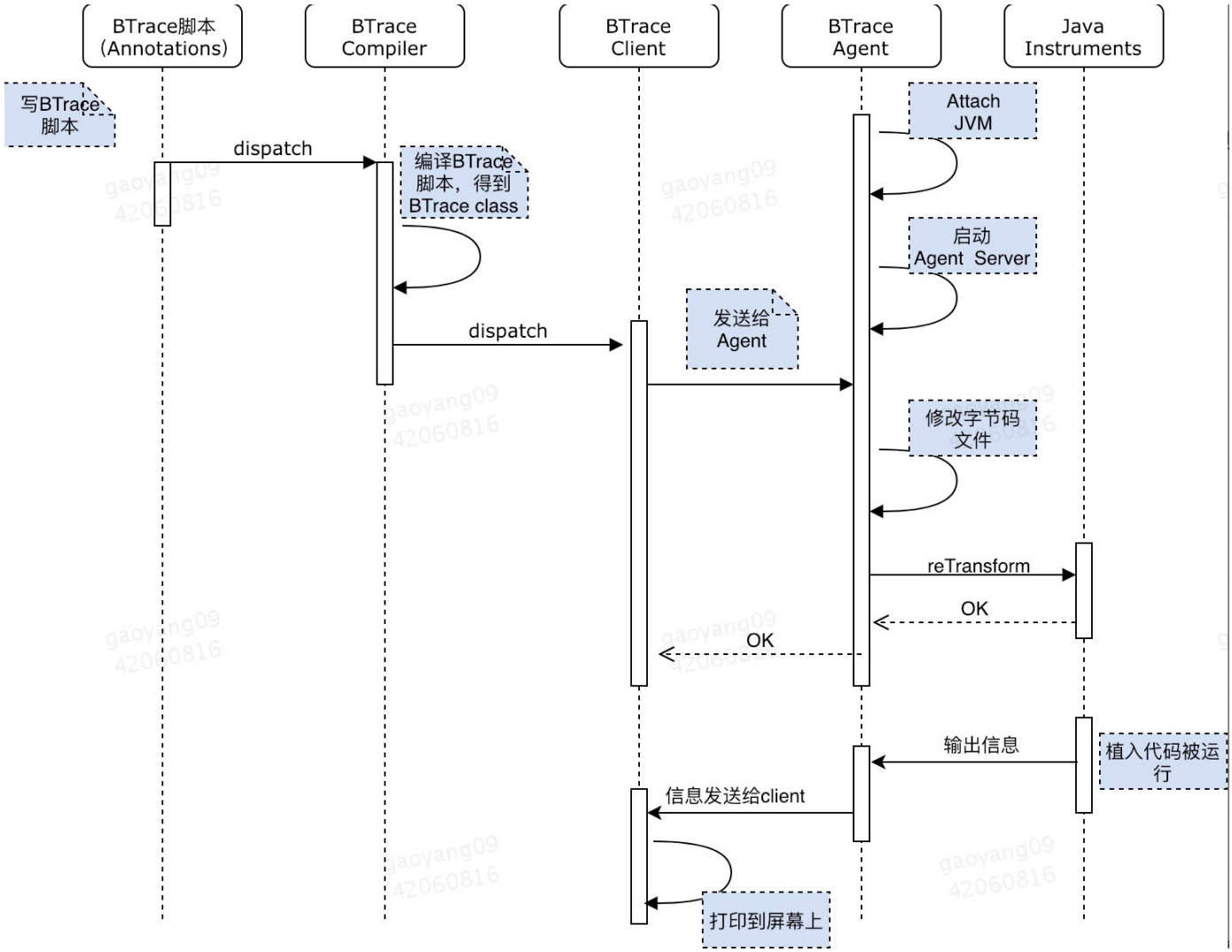
BTrace脚本：利用BTrace定义的注解，我们可以很方便地根据需要进行脚本的开发。

Compiler：将BTrace脚本编译成BTrace class文件。

Client：将class文件发送到Agent。

Agent：基于Java的Attach Api，Agent可以动态附着到一个运行的JVM上，然后开启一个BTrace Server，接收client发过来的BTrace脚本；解析脚本，然后根据脚本中的规则找到要修改的类；修改字节码后，调用Java Instrument的reTransform接口，完成对对象行为的修改并使之生效。

整个BTrace的架构大致如下：



Arthas

Arthas是Alibaba开源的Java诊断工具，深受开发者喜爱。

1、当你遇到以下类似问题而束手无策时，Arthas可以帮助你解决：

2、这个类从哪个 jar 包加载的？为什么会报各种类相关的 Exception？

3、我改的代码为什么没有执行到？难道是我没 commit？分支搞错了？

4、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？

5、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！

6、是否有一个全局视角来查看系统的运行状况？

7、有什么办法可以监控到JVM的实时运行状态？

Arthas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。

详细文档说明：arthas快速指南 (https://alibaba.github.io/arthas/quick-start.html)

jvm-sandbox

BTRACE好强大，也曾技痒想做一个更便捷、更适合自己的问题定位工具，既可支持线上链路监控排查，也可支持单机版问题定位。

有时候突然一个问题反馈上来，需要入参才能完成定位，但恰恰没有任何日志，甚至出现在别人的代码里，好想开发一个工具可以根据需要动态添加日志，最好还能按照业务ID进行过滤。

系统间的异常模拟可以使用的工具很多，可是系统内的异常模拟怎么办，加开关或是用AOP在开发系统中实现，好想开发一个更优雅的异常模拟工具，既能模拟系统间的异常，又能模拟系统内的异常。

好想获取行调用链路数据，可以用它识别场景、覆盖率统计等等，覆盖率统计工具不能原生支持，统计链路数据不准确。想自己开发一个工具获取行链路数据。

我想开发录制回放、故障模拟、动态日志、行链路获取等等工具，就算我开发完成了，这些工具底层实现原理相同，同时使用，要怎么消除这些工具之间的影响，怎么保证这些工具动态加载，怎么保证动态加载/卸载之后不会影响其他工具，怎么保证在工具有问题的时候，快速消除影响，代码还原

如果你有以上研发诉求，那么你就是JVM-SANDBOX(以下简称沙箱容器)的潜在客户。沙箱容器提供

动态增强类你所指定的类,获取你想要的参数和行信息甚至改变方法执行,动态可插拔容器框架

JVM-SANDBOX（沙箱）实现了一种在不重启、不侵入目标JVM应用的AOP解决方案。

箱的特性

无侵入：目标应用无需重启也无需感知沙箱的存在

类隔离：沙箱以及沙箱的模块不会和目标应用的类相互干扰

可插拔：沙箱以及沙箱的模块可以随时加载和卸载，不会在目标应用留下痕迹

多租户：目标应用可以同时挂载不同租户下的沙箱并独立控制

高兼容：支持JDK[6,11]

沙箱常见应用场景

线上故障定位

线上系统流控

线上故障模拟

方法请求录制和结果回放

动态日志打印

安全信息监测和脱敏

JVM-SANDBOX还能帮助你做很多很多，取决于你的脑洞有多大了。

实时无侵入AOP框架

在常见的AOP框架实现方案中，有静态编织和动态编织两种。

静态编织：静态编织发生在字节码生成时根据一定框架的规则提前将AOP字节码插入到目标类和方法中，实现AOP；

动态编织：动态编织则允许在JVM运行过程中完成指定方法的AOP字节码增强.常见的动态编织方案大多采用重命名原有方法，再新建一个同签名的方法来做代理的工作模式来完成AOP的功能(常见的实现方案如CgLib)，但这种方式存在一些应用边界：

侵入性：对被代理的目标类需要进行侵入式改造。比如：在Spring中必须是托管于Spring容器中的Bean

固化性：目标代理方法在启动之后即固化，无法重新对一个已有方法进行AOP增强

要解决无侵入的特性需要AOP框架具备 在运行时完成目标方法的增强和替换。在JDK的规范中运行期重定义一个类必须准循以下原则

不允许新增、修改和删除成员变量

不允许新增和删除方法

不允许修改方法签名

JVM-SANDBOX属于基于Instrumentation的动态编织类的AOP框架，**通过精心构造了字节码增强逻辑，使得沙箱的模块能在不违反JDK约束情况下实现对目标应用方法的无侵入运行时AOP拦截。**

核心原理

事件驱动

在沙箱的世界观中，任何一个Java方法的调用都可以分解为BEFORE、RETURN和THROWS三个环节，由此在三个环节上引申出对应环节的事件探测和流程控制机制。

```
// BEFORE
try {

   /*
    * do something...
    */

    // RETURN
    return;

} catch (Throwable cause) {
    // THROWS
}
```

基于BEFORE、RETURN和THROWS三个环节事件分离，沙箱的模块可以完成很多类AOP的操作。

1、可以感知和改变方法调用的入参

2、可以感知和改变方法调用返回值和抛出的异常

3、可以改变方法执行的流程

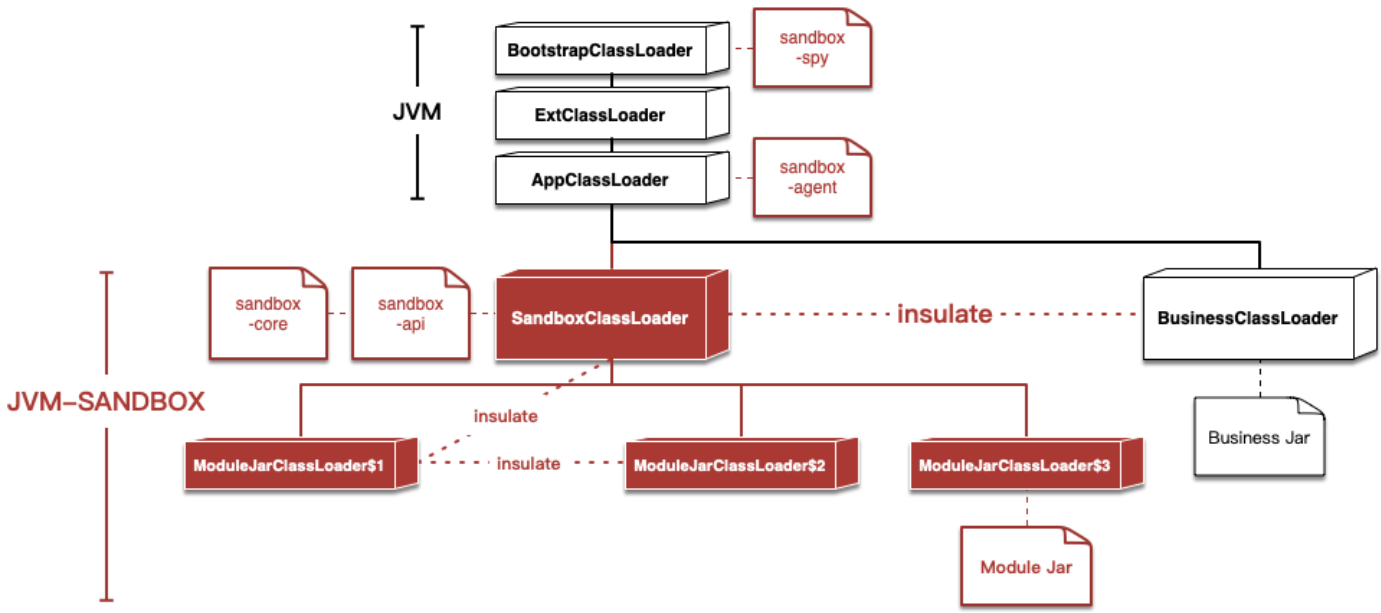
在方法体执行之前直接返回自定义结果对象，原有方法代码将不会被执行

在方法体返回之前重新构造新的结果对象，甚至可以改变为抛出异常

在方法体抛出异常之后重新抛出新的异常，甚至可以改变为正常返回

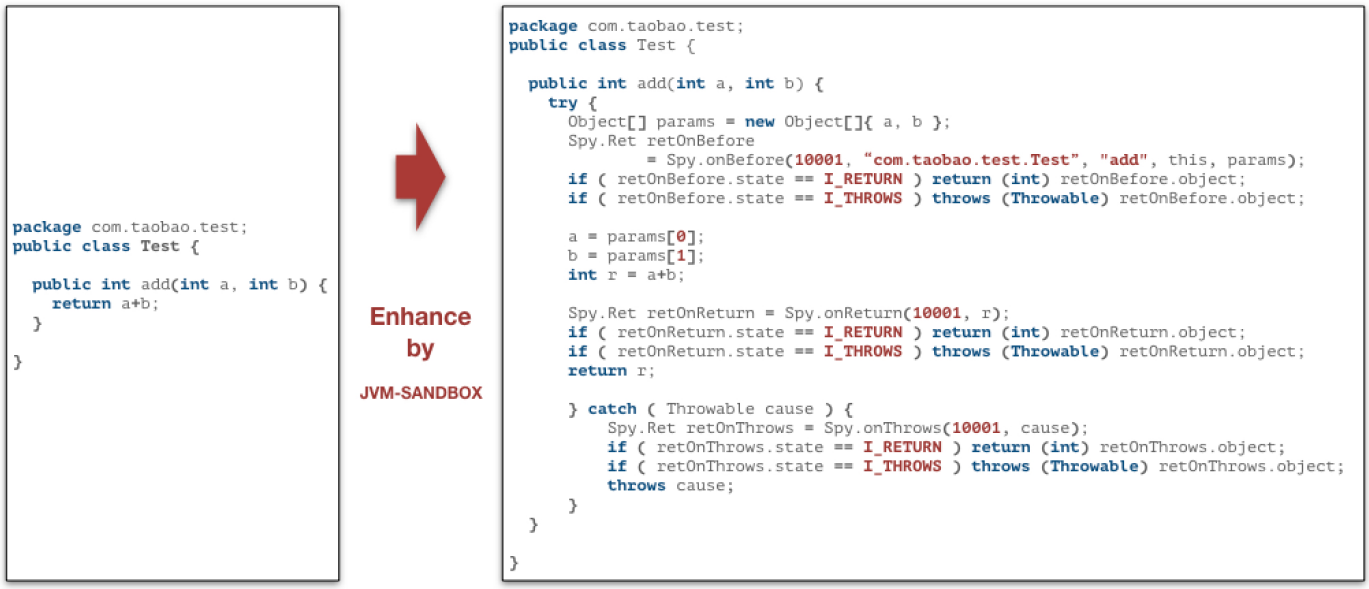
类隔离策略

沙箱通过自定义的SandboxClassLoader破坏了双亲委派的约定，实现了和目标应用的类隔离。所以不用担心加载沙箱会引起应用的类污染、冲突。各模块之间类通过ModuleJarClassLoader实现了各自的独立，达到模块之间、模块和沙箱之间、模块和应用之间互不干扰。

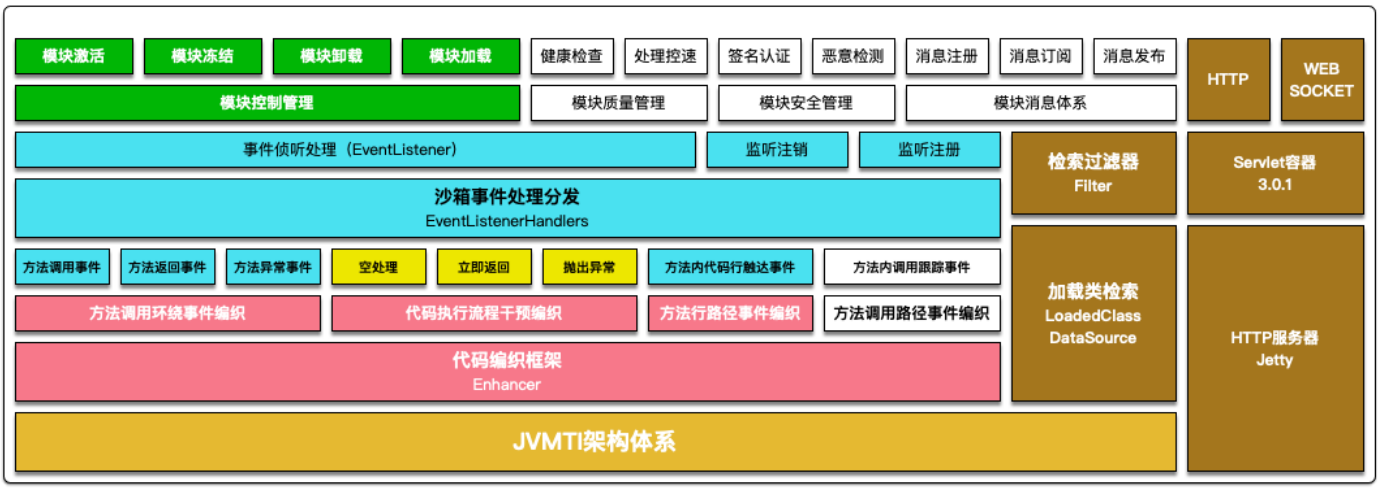


类增强策略

沙箱通过在BootstrapClassLoader中埋藏的Spy类完成目标类和沙箱内核的通讯


package com.taobao.test;
public class Test {
public int add(int a, int b) {
return a+b;
}
}
Enhance
by
JVM-SANDBOX
package com.taobao.test;
public class Test {
public int add(int a, int b) {
try {
Object[] params = new Object[]{ a, b };
Spy.Ret retOnBefore
= Spy.onBefore(10001, "com.taobao.test.Test", "add", this, params);
if (retOnBefore.state == I_RETURN) return (int) retOnBefore.object;
if (retOnBefore.state == I_THROWS) throws (Throwable) retOnBefore.object;
a = params[0];
b = params[1];
int r = a+b;
Spy.Ret retOnReturn = Spy.onReturn(10001, r);
if (retOnReturn.state == I_RETURN) return (int) retOnReturn.object;
if (retOnReturn.state == I_THROWS) throws (Throwable) retOnReturn.object;
return r;
} catch (Throwable cause) {
Spy.Ret retOnThrows = Spy.onThrows(10001, cause);
if (retOnThrows.state == I_RETURN) return (int) retOnThrows.object;
if (retOnThrows.state == I_THROWS) throws (Throwable) retOnThrows.object;
throws cause;
}
}
}

整体架构


模块激活
模块冻结
模块卸载
模块加载
健康检查
处理控速
签名认证
恶意检测
消息注册
消息订阅
消息发布
HTTP
WEB SOCKET
模块控制管理
模块质量管理
模块安全管理
模块消息体系
事件侦听处理（EventListener）
监听注销
监听注册
检索过滤器 Filter
Servlet容器 3.0.1
沙箱事件处理分发
EventListenerHandlers
方法调用事件
方法返回事件
方法异常事件
空处理
立即返回
抛出异常
方法内代码行触达事件
方法内调用跟踪事件
加载类检索 LoadedClass DataSource
方法调用环绕事件编织
代码执行流程干预编织
方法行路径事件编织
方法调用路径事件编织
HTTP服务器 Jetty
代码编织框架
Enhancer
JVMTI架构体系

快速安装

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

# 解压
unzip sandbox-stable-bin.zip
挂载目标应用

# 进入沙箱执行脚本
cd sandbox/bin

# 目标JVM进程33342
./sandbox.sh -p 33342
挂载成功后会提示

./sandbox.sh -p 33342
           NAMESPACE : default
             VERSION : 1.2.0
                MODE : ATTACH
         SERVER_ADDR : 0.0.0.0
         SERVER_PORT : 55756
      UNSAFE_SUPPORT : ENABLE
        SANDBOX_HOME : /Users/vlinux/opt/sandbox
   SYSTEM_MODULE_LIB : /Users/vlinux/opt/sandbox/module
     USER_MODULE_LIB : ~/.sandbox-module;
 SYSTEM_PROVIDER_LIB : /Users/vlinux/opt/sandbox/provider
  EVENT_POOL_SUPPORT : DISABLE
卸载沙箱

./sandbox.sh -p 33342 -S
jvm-sandbox[default] shutdown finished.
```

详情请查看：/jvm-sandbox/wiki (https://github.com/alibaba/jvm-sandbox/wiki)

解密阿里线上问题诊断工具Arthas和jvm-sandbox (https://www.jianshu.com/p/cda53bf5fd90)

Arthas使用指南 (https://segmentfault.com/a/1190000014618329?utm_source=tag-newest)

← PREVIOUS POST
(HTTP://39.106.3.150/ARCHIVES/JAVATRACE20190630)

NEXT POST →
(HTTP://39.106.3.150/ARCHIVES/15606790

FEATURED TAGS **(http://39.106.3.150/tags/)**

java 8 (http://39.106.3.150/tags/#java-8) java8 (http://39.106.3.150/tags/#java8)
redis (http://39.106.3.150/tags/#redis) 监控 (http://39.106.3.150/tags/#1561876610222)
全链路 (http://39.106.3.150/tags/#1561876610220) 容器 (http://39.106.3.150/tags/#1560852708518)
开源框架 (http://39.106.3.150/tags/#1560569459781) Spring (http://39.106.3.150/tags/#spring)
设计模式 (http://39.106.3.150/tags/#1559888728999) linux (http://39.106.3.150/tags/#linux)
SpringBoot (http://39.106.3.150/tags/#springboot) 大数据 (http://39.106.3.150/tags/#1559363598973)
区块链 (http://39.106.3.150/tags/#1559363594390) Java (http://39.106.3.150/tags/#java)

FRIENDS

(https://github.com/PowehiEdge)