**编 号：**

**版本号：VERX.X**

**APM项目文档**

**APM1.7.1- 插件研究**

XX公司

**2018年3月**

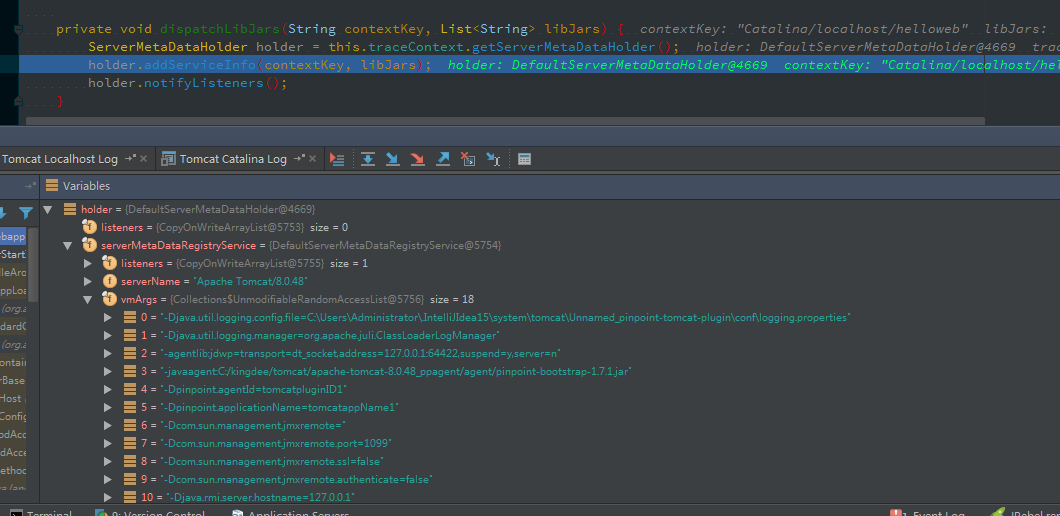
## Tomcat 插件埋点研究

### 1.1 识别当期部署应用WebappLoader

org.apache.catalina.loader.WebappLoader

|  |  |
| --- | --- |
| 版本 | 方法 |
| Tomcat6 | start |
| Tomcat7,8 | startInternal |

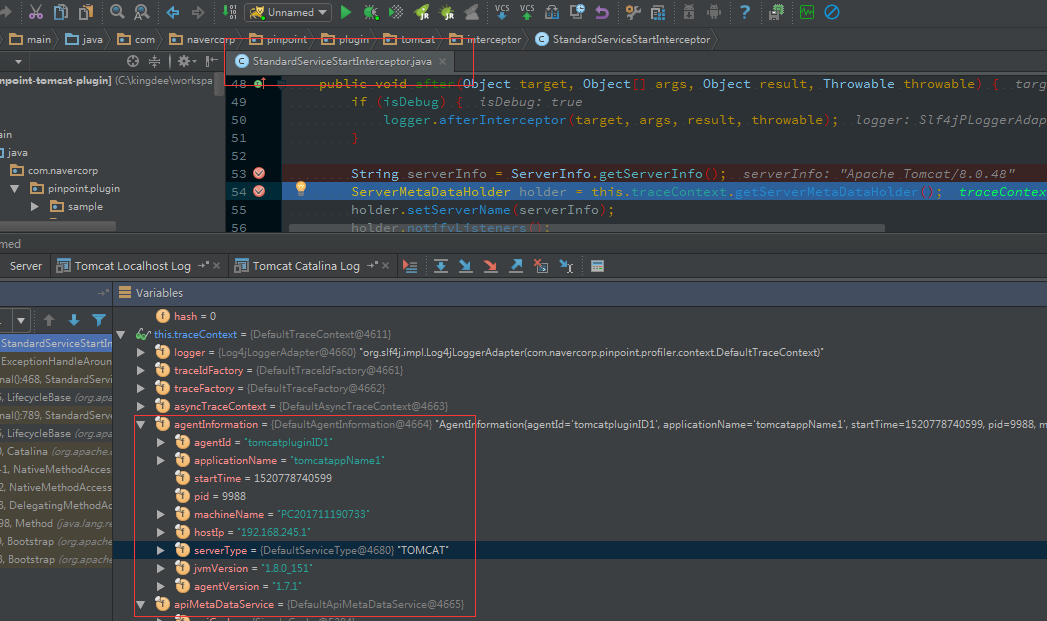
* Tomcat 自身lib jar,Tomcat版本，server.xml 中的一些配置，用户部署的应用，（可以访问的url）,用户依赖的jar.
* Jvm参数信息
* Agent 的各种信息



### 1.2拿到当前服务器信息StandardService

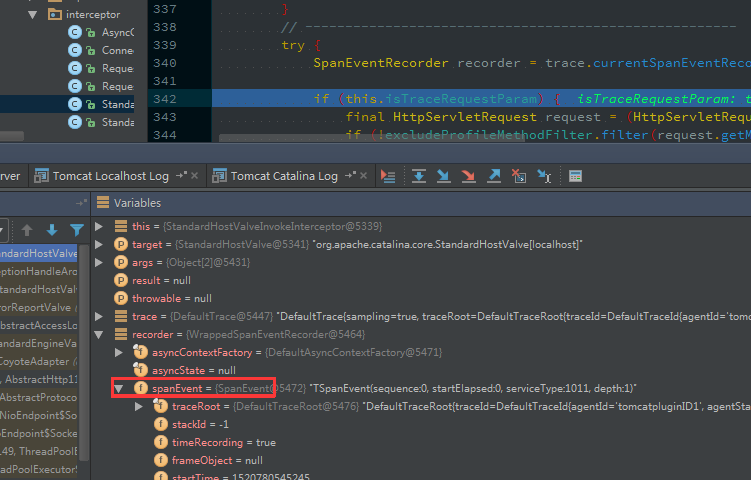
org.apache.catalina.core.StandardService

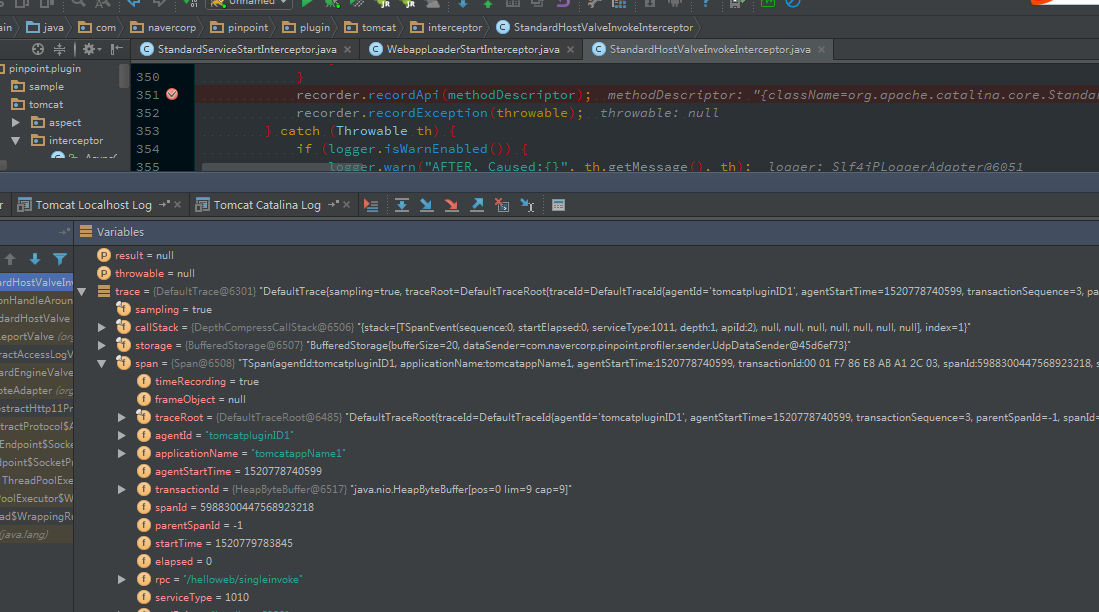
|  |  |
| --- | --- |
| 版本 | 方法 |
| Tomcat6 | start |
| Tomcat7,8 | startInternal |



### 1.3 识别每次请求，创建Trace StandardHostValve

org.apache.catalina.core.StandardHostValve

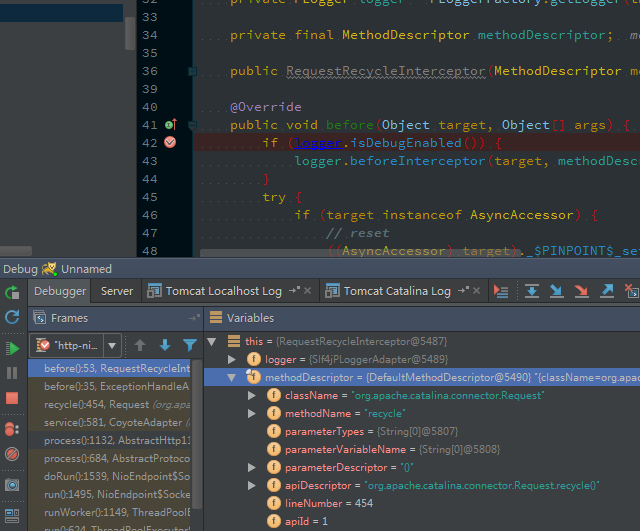




这个是链路跟踪，你每次调用都会记录，你访问 的信息，封装到span.

### 1.4、关键Request(同步、异步)

org.apache.catalina.connector.Request



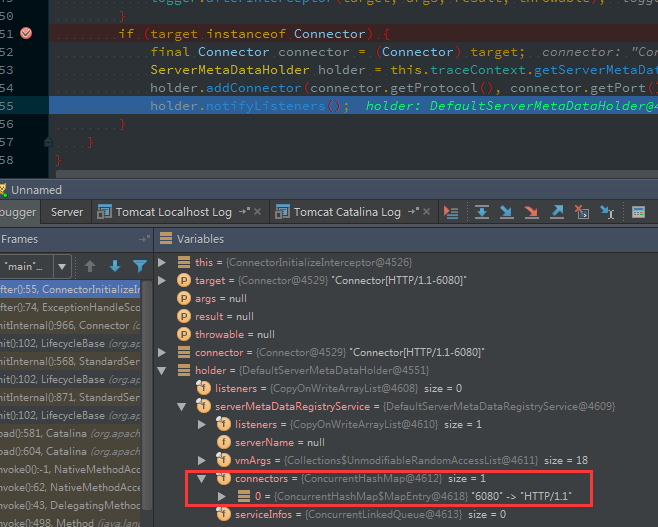
关于Request 这块

### 1.5、处理异步

略

### 1.7、端口启动监听Connector

org.apache.catalina.connector.Connector



## MYSQL 插件

### 2.1数据库连接Connection

com.mysql.jdbc.Connection

com.mysql.jdbc.ConnectionImpl

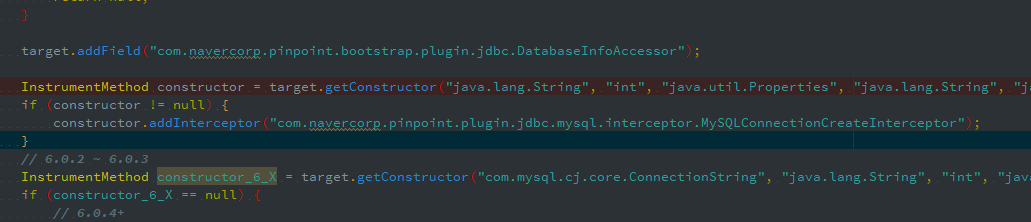
com.mysql.cj.jdbc.ConnectionImpl(6.X)

方法埋点：close、createStatement（各种重载）、prepareStatement、prepareCall、setAutoCommit、commit、rollback

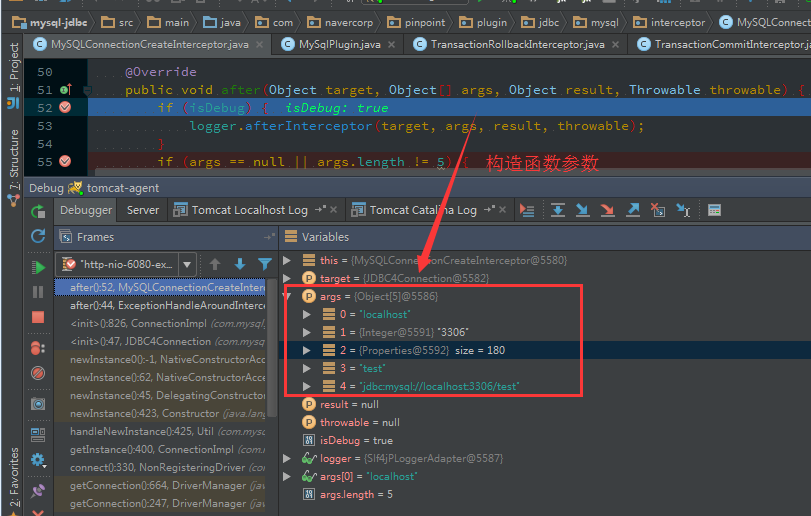
* **ConnectionImpl构造函数拦截，获取连接信息。**

拦截器：

MySQLConnectionCreateInterceptor



拦截Connection实现的的构造函数，对于mysql的ConnectionImpl

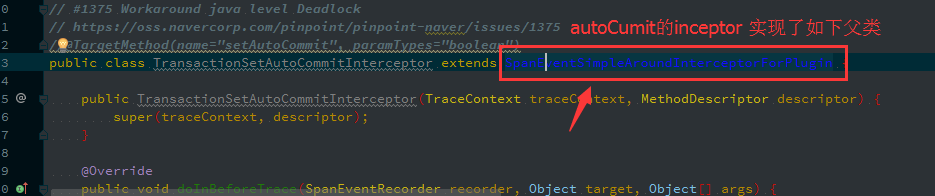
****

拦截，数据库连接信息，用数据库名，当做数据库的id.Connection是比较基础的一个类，其中包含了对数据库的各种设置 信息。

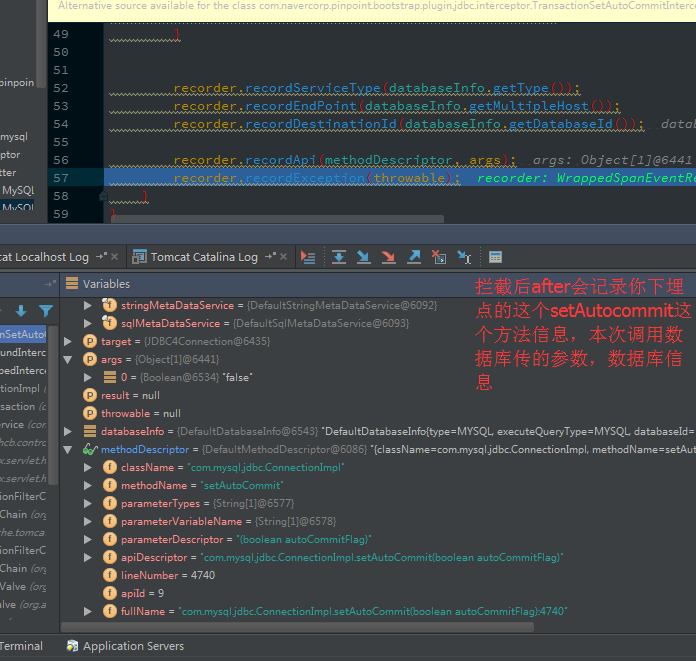
* **SetAutoCommit方法拦截,autoCommit 属性**

拦截器：TransactionSetAutoCommitInterceptor

值得注意的是该拦截器继承了SpanEventSimpleAroundInterceptorForPlugin父类。



会获取一个span信息，记录到recoder.



* **预编译sql prepareStatement(String sql) 截取sql信息**

prepareSatatement 的一个重载方法的参数是sql.

拦截器：

PreparedStatementCreateInterceptor

SpanEventSimpleAroundInterceptorForPlugin

这个拦截器给我的感觉是截取每条预处理的sql.

* 提交Commit() 埋点

思考：预处理截取了sql 信息，如果用户commit 没有执行，或执行了rollback，那截取的sql 应该就没用了，说明对于这块，agent/collector 一定有判断处理。

###### 2.1.3子目录。。。。

### 2.2数据库驱动相关：NonRegisteringDriver

com.mysql.jdbc.NonRegisteringDriver

|  |  |
| --- | --- |
| NonRegisteringDriver用处 |  |

* connector

该方法是根据数据库url,和数据库连接相关的参数返回一个java.sql.Connection对象

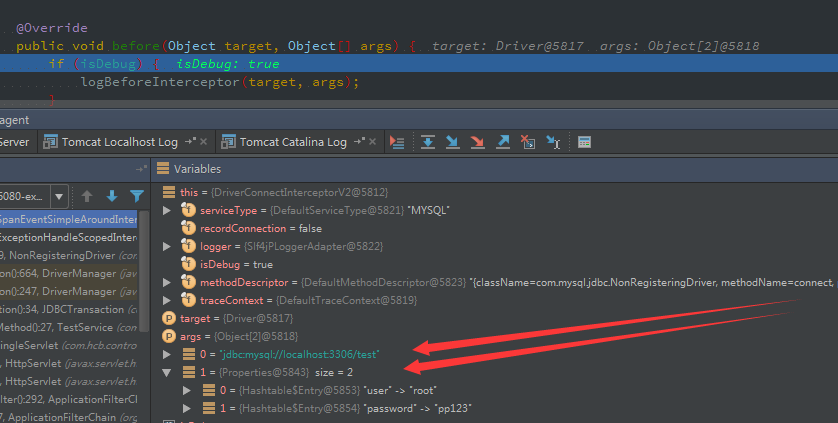
参考：

拦截器：

DriverConnectInterceptorV2

SpanEventSimpleAroundInterceptorForPlugin

可以看到方法参数

.

### 2.3 执行sql，获取结果结果集

com.mysql.jdbc.StatementImpl

com.mysql.jdbc.Statement

* executeQuery
* executeUpdate
* execute

### 2.4执行sql，获取结果结果集PreparedStatement

拦截器：PreparedStatementExecuteQueryInterceptor

* executeQuery
* executeUpdate
* execute

2.5执行sql , 获取结果集CallableStatement（继承了PreparedStatement）

* executeQuery
* executeUpdate
* execute

同上，区别参考：http://blog.csdn.net/qq\_22854537/article/details/51520392

### 2.5 执行sql JDBC4PreparedStatement

### JDBC4PreparedStatement 继承了PreparedStatement

PreparedStatementBindVariableInterceptor

* setRowId
* setNClob
* setSQLXML

### 2.6 执行sql JDBC4CallableStatement

com.mysql.jdbc.JDBC4CallableStatement

* setRowId
* setNClob
* setSQLXML

## jedis插件

### 3.0 jedis 源码了解

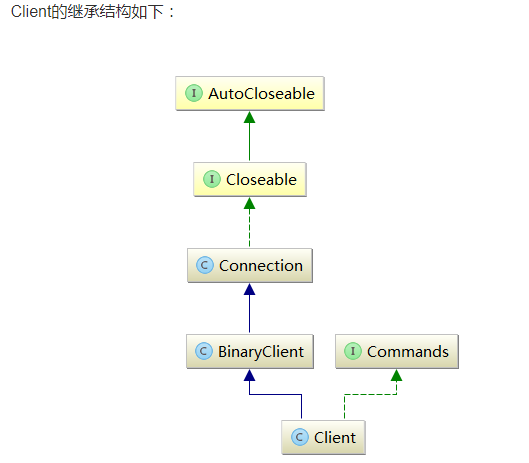


核心类有：

Jedis、BinaryJedis、Commands接口、Client

Jedis 继承了BinaryJedis 同时实现了一系列的Commands接口，BinaryJedis里主要和redis Server进行交互，一系列Commands接口主要是对redis支持的接口进行分类，像BasicCommands主要包含了info、flush等操作，BinaryJedisCommands 主要包含了get、set等操作，MultiKeyBinaryCommands主要包含了一些批量操作的接口例如mset等。

Jedis因为继承了BinaryJedis，大部分的操作都是在BinaryJedis中实现的，在BinaryJedis的构造方法中就实例化了Client。



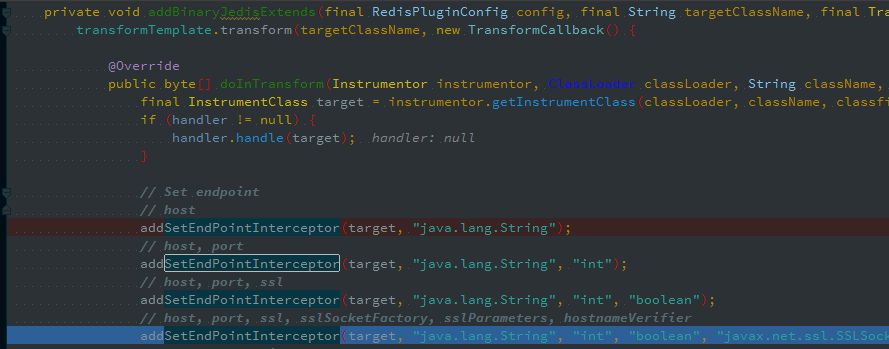
BinaryJedis中的方法主要是对Client做了代理，Client继承了BinaryClient，BinaryClient继承了Connection，实现了Commands接口。Client主要做了一些编解码的工作，BinaryClient做了Command的发送操作，而所有与redisServer交互的工作由Connection完成。

Connection 中用Protocol的sendCommand方法进行发送：

### 2.1 和redis server 交互redis.clients.jedis.BinaryJedis

BinaryJedis里主要和redis Server进行交互.

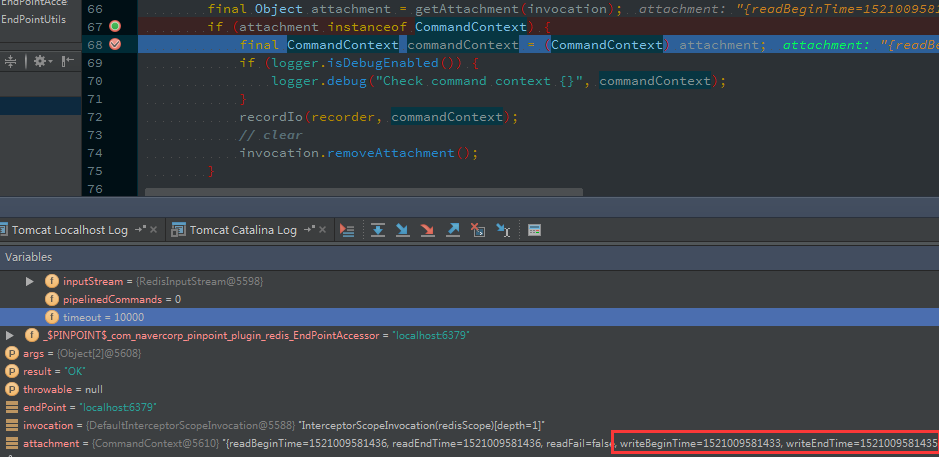
该类处理埋点先获取构造函数，但是本非下述代码中的构造函数都有，（if判断处理了。）

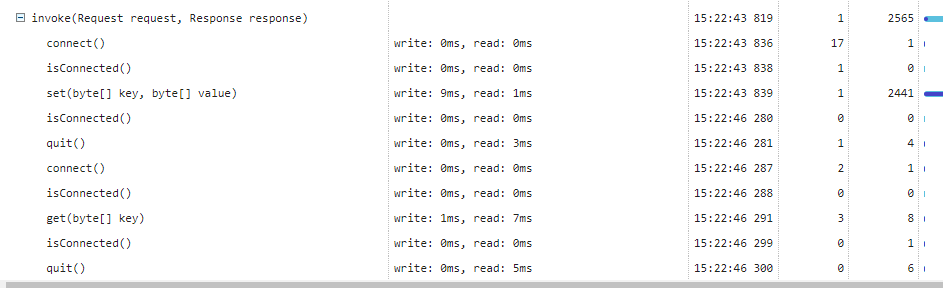


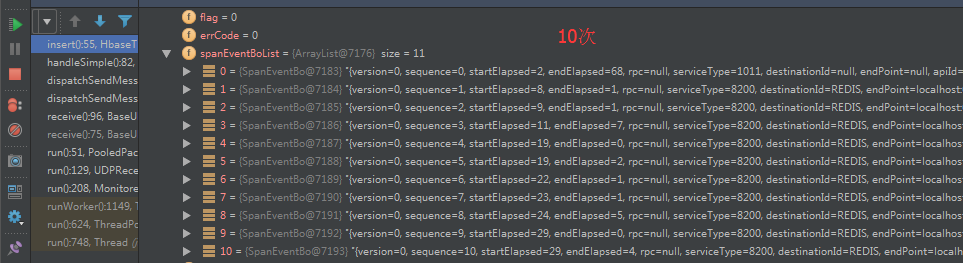
一次redis 的set操作，在插件中断点执行5次，分别对应下图中的5个方法堆栈，其中第3次为set(key,value)的操作，耗时恰好符合堆栈。

Jedis read&&write 耗时

下图是第3次方法堆栈。



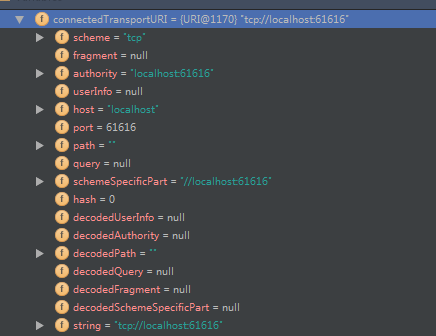




## activemq插件

### 获取连接属性：connectedTransportURI

org.apache.activemq.transport.failover.FailoverTransport ===>connectedTransportURI



org.apache.activemq.transport.tcp.TcpTransport

==========> socket

