技术案例大赛丨基于jvm-sandbox-repeater的流量回放测试实践

本文已参与2023技术案例大赛，点击查看原文

随着业务不断增长，测试用例、脚本的维护成本也再不断增高。往往研发同学修改1个接口参数或数据库字段，测试需要修改N倍以上的测试用例或脚本。而且在复杂业务接口中，有时候修改者也很难全面评估改动的影响范围，这就导致测试需要全面回归接口甚至模块化的功能。

为了解决上述的测试痛点问题，我们考虑引入流量回放技术来提效回归测试。

原文作者：钱叶侃ykqian

一、什么是流量回放

通过录制生产环境的真实流量，在测试环境模拟流量请求，通过对比回放请求与真实请求的子调用差异，来达到验证代码正确性的目的。

这种技术可以通过自动化录制和批量回放真实流量的方式，实现快速回归测试，以降低因代码变动导致的整体系统质量风险。

二、Jvm-Sandbox-Repeater介绍

1.工具简介

Jvm-Sandbox-Repeater，是jvm-sandbox实现动态拦截（AOP）沙盒工具下的一款模块，支持jvm系语言，具备jvm-sandbox所有特性。

经过调研，repeater具有以下优势：

- 支持多种请求类型插件：HTTP、dubbo、java等；
- 可实现动态代码侵入，无需重新构建服务端，适合服务众多的业务；
- 可配置化的子调用MOCK能力，专注于业务代码本身的校验；
- Github开源项目自带可视化控制台Demo，便于测试人员使用。

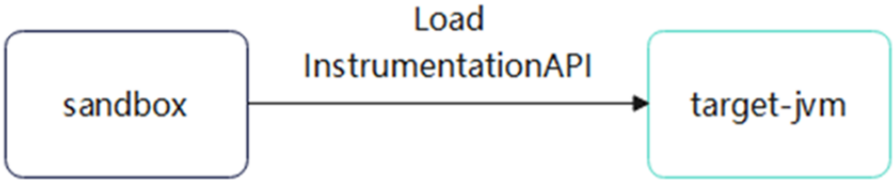
2.Sandbox简介

在解释repeater插件前，先简单描述下Sandbox的底层原理。

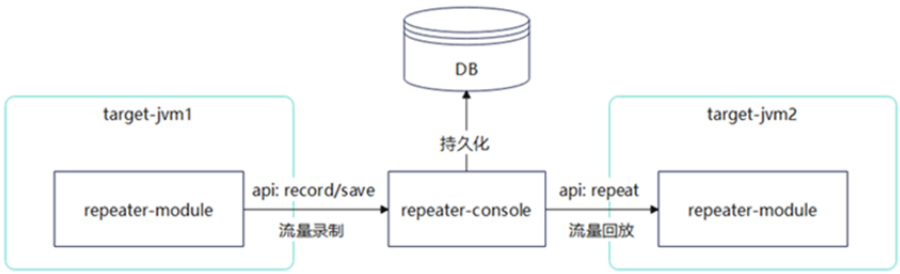
（1）侵入方式 sandbox与目标jvm建立连接实现通信主要有两种方式：agent或者attach模式。

- 当sandbox可伴随jvm一同启动时，使用agent模式。该模式通过java agent将沙盒和模块类库注入到jvm中。
- 当jvm已经启动，可采用attach模式，通过attach api向目标jvm注入代码。

两者最终都是通过对InstrumentationAPI调用来实现实时的字节码转换和增强。



3.Repeater模块简介



・jvm-sandbox-repeater模块依附sandbox注入目标jvm；

・repeater-console可进行监听配置、流量管理、流量回放等操作；

・通过repeater-module加载插件，基于请求入口、类、方法等配置监听目标服务，来实现目标jvm不同请求类型的流量录制；

・repeater-console流量录制后，写入数据库持久化，便于后续在其他jvm进行回放。

三、Jvm-Sandbox-Repeater使用实践

1. 部署

jvm-sandbox-repeater部署分为以下几个部分：

・数据库：mysql，用于数据持久化；

・控制台：repeater-console部署，console中已包含core、plugin等模块；

・注入器：sandbox和repeater模块安装（部署至目标服务节点）。

（1）源码下载

获取官方源码：https://github.com/alibaba/jvm-sandbox-repeater；

```
git clone https://github.com/alibaba/jvm-sandbox-repeater.git
```

官方源码长时间未维护，存在一些bug，可联系本文作者获取修改版源码。

（2）创建数据库

a.数据库SQL文件获取地址jvm-sandbox-repeater/repeater-console/repeater-console-dal/src/main/resources/database.sql；

b.使用sql完成数据库与表创建。

（3）console修改、构建与部署

a.修复console页面访问报错

・修改 ReplayController.java（回放详情页）；

```
return "/replay/detail";
// 修改为
return "replay/detail";
```

PATH:

jvm-sandbox-repeater/repeater-console/repeater-console-start/src/main/java/com/alibaba/repeater/console/start/controller/page/ReplayController.java

・修改 RegressPageController.java（测试页）；

PATH:

jvm-sandbox-repeater/repeater-console/repeater-console-start/src/main/java/com/alibaba/repeater/console/start/controller/test/RegressPageController.java

```
return "/regress/index";
// 修改为
return "regress/index";
```

b.编译构建console

```
cd jvm-sandbox-repeater
# 增加跳过测试命令避免构建因测试不通过失败
mvn clean install package -Dmaven.test.skip=true
```

构建后jar包路径：

/jvm-sandbox-repeater/repeater-console/repeater-console-start/target/repeater-console.jar

c.部署console

・修改properties配置文件-application-test.properties，结合业务服务器实际，修改port和DataSource；

・编写start.sh启动脚本；

```
#!/bin/sh
nohup java -jar repeater-console.jar --spring.profiles.active=test > /data/logs/tjzt_repeaterconsole_logs/start.log  2>&1 &
```

・启动服务。

```
sh start.sh
```

d.repeater-console访问地址

```
http://{host}:{port}/regress/index.htm
```

（4）console流量监听与回放配置

这里我们先写一个简单的接口服务来做示例，结合示例说明如何配置repeater的服务监听。

```
@RestController
public class GreetingController {

private static final String template = "Hello, %s!";
```

```
private final AtomicLong counter = new AtomicLong();

@RequestMapping("/greeting")
public Greeting greeting(@RequestParam(value="name", defaultValue="iflyrecTester") String name) {
    // incrementAndGet 函数作用只是起到计数和+1的作用，用于区分请求
    return new Greeting(counter.incrementAndGet(), String.format(template, name));
}
```

a.进入console控制台的配置管理–新增配置页面；

b.应用名和环境均设置为unknown；

设置为unknown是因为通过attach模式注入，通常获取不到应用名称等信息，默认值为unknown。后续有说明如何设置具体的应用名和环境名。

c.官方配置信息如下，可结合注释查看。

```
{
  "useTtl" : true, // 是否开启ttl线程上下文切换（开启之后，才能将并发线程中发生的子调用记录
    下来，否则无法录制到并发子线程的子调用信息）
  "degrade" : false, // 模块降级，开启后只回放不录制
  "exceptionThreshold" : 1000,  // 感知到异常次数超过阈值后，会降级模块
  "sampleRate" : 10000, // 采样率，10000=100%，必须是整数
  "pluginsPath" : null, // 插件地址，默认写null
  "httpEntrancePatterns" : [ "^/greeting.*$" ], // 需要录制的http请求接口白名单，支持正则表达式匹配
  "javaEntranceBehaviors" : [], // java入口插件动态增强行为（需要录制+正常回放的流量）
  "javaSubInvokeBehaviors" : [{ // java子调用插件动态增强行为（需要录制+MOCK回放的流量）
    "classPattern" : "hello.GreetingController",     // java子调用-类匹配器，支持正则
    "methodPatterns" : [ "greeting" ],  // java子调用-方法匹配器，支持正则
    "includeSubClasses" : false // 是否匹配子类
  }],
  "pluginIdentities" : [ "http", "java-entrance", "java-subInvoke", "mybatis", "ibatis" ],
   // 启用的插件，注意上面的http、javaEntrance、javaSubInvoke需要在这里配置了才会生效
  "repeatIdentities" : [ "java", "http" ]   // 回放器插件
}
```

说明：此配置是针对上方示例服务设置的，可以录制并mock回放/greeting接口

关于回放的说明

①javaSubInvokeBehaviors中配置需要mock的调用，javaEntranceBehaviors中配置无需mock的调用；

②匹配器主要通过classPattern和methodPatterns组合匹配，如果希望匹配某controller类下所有方法，一定要在methodPatterns里写成["*"]，而不是网络资料里的[""]。

（5）sandbox–repeater部署

repeater模块主要部署到目标jvm的服务器节点，用于部署给录制用jvm和回放用的jvm。

下面介绍手动部署步骤。

a.下载jvm–sanbox[sandbox–1.3.3–bin.tar]和jvm–sandbox–repeater module[repeater–stable–bin.tar]

```
# 下载地址
https://github.com/alibaba/jvm-sandbox-repeater/releases/
```

b.将sandbox–1.3.3–bin.tar解压至~/

```
cd ~
tar xvf sandbox-1.3.3-bin.tar
```

解压后，当前用户目录下会出现sandbox文件夹~/sandbox。

c.将repeater–stable–bin.tar解压至~/，并将解压出来的repeater目录重命名为.sandbox–module

若当前用户目录下已存在sandbox模组目录，则将repeater目录所有文件移动到.sandbox–module中。

```
cd ~
tar xvf repeater-stable-bin.tar
mv repeater ./.sandbox-module
```

d.修改repeater模组配置文件

文件路径： ~/.sandbox–module/cfg/repeater.properties

将配置文件中url指向console部署服务器，注意端口号与console配置的port保持一致。

```
# 录制消息投递地址
broadcaster.record.url=http://127.0.0.1:8050/facade/api/record/save
# 回放结果投递地址
broadcaster.repeat.url=http://127.0.0.1:8050/facade/api/repeat/save
# 回放消息取数据地址
repeat.record.url=http://127.0.0.1:8050/facade/api/record/%s/%s
# 配置文件拉取地址
repeat.config.url=http://127.0.0.1:8050/facade/api/config/%s/%s
# 心跳上报配置
repeat.heartbeat.url=http://127.0.0.1:8050/module/report.json
# 是否开启脱机工作模式
repeat.standalone.mode=false
# 是否开启spring advice拦截
repeat.spring.advice.switch=false
```

e.将sandbox注入目标jvm（attach方式）

- 获取目标jvm PID；

```
ps -ef|grep {target jvm}.jar
```

· 启动sandbox；

```
cd ~/sandbox/bin/
# 建议-P端口号使用jvm_pid前方加一个数字，比如业务服务端口9031，给sandbox就使用49031
sh sandbox.sh -p {jvm_pid} -P {任意未被占用的端口}
```

–P：目标jvm的进程id，–P：sanbox端口：

```
# 若同一台设备需要启动多个沙盒，用-n命令区分命名空间，空间命名建议用服务名【可能存在bug】
sh sandbox.sh -p {jvm_pid} -P {任意未被占用的端口} -n {namesapce}
```

· 启动成功后，控制台输出如下；

```
[root@t241237 bin]# sh sandbox.sh -p 15529 -P 8060
                    NAMESPACE : default
                      VERSION : 1.3.3
                         MODE : ATTACH
                  SERVER_ADDR : 0.0.0.0
                  SERVER_PORT : 8060
              UNSAFE_SUPPORT : ENABLE
                 SANDBOX_HOME : /root/sandbox/bin/..
            SYSTEM_MODULE_LIB : /root/sandbox/bin/../module
              USER_MODULE_LIB : /usr/local/sandbox/sandbox-module;~/.sandbox-module;
          SYSTEM_PROVIDER_LIB : /root/sandbox/bin/../provider
           EVENT_POOL_SUPPORT : DISABLE
```

agent模式由于需要伴随jvm启动，业务价值不大因此这里未作深入研究，感兴趣的读者可以自行尝试

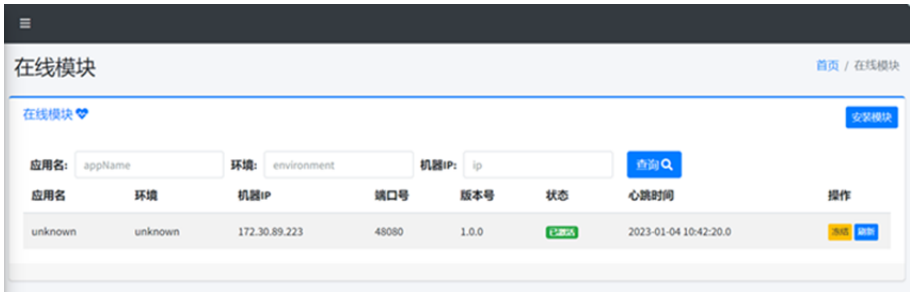
· 若需要停止sandbox，使用–S命令。

```
cd ~/sandbox/bin/
sh sandbox.sh -p {jvm_pid} -S

# 如果启动时使用了-n命令，停止时也需要加上-n
```

2. 通过console实现录制/回放

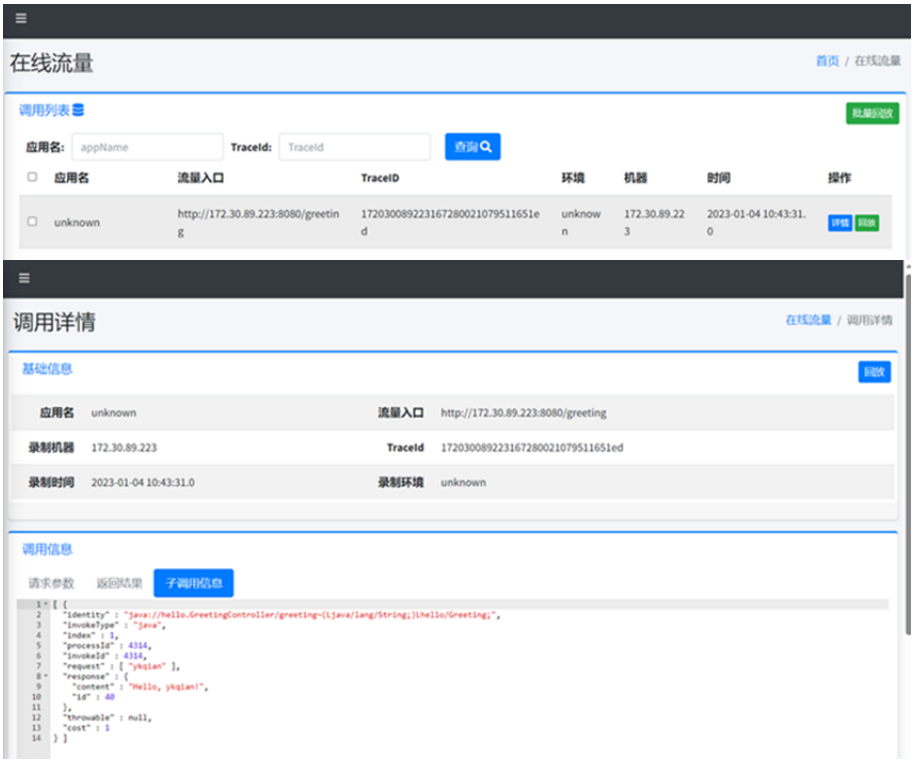
a.完成上述部署步骤，进入控制台–在线模块可以查看到成功注入jvm的repeater模块心跳（greeting示例）。



b.给被注入jvm的服务发送一次流量。

```
http://{host}:8080/greeting?name=ykqian
```

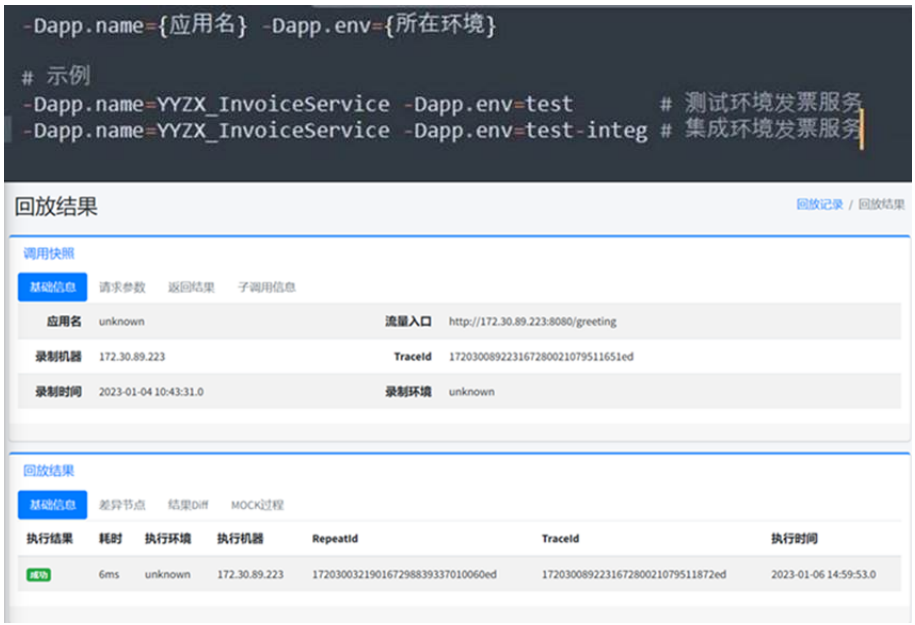
c.进入console 在线流量页面，可以查看到刚刚录制好的流量，此页面可以查看流量详情也可以直接点击回放。



d.流量回放直接点击录制记录后方的回放即可回放，且自动完成断言。

3. 环境与应用名区分

从上面的截图可以看到，监听的服务和环境名称都是unknown，不便于管理和区分。这里笔者提供一种解决办法。



在jvm需启动时增加自定义参数，启动命令java后增加参数格式。

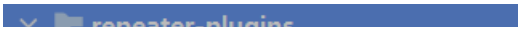
但是使用此方法，导致jvm必须重启，违背了attach模式的初衷。目前笔者思考的方法是通过修改服务监听配置代码，增加服务名称和环境设置，但本篇主要为实践经验未作深入探索，感兴趣的读者可自行尝试。

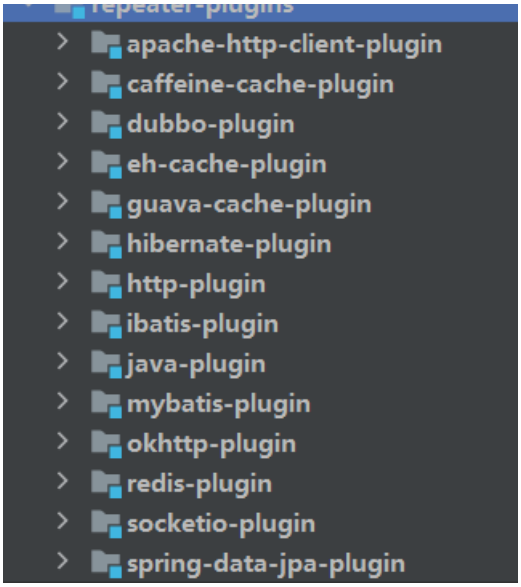
4. Mock子调用的探索

（1）支持mock的子调用方式

第三方组件中，mysql（ibatis、mybatis、hibernate等）、redis（jedis）数据库支持mock，es暂不支持。请求协议中，http、websocket支持mock。

现有的插件合集





（2）mock策略

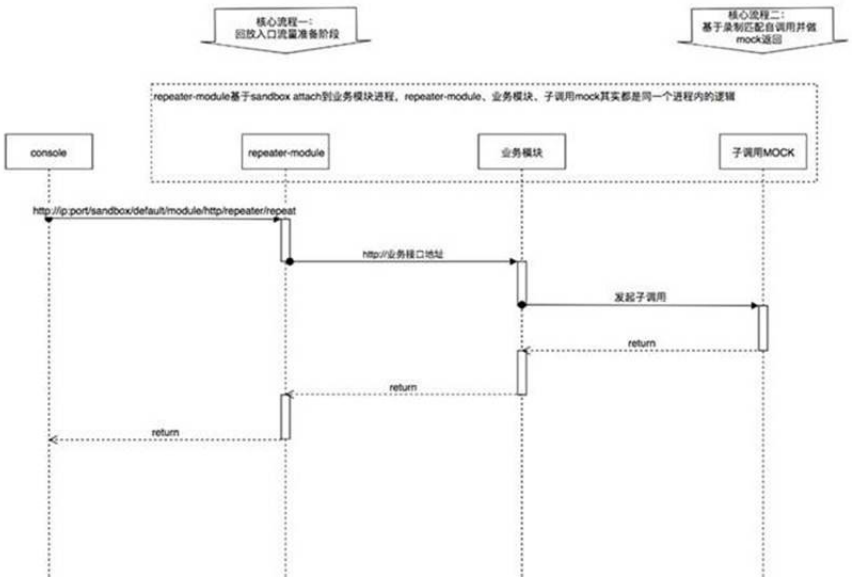
repeater有3种mock策略，分别是：

・默认策略：拦截所有子调用，全部mock；

・参数匹配策略：根据子调用参数相似度，若命中直接返回当时的录制结果，否则返回相似度最高一条（console中使用的策略）。具体源码可见：com/alibaba/jvm/sandbox/repeater/plugin/core/impl/spi/ParameterMatchMockStrategy.java。

・反射比对策略：首先使用URI精确匹配，收集所有匹配到的子调用。再遍历所有收集的子调用，通过反射进行参数比对。若参数无差异直接返回匹配成功，否则返回差异最少的一条子调用。具体源码可见：com/alibaba/jvm/sandbox/repeater/plugin/core/impl/spi/ReflectCompareStrategy.java。

（3）mock回放流程



回放时业务代码正常处理，仅mock子调用，可参考上述时序图（来源互联网）。

具体源码可见：com/alibaba/jvm/sandbox/repeater/plugin/core/impl/AbstractInvocationProcessor.java。

①RepeatCache.getRepeatContext(traceId)获取录制记录。

②判断是否是子调用mock。

③构建mock请求。

④根据mock策略，执行mock请求，获取mock响应。

⑤根据mock响应结果执行后续处理（跳过、立即抛出返回、立即抛出异常）。

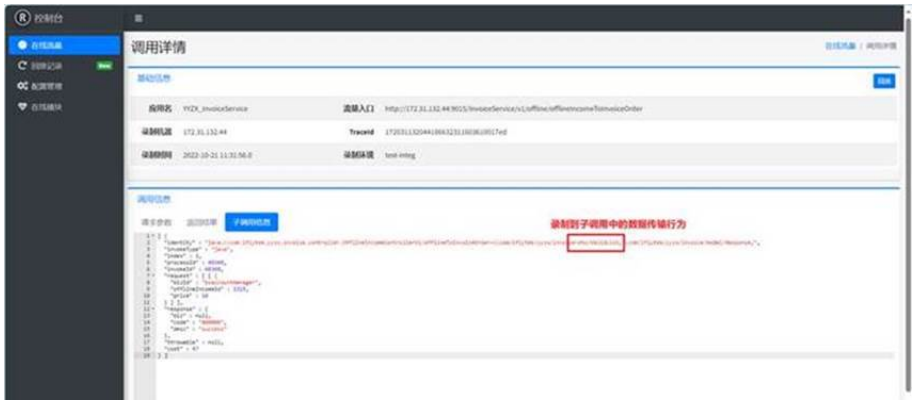
（4）console中mock实践

以发票服务中线下收入转发票接口为例

・集成环境录制

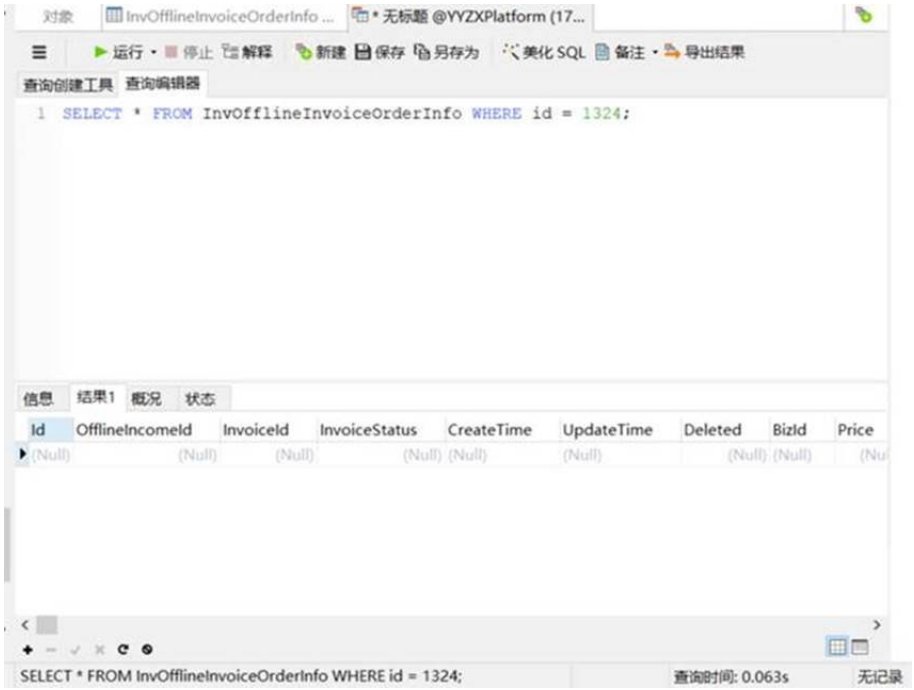
从图中可以看出，在集成环境，我们录制了一个线下收入转电子发票的接口流量：线下收入的id为1324，价格

为10。



・检查测试环境

检查数据库，测试环境没有对应的id为1324的线下收入记录。



・测试环境发起回放，并选择开启mock

测试环境虽然没有该ID的线下收入，在测试环境回放时，由于向测试环境发起的回放子调用参数完全一致（参数匹配MOCK策略），命中了先前在集成环境录制的子调用请求参数，因此直接返回mock结果。

・校验测试环境数据库

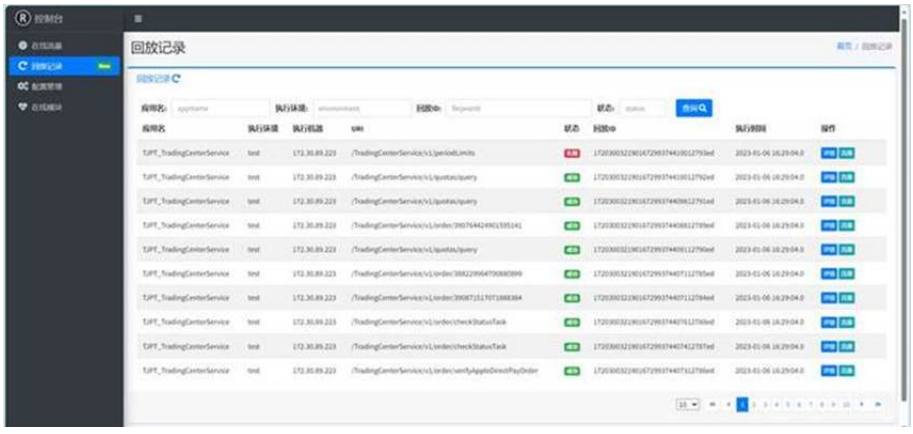
再次检查测试环境数据库，没有任何新增记录，本次mock回放完成。

四、Jvm-Sandbox-Repeater优化

在实践过程中，我们发现jvm-sandbox-repeater有以下不足，并进行了优化：

・虽然有可视化控制台，但是控制台无流量回放查看入口。

解决：console前端使用velocity实现，仿照其他页面编写回放记录管理页面。



・业务服务接口过多，只有少量接口无需录制流量，但不适用通配符逐个编写匹配器过于繁琐。

解决：增加黑名单配置，在黑名单的接口不进行录制。

```
// 根据黑名单过滤http流量
List<String> blackPatterns = ApplicationModel.instance().getConfig().getBlackEntrancePatterns();
LogUtil.info(String.valueOf(blackPatterns));
if (matchRequestURI(blackPatterns, req.getRequestURI())) {
    LogUtil.debug("current uri {} matched in blackEntrancePatterns, ignore this request", req.getRequestURI());
    Tracer.getContext().setSampled(false);
    return;
}
```

・部分接口流量过大且重复性高，而工具自带的监听采样率是全局生效。

解决：在http插件中实现单接口采样率，可针对单个高重复性接口进行限制。

```
// 设置接口采样率
JSONObject apiSampleRate = ApplicationModel.instance().getConfig().getApiSampleRate();
Integer sampleRate = 0;
// 配置中存在该URI则使用对应采样率，否则使用全局采样率
// 注意，最终总采样率是 api采样率 * 全局采样率
if (apiSampleRate.getString(req.getRequestURI()) != null) {
    sampleRate = Integer.valueOf(apiSampleRate.getString(req.getRequestURI()));
    LogUtil.debug("uri {} set api sample rate {}", req.getRequestURI(), sampleRate);
} else {
    sampleRate = ApplicationModel.instance().getConfig().getSampleRate();
    LogUtil.debug("uri {} not set api sample rate, use global sample rate", req.getRequestURI());
}
if (sampleRate != null && sampleRate > 0 && isSample(sampleRate)) {
    LogUtil.debug("this request sampled, uri {}", req.getRequestURI());
    Tracer.getContext().setSampled(true);
} else {
    LogUtil.debug("current uri {} sample rate {} invalid or missed, ignore this request", req.getRequestURI(), sampleRate);
    Tracer.getContext().setSampled(false);
    return;
}
```

经过优化，服务监听回放配置可参考如下结构设置。

```
{
    "useTtl": true,
    "degrade": false,
    "exceptionThreshold": 1000,
    "sampleRate": 10000,
    "pluginsPath": null,
    "httpEntrancePatterns": ["^/greeting.*$"],
    // [new] http接口黑名单，命中后不录制
    "blackEntrancePatterns": ["^/greeting.*$"],
    // [new] 接口采样率设置，命中后按照接口采样率*全局采样率录制，未命中按照全局采样率
    "apiSampleRate": {"/greeting":5000},
    "javaEntranceBehaviors": [],
    "javaSubInvokeBehaviors": [{
            "classPattern": "hello.GreetingController",
            "methodPatterns": ["*"],
            "includeSubClasses": false
        }
    ],
    "pluginIdentities": ["http", "java-entrance", "java-subInvoke", "mybatis", "ibatis"],
    "repeatIdentities": ["java", "http"]
}
```

五、待优化项

经过实践，笔者认为工具存在以下痛点：

・**【待优化】**console不支持流量编辑和断言编辑，若请求发生变化（如header中的鉴权），必定认为回放失败；可以考虑在console中增加流量和断言规则的编辑；

・**【待优化】**冗余流量过多，完全依赖采样率可能会造成漏采问题，因此建议流量增加清洗或精细化录制能力，来实现测试效率提升；

・**【待优化】**服务监听配置复杂，上手成本高，可以考虑做成可视化配置操作；

如果能够完成上述优化，使工具其更加稳定、易用，那它必定能够更好地应对复杂的业务场景。

六、总结

Jvm-Sandbox-Repeater作为一款流量回放工具，其拥有多类型请求支持、丰富的子调用mock、动态化的代码侵入等丰富特性，能够有效完成基本的流量回放事务。在探索过程中，我们在4个业务服务上尝试应用该工具，录制流量共30000+，回放流量2800+，覆盖接口100+。对比手工测试，预估工具在回归测试的效率提升上能达到约30%。