目 录 (五十五期·下)

浅谈银行开放平台应用系统性能调优

作者：赵俊杰、张鸿、严晨光

随着 5G、大数据、人工智能、生物识别等信息技术的高速发展，传统商业银行均加快了数字化和智慧化转型步伐。开放平台应用系统作为数据化银行的核心，其性能高低不仅关系着银行的声誉，也影响着转型的进程。本文针对银行开放平台应用系统的特点，介绍了常用性能监控工具及常见性能问题调优方法，并给出了具体的应用实例，希望对测试人员性能调优能有所帮助。

一、银行开放平台应用系统特点

开放平台应用系统泛指依托 X86 服务器、小型机、F5、存储等硬件设备和 Linux、Unix、Windows 等系统软件构建的应用系统，也泛指除 IBM 大型主机平台应用系统以外的所有应用系统。银行开放平台应用系统具有以下特点：一是应用系统数量多且涉及业务复杂，银行应用系统涉及客户营销、产品、渠道、运营、风控、经营管理和数据服务等业务领域；二是应用系统技术架构和软件多样化，系统中包含了 PASS 云平台、WebSphere、Tomcat、Oracle、Mysql、Redis、Gbase 以及 Kafka、Hbase、Hive 等大数据平台生态圈的诸多软件，系统架构更是千差万别；三是核心系统和渠道类系统用户量或交易量大，对性能要求高；四是系统之间关联关联复杂，测试环境配置复杂且性能问题定位难度较大。

二、常用性能测试指标及监控工具

1、常用性能指标

银行应用系统性能指标通常包含括但不限于并发用户数或 TPS、响应时间、成功率、资源利用率。其中，交易量较大的系统建议重点关注 TPS，交易量较小的系统建议重点关注并发用户数，响应时间取值为压力测试工具测试结果中 90%用户响应时间，成功率要求为 99.6%以上，应用服务器和数据库服务器 CPU 和内存使用率一般不超过

80%。此外，可根据系统特点和测试目标酌情关注其它性能指标，如：监控类系统可关注 TCP 连接数、网卡流量等性能指标。常用性能指标监控工具列表 1 如下。

表 1 常用性能指标监控工具

监控类型	监控资源名称	监控软件或命令
操作系统	Linux	nmon、top、vmstat、netstat、iostat
	Windows	性能监视器
	AIX	nmon、topas、vmstat、sar、svmon、iostat、netstat
中间件	WebSphere	WAS 控制台-性能监视基础架构、jvisualvm、jconsole
	Tomcat	tomcat status、jvisualvm、jconsole
	IIS	性能监视器
	CICS	CSTD
数据库	Oracle	OEM、AWR 报告
	Sybase	sp_sysmon、Proactive DBA
	DB2	Db2top
	Mysql	show_global satus like “监控项”
	SqlServer	性能监视器
	Redis	info
大大数据平台相关	-	中兴 DAP Manager（可监控中兴 ZDH 所有组件）
应用程序	-	商业应用性能管理工具：Dynatrace、New Relic、听云、OneAPM、ARMS 等，开源应用性能管理工具：Pinpoint、SkyWalking、Zipkin、CAT 等。

三、常见性能问题及调优方法

性能调优是对系统软硬件各个方面的设置和关联进行调整和优化，使之发挥出最大的效能，从而高效支撑业务的开展。性能调优推荐按照“硬件-网络-系统配置参数-应用程序-系统架构”依次进行分析、调优。银行开放平台应用系统性能问题多集中在交易响应时间长、应用服务器或数据库服务器 cpu 利用率高、JVM 内存溢出、SQL 语句无相关索引或索引使用不当、日志级别设置不当、查询结果未分页、Oracle 数据库未使用绑定变量、应用程序与数据库连接未使用连接池等，常见问题的具体调优方法如下。

1、CPU 利用率高

CPU 利用率高通常涉及应用服务器和数据库服务器，银行应用系统中应用服务器操作系统和数据库服务器以 Linux 和 oracle 为主流，本文仅以 Linux 和 oracle 为例进行说明。

Linux 操作系统 CPU 利用率高：通过 top 命令查看 CPU 占用率高的进程，再根据进程 PID 查看 CPU 占用率高的线程 ID（top -Hp 进程 PID），将线程 ID 转换为 16 进制，生成 javacore 文件（Kill - 3 进程 PID），利用 16 进制的线程 ID 在 javacore 文件中查看线程的详细执行信息，从而定位 CPU 利用率高的应用程序代码。

Oracle 数据库服务器 CPU 利用率高：通过 top 命令查看 CPU 占用率高的进程 PID，通过进程 PID 定位出 sql 语句的 sql_id(select sql_id from v$session where paddr=(select addr from v$process where spid=’ XXXX’)),通过 sql_id 查看具体 sql(select sql_text from v$sql where sql_id=’ XXXX’), 再通过 dbms_xplan.display_awr（ ）与 sql_id（ select plan_table_out from table（ dbms_xplan.display_awr(‘XXXX’)))查看语句的执行计划，进而对 sql 语句进行优化。

2、JVM 堆内存溢出

堆是 JVM 运行时内存中最大的区域，也是和程序开发密切相关区域，所有的对象实例（包括基本类型）、数组都存放在这个区域。JVM 堆内存使用量持续增长，当增长到最大堆后，将无法分配新内存，系统出现内存溢出，WebSphere 堆内存溢出发生后，系统会自动生成堆内存转储 headdump 文件，同时 SystemOut.log 中会出现关键字” java.lang.OutOfMemoryError”，native_stderr.log 中会出现关键字” java/lang/OutOfMemoryError”。

内存溢出可利用 HeapAnalyzer 定位占用内存大的对象（如图 1 所示），同时使用 GC 日志分析工具分析详细垃圾回收日志 （native_stderr.log），可以分析出现内存溢出的过程，确认触发内存溢出的直接原因，评估垃圾回收工作的性能，找出合适的 GC 策略和调优参数。

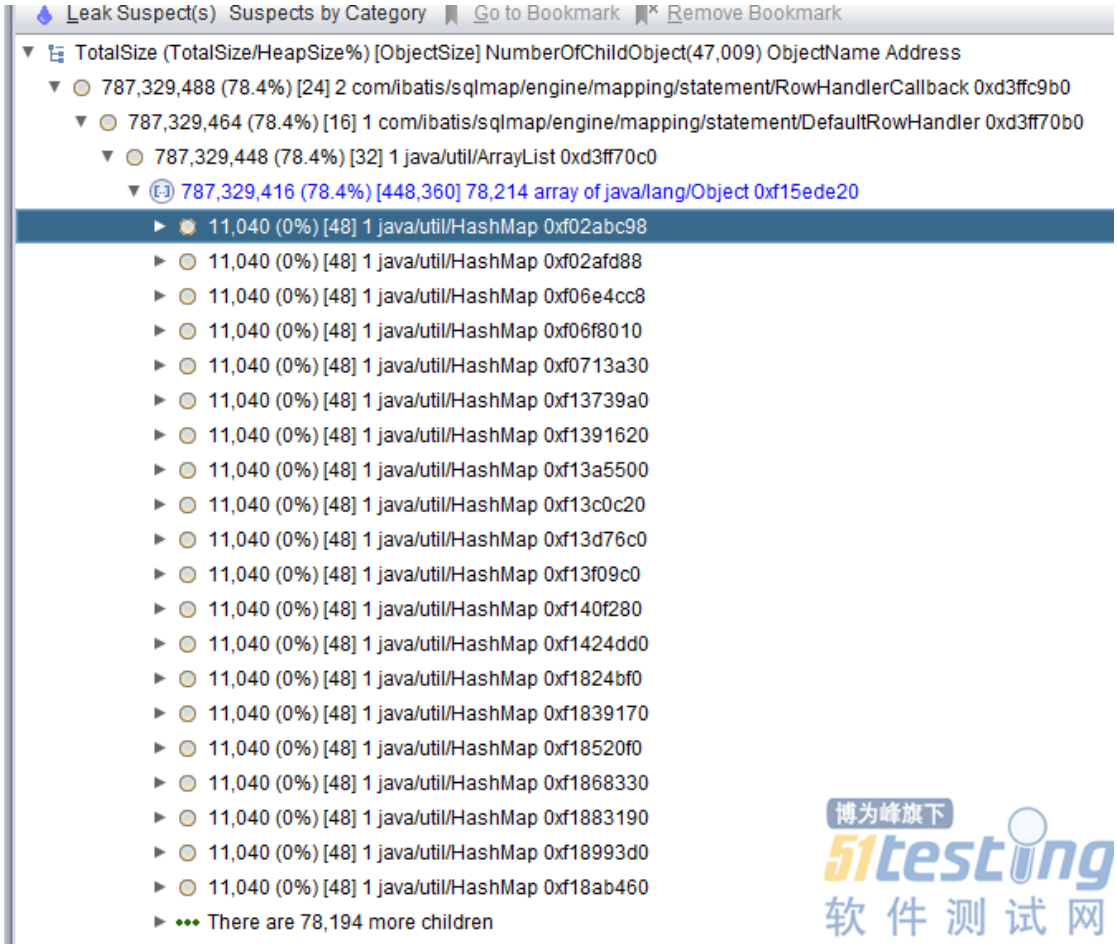


图 1 HeadDump 分析

3、Websphere 会话覆盖参数

应用服务器为 Websphere 的应用系统，如采用 Session 方式控制用户访问，在当大量用户瞬间并发访问时，由于 WebSphere 会话线程不释放或释放缓慢，可能会出现请求无响应的情况。此时，可以考虑配置 Websphere 会话管理参数，具体登录到 was 控制台，进入会话管理界面（如图 2 所示），选中该界面中的覆盖会话管理-启动 cookie，内存中最大会话量填写 3000，然后点击应用。

图 2　WAS 控制台会话管理界面

4、交易响应时间过长

使用 Dynatrace 监控工具，在 Edit system profile --> sensors --> Browse 路径下设置待测交易的类路径，通过事务流可以定位是应用服务器端响应慢还是数据库端响应慢，再查看 purepath 树，即可定位到导致响应时间长的应用程序代码或 sql，对于长 sql 可进一步分析其查询计划，查看索引使用是否合理或是否存在全表扫描，进而根据情况调整表索引或 sql 逻辑结构。

5、Oracle 未使用绑定变量

绑定变量常适用于频繁执行的 SQL 语句，且语句涉及的查询数据分布较均匀，分区较均衡。SQL 语句未使用绑定变量时，每次执行都会导致 Oracle 数据库进行硬解析，而每次硬解析都会查询一次 gv$sql 全局动态视图，进行频繁的数据库表对象的统计信息收集，造成极大的性能消耗，从而使应用程序运行耗时更长。

通过以下两个步骤能够筛选出可以使用绑定变量的 SQL 语句。

1）利用 force_matching_signature 查询可能需要使用绑定变量的语句数量:

```
select v.force_matching_signature ,count(*) from v$sql v where FORCE_MATCHING_SIGNATURE
<> EXACT_MATCHING_SIGNATURE and PARSING_SCHEMA_NAME <> 'SYS' group  by
v.force_matching_signature having count(*)>&a order by  2;
```

2）查出疑似可使用绑定变量的 SQL 语句

```
select PARSING_SCHEMA_NAME,sql_text,sql_id  from  v$sql where
force_matching_signature='XXXX';
```

获得查询结果后，可与开发人员确认是否需使用绑定变量。

四、应用实例

例 1: 某系统“项目列表查询”功能 50 用户同时运行，平均响应时间为 16.798 秒，超过了通过准则中设定的 5 秒，通过 Dynatrace 事务流分析发现，外部服务调用花费的时间占交易总时间的 90.17%（如图 1 所示）。

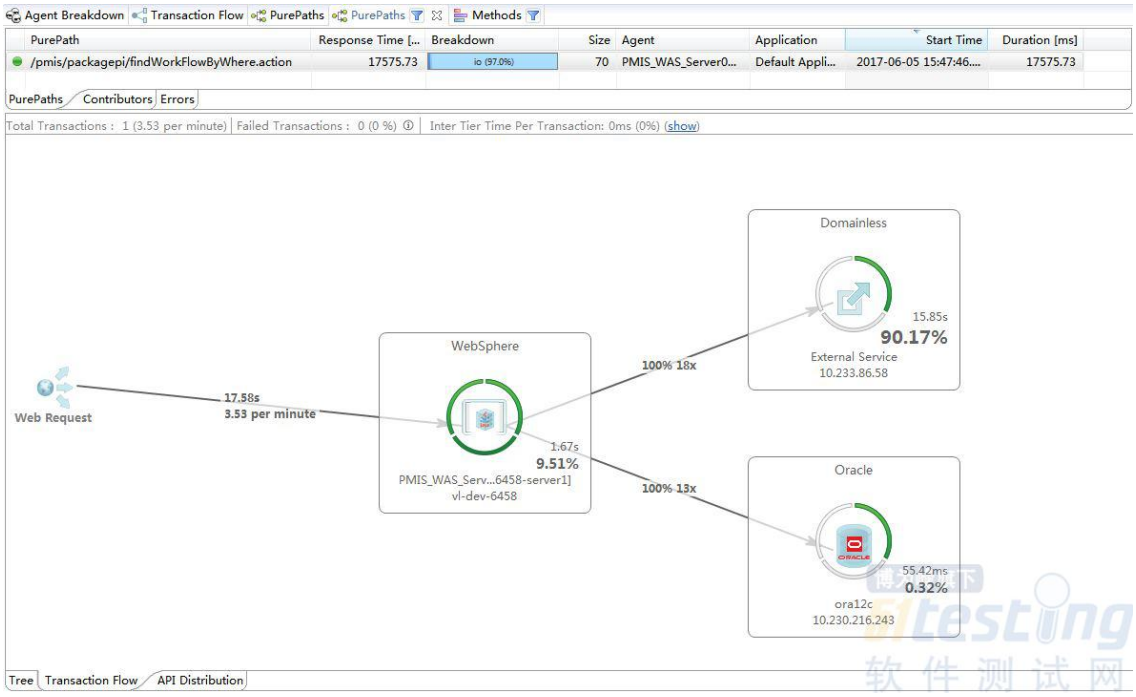


图 3 事务流图

进一步分析 purepath 树发现工作流调用耗费时间较长，具体为 getInputStream（）方法调用次数多且每次调用花费时间长（如图 2 所示）。

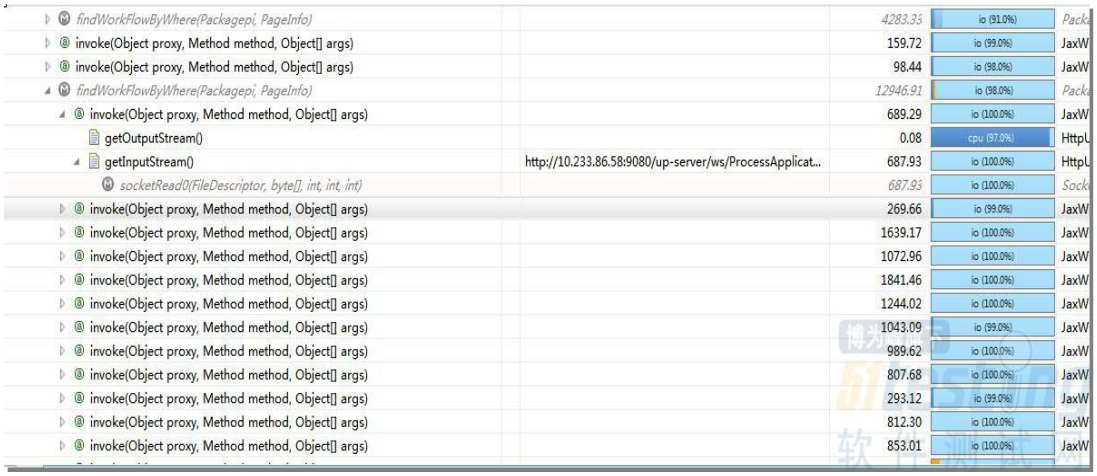
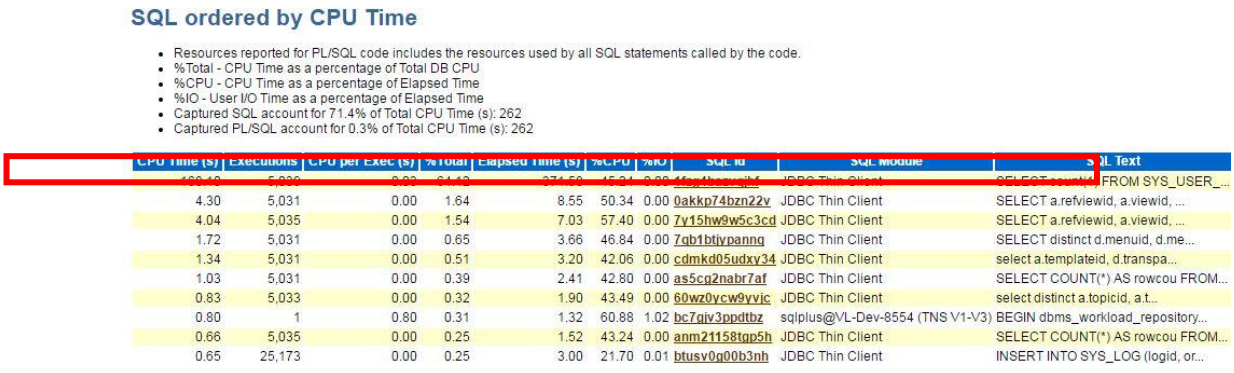


图 4　purepath 树

经分析 getInputStream（）方法相关的工作流平台，发现工作流平台上的流程版本太多，每次查询均需调用所有的版本进行处理，清理多余的流程版本后，“项目列表查询”功能 50 用户同时运行平均响应时间下降为 4.879 秒。

例 2: 某系统一查询交易并发执行时 Oracle 数据库 CPU 利用率持续为 85%左右。通过查看 AWR 报告中 SQL ordered by CPU TIME 和 SQL ordered by Gets，发现 CPU 耗用时间较多和逻辑读较高的 SQL 均为 SELECT count(1)　FROM SYS_USER_STAFF where extend1 > '-1'（如图 5、图六所示），初步确定引发 CPU 利用率偏高的原因是该 SQL 语句逻辑读较高。



SQL ordered by CPU Time

- Resources reported for PL/SQL code includes the resources used by all SQL statements called by the code.
- %Total - CPU Time as a percentage of Total DB CPU
- %CPU - CPU Time as a percentage of Elapsed Time
- %IO - User I/O Time as a percentage of Elapsed Time
- Captured SQL account for 71.4% of Total CPU Time (s): 262
- Captured PL/SQL account for 0.3% of Total CPU Time (s): 262

CPU Time (s)	Executions	CPU per Exec (s)	%Total	Elapsed Time (s)	%CPU	%IO	SQL Id	SQL Module	SQL Text
4.30	5,031	0.00	1.64	8.55	50.34	0.00	0akkp74bzn22v	JDBC Thin Client	SELECT a.refviewid, a.viewid, ...
4.04	5,035	0.00	1.54	7.03	57.40	0.00	7y15hw9w5c3cd	JDBC Thin Client	SELECT a.refviewid, a.viewid, ...
1.72	5,031	0.00	0.65	3.66	46.84	0.00	7qb1btjypannq	JDBC Thin Client	SELECT distinct d.menuid, d.me...
1.34	5,031	0.00	0.51	3.20	42.06	0.00	cdmkd05udxy34	JDBC Thin Client	select a.templateid, d.transpa...
1.03	5,031	0.00	0.39	2.41	42.80	0.00	as5cq2nabr7af	JDBC Thin Client	SELECT COUNT(*) AS rowcou FROM...
0.83	5,033	0.00	0.32	1.90	43.49	0.00	60wz0ycw9yvjc	JDBC Thin Client	select distinct a.topicid, a.t...
0.80	1	0.80	0.31	1.32	60.88	1.02	bc7gjv3ppdtbz	sqlplus@VL-Dev-8554 (TNS V1-V3)	BEGIN dbms_workload_repository...
0.66	5,035	0.00	0.25	1.52	43.24	0.00	anm21158tgp5h	JDBC Thin Client	SELECT COUNT(*) AS rowcou FROM...
0.65	25,173	0.00	0.25	3.00	21.70	0.01	btusv0q00b3nh	JDBC Thin Client	INSERT INTO SYS_LOG (logid, or...

图 5 SQL ordered by CPU Time

SQL ordered by Gets

- Resources reported for PL/SQL code includes the resources used by all SQL statements called by the code.
- %Total - Buffer Gets as a percentage of Total Buffer Gets
- %CPU - CPU Time as a percentage of Elapsed Time
- %IO - User I/O Time as a percentage of Elapsed Time
- Total Buffer Gets: 9,574,866
- Captured SQL account for 99.0% of Total

Buffer Gets	Executions	Gets per Exec	%Total	Elapsed Time (s)	%CPU	%IO	SQL Id	SQL Module	SQL Text
7,356,135	5,039	1,459.84	76.83	371.58	45.2	0	1fzg4bazvqjhf	JDBC Thin Client	SELECT count(1) FROM SYS_USER_...
183,271	25,173	7.28	1.91	3.00	21.7	0	btusv0g00b3nh	JDBC Thin Client	INSERT INTO SYS_LOG (logid, or...
171,173	10,068	17.00	1.79	1.90	33.7	0	40uc9r6m47quk	JDBC Thin Client	SELECT b.TEMPLATEID, a.tabid, ...
151,050	5,035	30.00	1.58	7.03	57.4	0	7y15hw9w5c3cd	JDBC Thin Client	SELECT a.refviewid, a.viewid, ...
150,930	5,031	30.00	1.58	8.55	50.3	0	0akkp74bzn22v	JDBC Thin Client	SELECT a.refviewid, a.viewid, ...
130,858	10,066	13.00	1.37	1.03	18.2	0	2vfx8trdv6w3g	JDBC Thin Client	SELECT t.*, (select SYS_ORG.OR...
115,702	10,064	11.50	1.21	1.02	28.7	0	2p99tvxwvbc3g	JDBC Thin Client	SELECT a.fldid, a.fldname, b.v...
110,726	5,033	22.00	1.16	1.90	43.5	0	60wz0ycw9yvjc	JDBC Thin Client	select distinct a.topicid, a.t...
110,682	5,031	22.00	1.16	2.41	42.8	0	as5cg2nabr7af	JDBC Thin Client	SELECT COUNT(*) AS rowcou FROM...
105,729	35,242	3.00	1.10	2.25	21.5	0	g5vmq0vvqskrf	JDBC Thin Client	SELECT transcode, transfielden...

图 6 SQL ordered by CPU Time

开启 aototrace（set autotrace on），执行该 sql 语句，分析执行计划和统计信息发现，该 sql 执行时逻辑读偏高，通过将 sql 语句改写为 SELECT count(1)　FROM SYS_USER_STAFF where extend1 ='1'，sql 语句执行时逻辑读大大降低，同时相关交易并发时 CPU 利用率由 85%左右下降为 37.2%。

五、结束语

性能调优是一个涉及网络、硬件、软件和应用程序等多层次、综合性的优化过程，随着应用性能管理已经从最开始以网络监控为核心，发展到以基础组件为核心，到当前的端到端应用性能管理为核心，基于 SaaS 交付的应用性能管理产品将成为主流，未来性能调优过程也将由人工分析逐步向自动化、智能化方向迈进。

浏览器网络捕获器在 Python 网络编程中的辅助作用

◆ 作者：唐步天 谢波艳

Python 的网络编程的最典型应用就是搜索引擎的网络爬虫了。 对于个人而言，更多的是利用 python 编程操作浏览器的 HTTP 请求 ，这其中有如下三种主要场景:

1. 请求页面场景: 这种场景通常是直接对 URL 页面的请求，通常不包含用户私有化的交易数据，通常使用浏览器请求的 GET 方式 。

2. 数据提交场景: 这种场景通常是浏览器展示的页面上的用户输入要素提交后台的过程，这种提交可以有两种方式 ，简单的方式是 GET 方式，只需要将提交页面的目标 URL 附加上输入要素即可， 复杂的方式是 POST 方式，需要在 HTTP 通信体中包含提交的用户输入要素。这其中用户数据格式又有 JSON 等格式。

3. 模拟浏览器用户输入和点击等场景: 这种场景的目的是数据的输入，目标数据输入后，实际应用的结果是转化成了上述的数据提交场景。

所以 ，场景 3 实现了用户视觉化的提交，而场景 2 可以实现用户非视觉化的提交。如果在在场景 1 和场景 2 之间实现一个数据程序写入的过程。则 3 个场景的过程就完全转化为 2 个场景的过程。

一个典型的历经三个场景的过程是用户登陆的过程:

可以看到，在登录以后，写邮件或处理邮件的过程，又是场景 1 到场景 3 的重复，但与登录不同的是，写邮件或处理邮件的过程会包含数据处理的过程。 要实现数据自动处理过程，就要实现页面接入数据的读取与分析，浏览器的网络捕获器可以辅助实现该功能，有助于 python 编程人员进行数据读取与分析，而无需再安装专门的网络截包软件、代理软件或专用性能测试之软件，从而较方便快捷地实现 2 场景的编程过程和调试过程。

Python 对场景 1 的编程过程示范代码如下:

```
import urllib.request
import urllib.request
booksurl="http://www.xxxx.com/bookshelf"
response = urllib.request.urlopen(booksurl)
html = response.read()
#print(html)
htmlstr=html.decode()
```

Python 对场景 2 的编程过程示范代码如下:

1.GET 方式:

```
import urllib.request
booksurl="http://www.xxxx.com/bookshelfask.jsp?id=37"
response = urllib.request.urlopen(booksurl)
```

```
html = response.read()
#print(html)
htmlstr=html.decode()
```

2. POST 方式示例 1:

```
import json
values ={'bookAttribute': 'false', 'readState': 'false'}
jdata = json.dumps(values)
jdatab=jdata.encode(encoding='UTF8')
headers={"Content-Type": "application/json;charset=UTF-8","Connection":"keep-alive","User-Agent":"Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko","Accept":"application/json, text/plain, */*","Referer":"http://www.xxxx.com/stu /index.html","Accept-Language":"zh-CN","Accept-Language":"zh-CN","Accept-Encoding":"gzip, deflate","Host":"www.xxxx.com","Cache-Control":"no-cache"}
    usingurl= "http://www.xxxx.com/ 72200/ XXXX/bookinshelf"
      req = urllib.request.Request(usingurl, jdatab,headers)
      response=None
      response = urllib.request.urlopen(req)
      if (response==None):
        print("fail: "+book)
```

3.POST 方式示例 2:

```
import json
headers={"Content-Type": "application/json;charset=UTF-8","Connection":"keep-alive","User-Agent":"Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Trident/5.0)","Accept":"application/json, text/plain, */*","Referer":"http://www.xxxx.com/student/protype/index.html#/readingCenter/multipleTesting","Accept-Language":"zh-CN","Accept-Language":"zh-CN","Host":"www.xxxx.com","Origin":"http://www.xxxx.com","Cache-Control":"no-cache"}
#header should not have Accept-Encoding: gzip, deflate, or returned str will be error in decode() function.

submitstr='{"bookId":XXXX,"studentId":"72200"}'
values =submitstr
#here,values is string type,no need json.dumps(),call encode() to be bytestr
      jdatab=values.encode()
      req = urllib.request.Request("http://www.xxxx.com/veripaper", jdatab,headers)
```

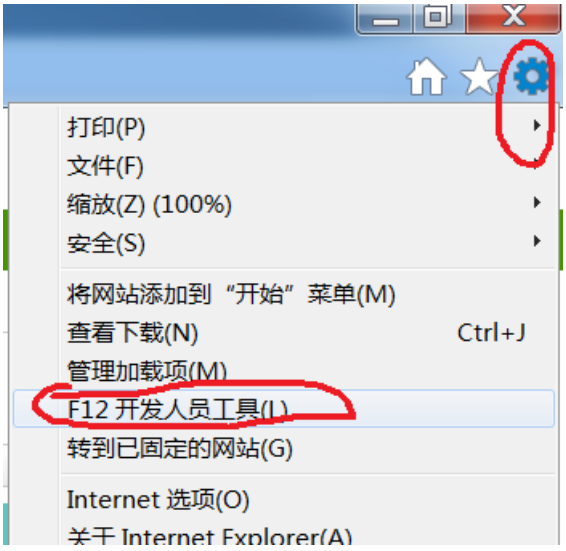
```
response=None
response = urllib.request.urlopen(req)
if (response==None):
    print("fail: "+book)
else:
    print("test: "+book)
html = response.read()
#print(html)
htmlstr=html.decode()
```

在上述代码中看到的 HTTP 头，就可以用浏览器的网络捕获器工具得到，在调试过程中，代码调试不正确时，得不到 HTTP 的正确返回，此时同样可以通过浏览器提交请求，用浏览器的网络捕获器工具截 HTTP 通信内容，分析问题。

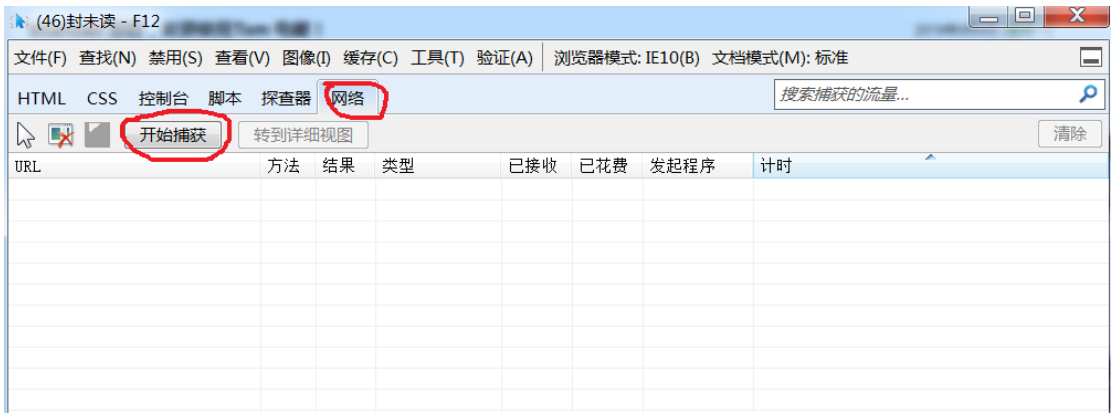
下面是 IE 浏览器和 google 浏览器的网络捕获器工具的用法。

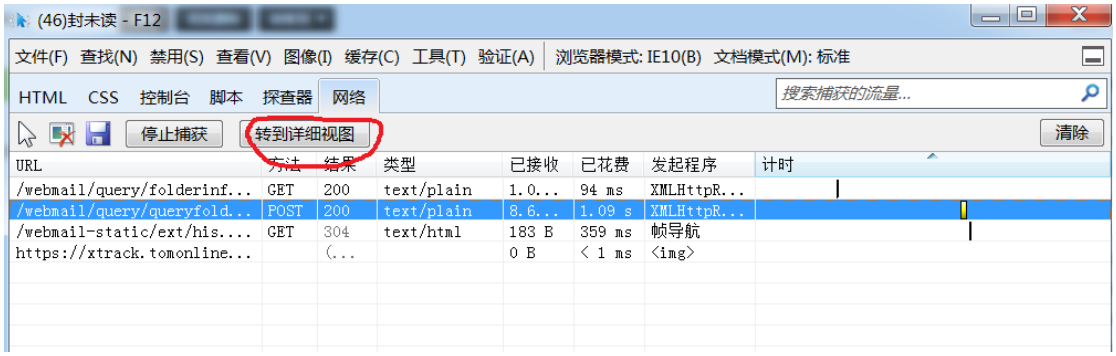
1、IE 浏览器网络捕获器:

通过 IE 菜单

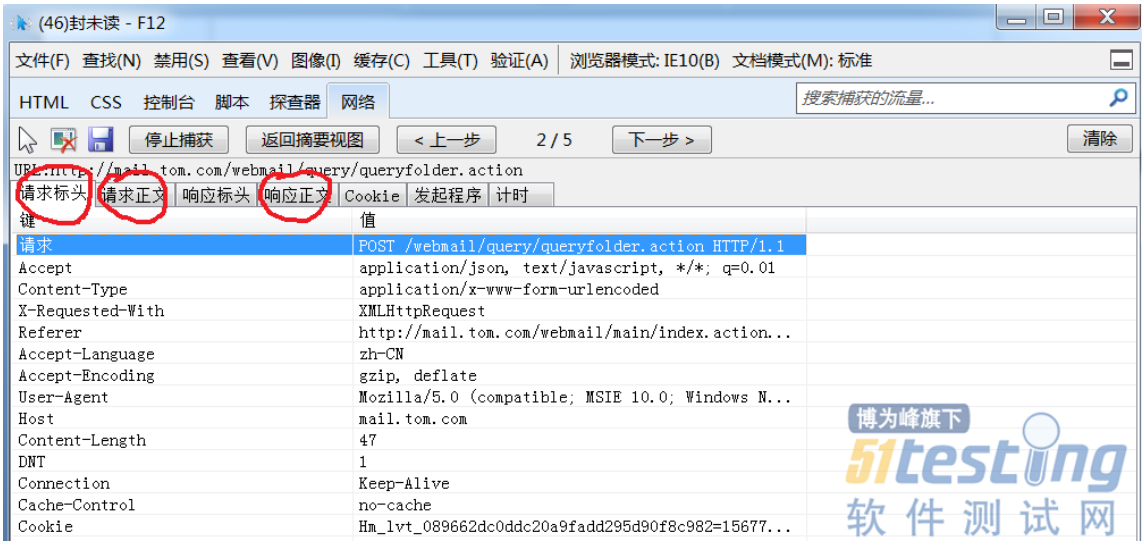


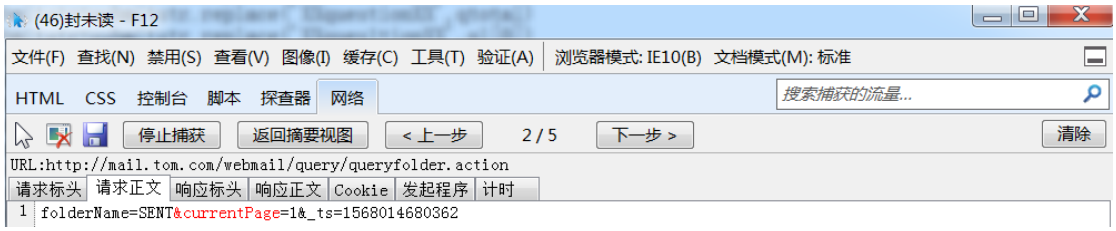
点“网络”页，的“开始捕获”按钮

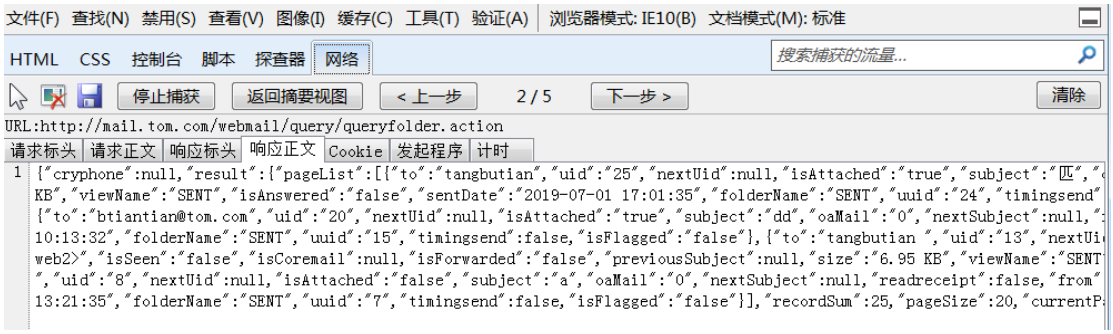




主要看“请求标头”“请求正文”和“响应正文”页，这些信息可以拷到 Python 程序中使用。





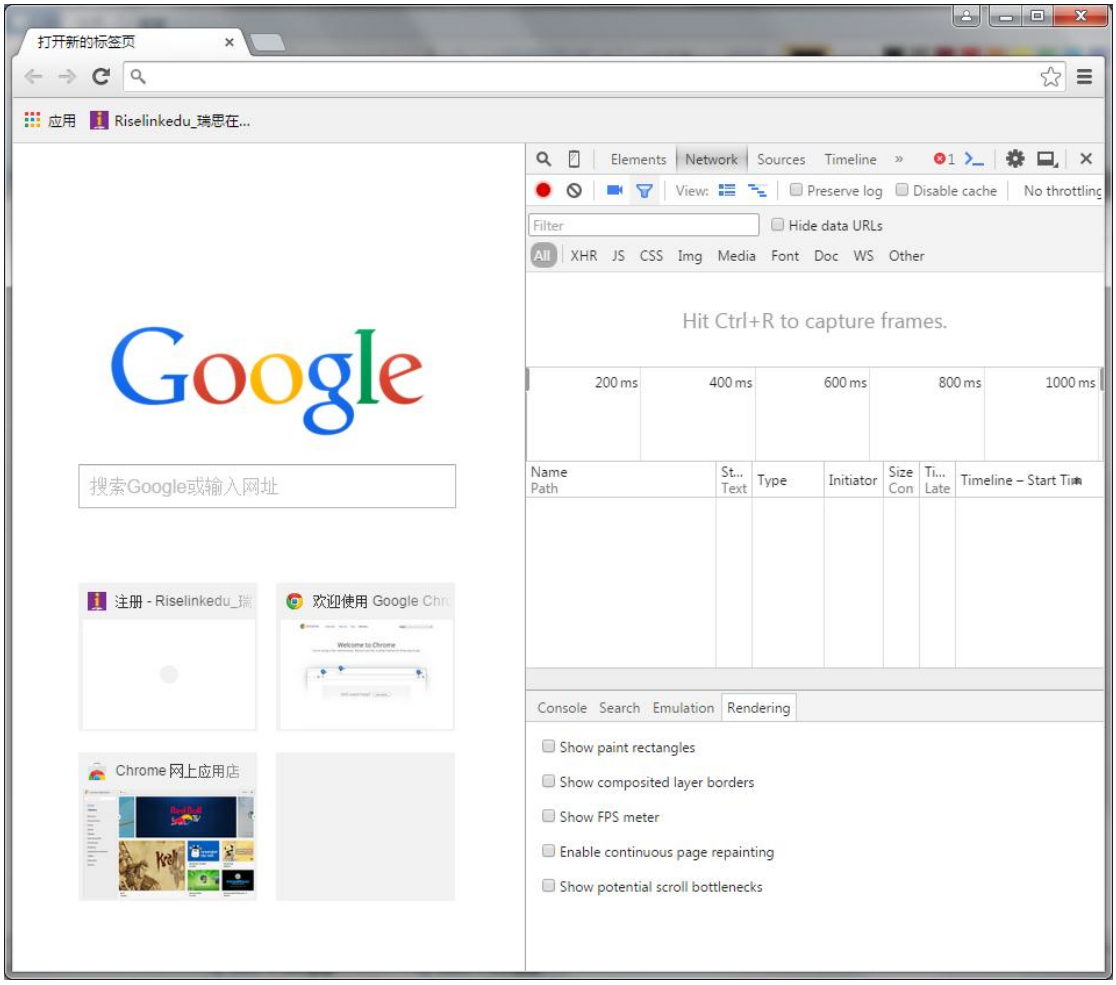


最后，可以用“停止捕获”按钮结束捕获过程。

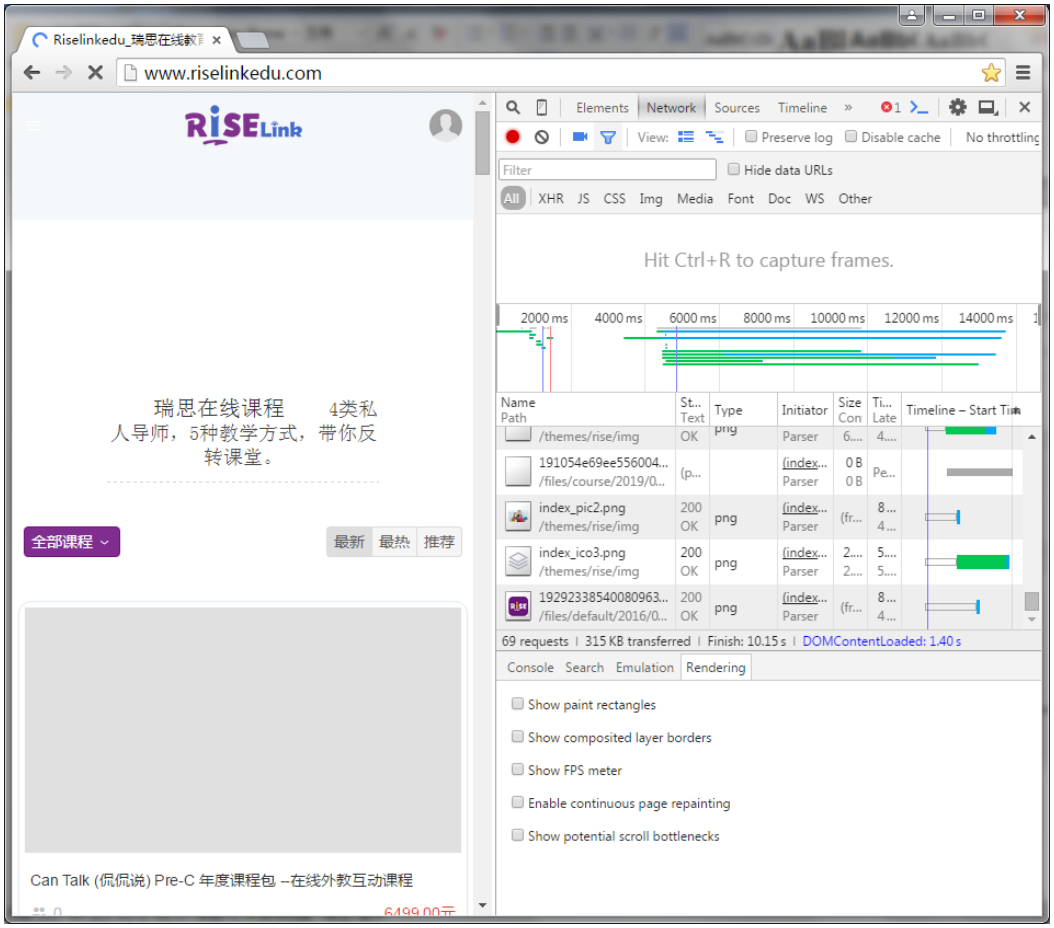
2. Google 浏览器网络捕获器:

通过 Google 浏览器的菜单

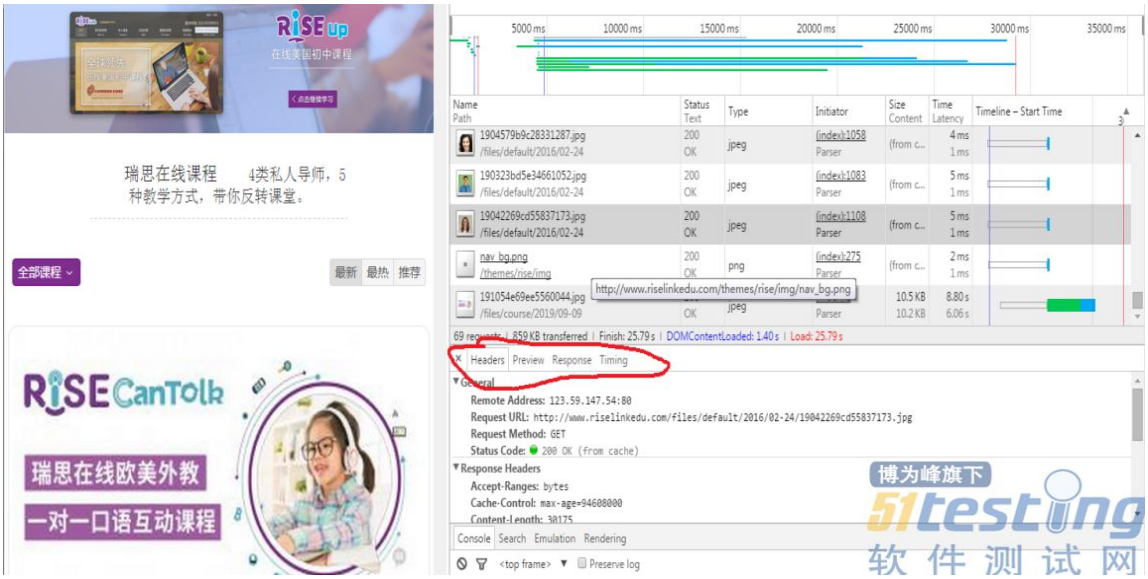
这时，看到浏览器分栏了，在右栏就是捕获器



在浏览器执行网页操作，右边捕获器就可捕获到网络访问。



选择其中一个网络访问，下图就出现该 HTTP 访问的头（Headers）及响应（Response）等信息。这些信息可以拷到 Python 程序中使用。



最后，可以用浏览器分栏的右栏上面的红色圆点按钮结束捕获过程。

小结： Google 和 IE 浏览器的网络捕获器都可以实现 HTTP 捕获功能，捕获的结果可以在 python 编程中使用，比较方便，根据笔者经验，在 Web 应用兼用性不好，IE 访问截获有障碍时，用 Google 浏览器实现 HTTP 捕获会有很好效果。

解析 PCAP 文件创建自定义网络报文

◆ 作者：王 练

摘要：在实际的网络测试中，为了应对多种的测试场景，经常要对网络报文进行编辑。目前很多软件都提供了报文的修改功能，如科来，TCP Replay。对于少量的报文修改，提供了非常便捷的操作界面，很容易完成模拟报文的构造。然而当需要完成数以万记的报文修改时，手工编辑的方式就显得效率太低了。

本文从通用的 PCAP 文件格式谈起，首先详细介绍 PCAP 文件的格式，之后通过一个测试中的示例，说明如何批量完成 PCAP 文件的修改，进而得到自定义的网络报文。虽然是针对特定场景、特定数据字段的报文修改解决方案，但提供的思路对于任何使用 PCAP 的测试情况都是适用的。

一、PCAP 文件格式

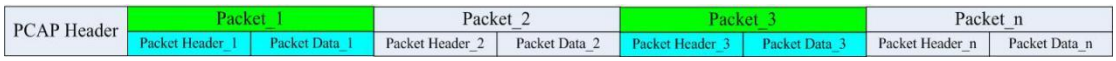
PCAP 文件是常用的数据报存储格式，内部以固定的格式存储网络报文数据。通过带有二进制内容查看功能的文本编辑器（如 UltraEdit）可以查看其中的内容。下图是一个 SNMP Trap 报文的内容查看情况。

```
00000000h: D4 C3 B2 A1 02 00 04 00 00 00 00 00 00 00 00 00 ; 悦病............
00000010h: 00 00 04 00 01 00 00 00 36 13 28 5D 00 00 00 00 ; ........6.(]....
00000020h: 0A 01 00 00 0A 01 00 00 00 0C 29 02 DB 1A 1C 1B ; ..........).?..
00000030h: 0D 2B D7 77 08 00 45 00 00 FC F1 23 00 00 40 11 ; .+譴..E..  #..@.
00000040h: C1 A9 DE 03 00 01 AC 10 3D 0F 00 BC 00 A2 00 E8 ; 俩?..?=..???
00000050h: 6E AD 30 81 DD 02 01 01 04 0E 74 72 61 70 20 63 ; n?佪.....trap c
00000060h: 6F 6D 6D 75 6E 69 74 79 A7 81 C7 02 02 01 76 02 ; ommunity  ?..v.
00000070h: 01 00 02 01 00 30 81 BA 30 10 06 08 2B 06 01 02 ; .....0侔0...+...
00000080h: 01 01 03 00 43 04 07 94 6F F4 30 17 06 0A 2B 06 ; ....C..攥?...+.
00000090h: 01 06 03 01 01 04 01 00 06 09 2B 06 01 06 03 01 ; ..........+.....
000000a0h: 01 05 03 30 11 06 09 2B 06 01 02 01 02 02 01 01 ; ...0...+........
000000b0h: 02 04 21 FD 04 01 30 0E 06 09 2B 06 01 02 01 02 ; ..!?.0...+.....
000000c0h: 02 01 07 02 01 01 30 0E 06 09 2B 06 01 02 01 02 ; ......0...+.....
000000d0h: 02 01 08 02 01 02 30 17 06 0A 2B 06 01 06 03 01 ; ......0...+.....
000000e0h: 01 04 03 00 06 09 2B 06 01 04 01 C5 36 17 02 30 ; ......+....?..0
000000f0h: 15 06 0D 2B 06 01 04 01 C5 36 0F 01 03 01 19 00 ; ...+....?......
00000100h: 02 04 C0 A8 14 0F 30 16 06 0E 2B 06 01 04 01 C5 ; ..括..0...+....?
00000110h: 36 0F 01 01 01 01 09 00 02 04 5A 4C 3F A0 30 12 ; 6.........ZL??.
00000120h: 06 0C 2B 06 01 04 01 C5 36 0F 01 03 01 46 02 02 ; ..+....?....F..
00000130h: 01 76                                           ; .v
```

PCAP 之所以可以作为一个通用的格式，是因为使用非常广泛（当然也可以反过来说）。WireShark 抓包的结果可以另存为 PCAP 格式（默认的是 pcapng）；科来也可以导入 PCAP 格式的报文；同时 tcpdump 的存储格式也是 PCAP。当然，其他格式的报文也都可以通过 WireShark 进行另存后得到 PCAP 格式。这就意味着，几乎所有的报文文件，都可以转化为 PCAP。所以，当通过 PCAP 获取了自定义的网络报文后，可以适用所有的环境。

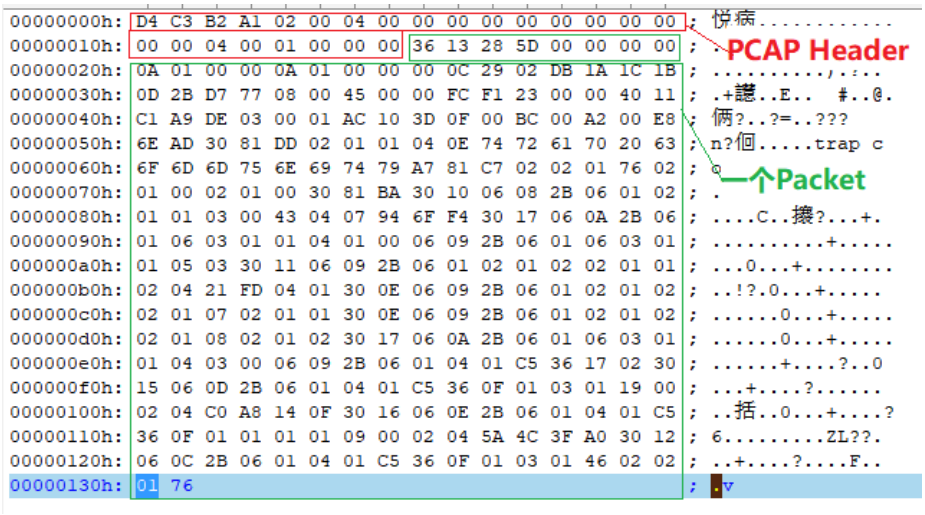
虽然以二进制方式看到的 PCAP 文件让人感觉摸不到头脑，而实际上 PCAP 的文件格式是非常清晰的。

首先，从整体来看，PCAP 是以一个文件头和若干个报文数据组成的文件。如下图所示。



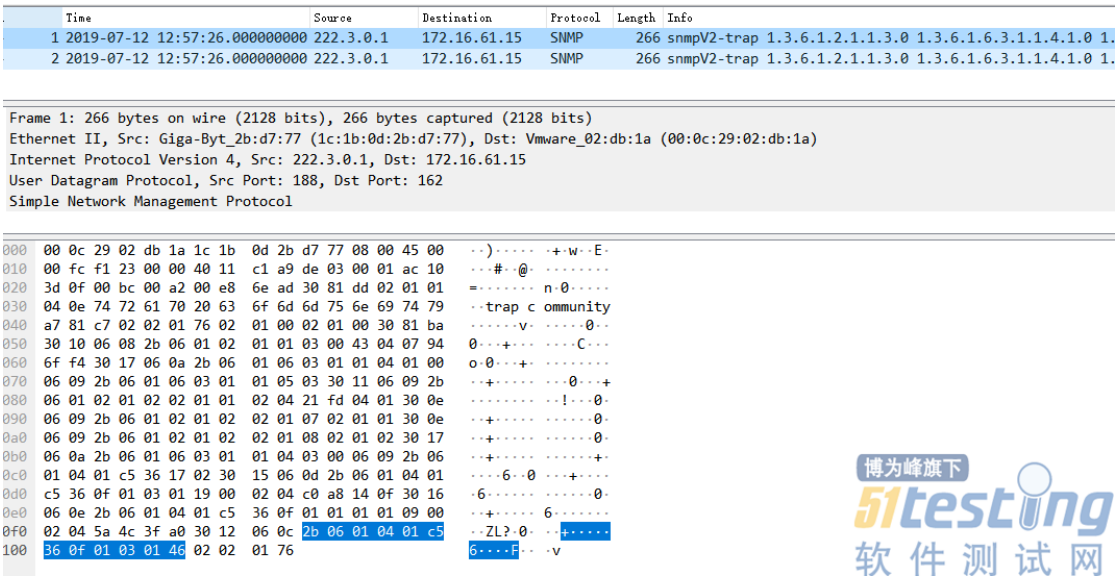
文件头我们标记为 PCAP Header，一个 PCAP 文件存在一个 PCAP Header 格式。

之后是若干的报文数据，我们标记为 Packet，更细的一个 Packet 中存在包头（Packet Header）和包数据（Packet Data）。一个 PCAP 文件存在至少一个包格式。我可以理解为若干个包组成一个包的链表，存储在 PCAP 文件中。事实上， 我们后续的文件解析也正是将其放在一个链表中。



对于上面的例子，我们可以进行解析。红色的为一个 PCAP Header，绿色的为一个 Packet 数据。对于多个 Packet 的情况，就是后面的绿色内容进行循环。我们是否可以直接对 Packet 的数据进行复制粘贴呢？答案是肯定的。对于上面的文件，我们从 36 13 开

始复制，到 01 76，在文件末尾粘贴后，就得到了具有两个一模一样的 SNMP Trap 的报文文件。



接下来，对每个格式进行说明。其实从上面的红色标记已经可以看到了，PCAP Header 是一个固定的格式，而 Packet 会根据不同的数据包有所区别。PCAP Header 共计 24 个字节，具体格式如下。

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
00000000h:	Magic				Major Version		Minor Version		Timezone				Sigflags			
	D4 C3 B2 A1				02 00		04 00		00 00 00 00				00 00 00 00			
00000010h:	Snap Length				Link Type											
	00 00 04 00				01 00 00 00											

字段	含义	长度	示例内容
Magic	文件识别头，pcap 固定为：0xA1B2C3D4	4 字节	D4 C3 B2 A1
Major Version	主版本号	2 字节	02 00
Minor Version	次要版本号	2 字节	04 00
Timezone	当地的标准时间	4 字节	00 00 00 00，表示为 GMT 时间
Sigflags	时间戳的精度	4 字节	00 00 00 00
Snap Length	最大的存储长度	4 字节	00 00 04 00
Link Type	链路类型	4 字节	01 00 00 00

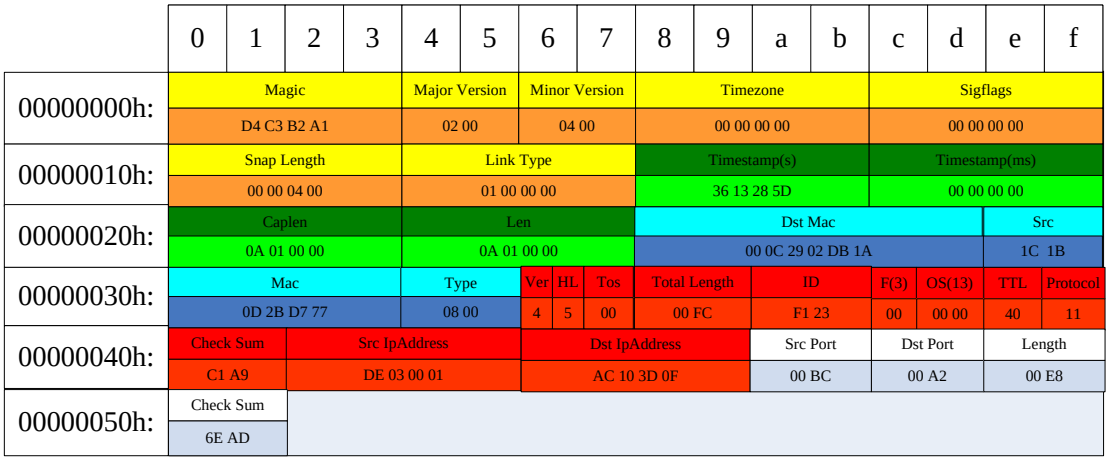
接下来，看 Packet 的格式，先看 Packet Header（如下绿色系内容）。

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
00000000h:	Magic				Major Version		Minor Version		Timezone				Sigflags			
	D4 C3 B2 A1				02 00		04 00		00 00 00 00				00 00 00 00			
00000010h:	Snap Length				Link Type				Timestamp(s)				Timestamp(ms)			
	00 00 04 00				01 00 00 00				36 13 28 5D				00 00 00 00			
00000020h:	Caplen				Len											
	0A 01 00 00				0A 01 00 00											

字段	含义	长度	示例内容
Timestamp(s)	时间戳高位，精确到 seconds，这是 Unix 时间戳。捕获数据包的时间一般是根据这个值	4 字节	36 13 28 5D，转换为 2019-07-12 12:57:26
Timestamp(ms)	时间戳低位，能够精确到 microseconds	4 字节	00 00 00 00
Caplen	当前数据区的长度，即抓取到的数据帧长度，由此可以得到下一个数据帧的位置	4 字节	0A 01 00 00，表示为 266 字节。
Len	离线数据长度，网络中实际数据帧的长度，一般不大于 Caplen，多数情况下和 Caplen 值一样	4 字节	0A 01 00 00

实际解析时需要注意高位优先（big-endian）和低位优先（little-endian）的问题，PACP 文件的存储是低位优先，也就是所谓的小端模式：高字节数据存放在高地址处。

接下来的 Packet Data 数据，不同的包有不同的格式，最常见的是通过以太网封装的 TCP/IP 数据。上述的 SNMP Trap 就是封装在以太网帧中的 IP 数据包，以 UDP 作为传输层，具体解析如下。



各字段详细说明如下：

内容	字段	含义	长度	示例内容
以太帧头（蓝色系）	Dst Mac	目的 MAC 地址	6 字节	00 0C 29 02 DB 1A
	Src Mac	源 MAC 地址	6 字节	1C 1B 0D 2B D7 77
	Type	数据帧类型	2 字节	08 00，就是众所周知的 IP 报文。
IP 报头（红色系）	Version	IP 的版本号。	4 比特	4，2 进制为 0100，表示 Ipv4。
	Head Length	IP 首部的总长度，其中包括选项字段（如果有）。	4 比特	5，该字段的表示的长度单位为 4 字节，所以 5 的含义是 20 字节。
	Tos	服务量优先级。	1 字节	00
	Total Length	整个 IP 数据报的长度。	2 字节	00 FC
	ID	唯一的标识主机发送的每一份数据报	2 字节	F1 23
	Flag	每一位都有特定的含义，用来分片和重组。	3 比特	00
	Offset	每一个分片的数据字段偏移原始数据报开始处的位置。	13 比特	00
	TTL	该 IP 数据包可以经过的路由器的最大数量	1 字节	40
	Protocol Type	在 IP 上层承载的是什么协议	1 字节	11，表示为 UDP 协议。

	Check Sum	首部校验和字段	2 字节	C1 A9
	Src IpAddress	源地址。	4 字节	DE 03 00 01，点分制为 222.2.0.1。
	Dst IpAddress	目的地址	4 字节	AC 10 3D 0F，点分制为 172.16.61.15。
UDP 报头（白色系）	Src Port	源端口	2 字节	00 BC，即 188
	Dst Port	目的端口	2 字节	00 A2，即 162
	Length	报文长度	2 字节	00 E1，长度为 232 字节
	Check Sum	校验和	2 字节	6E AD

后续的数据部分就是对 SNNP Trap 的净荷解析了，根据特定的格式即可进行。

通过上述的分析我们可以看到，PCAP 的格式非常规整和清晰。通过 PCAP Header 定位文件的相关信息。之后重复进行 Packet 数据的迭代，每个 Packet 根据本身的包结构进行存储。如 SNMP 数据，遵循 SNMP->UDP->IP->Ethernet 的封装结构。每个 Packet 互相独立，互不干扰。

三、测试需求说明

在实际的测试中，会遇到各种各样的模拟报文需求。目前项目中需要的是针对服务器的 SNMP Trap 处理能力进行测试，即每秒的 Trap 处理吞吐量。通过实际需求的收集和软件架构的分析，设计指标为万级别的吞吐量。

实际的情况无法产生如此大量的 Trap 信息，只能通过模拟的方式进行测试。之前的测试是通过实际设备抓包得到原始的报文，再修改必要的字段进行。目前万级别的报文手工修改效率太低，通过上述对 PCAP 文件的分析，可以尝试编写程序，批量生成模拟报文。

四、解析 PCAP 文件

有了前面对于 PCAP 文件的格式分析，解析 PCAP 报文已经非常容易，剩下的就是编程实现，以 Java 实现为例。定义 PCAP 文件格式如下。

```
public class PcapFileReader{

    private PcapHeader fileHeader;
    private List<PcapPacketData> packetDataList;

    public PcapFileReader(String fileName) throws IOException{
        FileInputStream fis = null;
        fis = new FileInputStream(fileName);

        byte[] fileHeaderBuffer = new byte[24];
        int fileHeaderLength = fis.read(fileHeaderBuffer);
        this.fileHeader = parseFileHeader(fileHeaderBuffer);

        packetDataList = new ArrayList<PcapPacketData>();
        while(fis.available()!=0){
            PcapPacketData packetData = new PcapPacketData(fis);
            packetDataList.add(packetData);
        }
    }
```

实际用 PcapHeader 存储 PCAP 文件的 Header 部分，Packet 数据通过 List<PcapPacketData>存储。解析后，一个 PCAP 文件就对应于内存中的一个 PcapFileReader 对象。

PcapHeader 的主要定义如下。

```
public class PcapHeader implements Serializable {

    private int magic;                  // 标识位，这个标识位的值是16进制的0xa1b2c3d4（4个字节）
    private short magorVersion;      // 主版本号（2个字节）
    private short minorVersion;      // 副版本号（2个字节）
    private int timezone;               // 区域时间（4个字节）
    private int sigflags;               // 精确时间戳（4个字节）
    private int snaplen;                // 数据包最大长度（4个字节）
    private int linktype;               // 链路层类型（4个字节）
```

可以看到，就是对上述格式的代码实现。将 PCAP 文件的固定字节（前 24 字节）数据读入到缓存后，通过固定格式的解析实例化 PcapHeader 的各个字段。以 Magic 字段为例，将前 4 字节内容转化为 int 类型，放入到 PcapHeader 的 magic 字段，代码如下。

```
public PcapHeader parseFileHeader(byte[] file_header) throws IOException {
    PcapHeader fileHeader = new PcapHeader();
    byte[] buff_4 = new byte[4];     // 4字节的数组
    byte[] buff_2 = new byte[2];     // 2字节的数组

    int offset = 0;
    for (int i = 0; i < 4; i ++) {
        buff_4[i] = file_header[i + offset];
    }
    offset += 4;
    int magic = DataUtils.byteArrayToInt(buff_4);
    fileHeader.setMagic(magic);
```

其他字段如法炮制。

对于 Packet 数据，前面的代码中对列表进行了处理。读取 24 字节的 PCAP Header 之后，持续的读入数据到内存中，一个一个的处理 Packet 数据，设置字段后加入列表中。对于每个 Packet 的处理，说明如下。

```
public class PcapPacketData {
    private PcapDataHeader dataHeader;  //报文头
    private PcapEthernetFrame ethernetFrame;//以太帧头
    private IPHeader ipHeader;//IP包头
    private byte[] payloadBuffer;//TCP或者UDP头和数据
    int bufferLength = 0;   //Pcap文件的一个包数据大小

    public PcapPacketData(FileInputStream fis) throws IOException{
        byte[] dataHeaderBuffer = new byte[16];
        int dataHeaderLength = fis.read(dataHeaderBuffer);
        this.dataHeader = parseDataHeader(dataHeaderBuffer);

        byte[] ethernetFrameBuffer = new byte[14];
        int ethernetFrameLength = fis.read(ethernetFrameBuffer);
        this.ethernetFrame = parsePcapEthernetFrame(ethernetFrameBuffer);

        byte[] ipHeaderBuffer = new byte[20];
        int ipHeaderLength = fis.read(ipHeaderBuffer);
        this.ipHeader = parseIpHeader(ipHeaderBuffer);

        this.payloadBuffer = new byte[this.ipHeader.getTotalLen()-DataUtils.ipHeaderLength(this.ipHeader.getVarHLen())];
        int payloadsLength = fis.read(payloadBuffer);

        bufferLength = dataHeaderBuffer.length+ethernetFrameBuffer.length+ipHeaderBuffer.length+payloadBuffer.length;

    }
```

定义 Packet 数据格式如上，包括 Packet Header：PcapDataHeader，以太帧头：PcapEthernetFrame，IP 报头：IPHeader，以及 IP 数据区内容（包括 TCP 或 UDP 报头和数据）。由于修改的字段只涉及到 IP 报头，所以只解析到 IP 报头即可。如果有其他应用需要，可以根据情况对传输层以及应用层数据进行解析。

以 Packet Header 数据为例，读取数据到缓冲区，之后通过 parseDataHeader 函数进行该格式的赋值。具体示例如下。

```
public PcapDataHeader parseDataHeader(byte[] data_header){
    byte[] buff_4 = new byte[4];
    PcapDataHeader dataHeader = new PcapDataHeader();
    int offset = 0;
    for (int i = 0; i < 4; i ++) {
        buff_4[i] = data_header[i + offset];
    }
    offset += 4;
    int timeS = DataUtils.byteArrayToInt(buff_4);
    dataHeader.setTimeS(timeS);
```


将缓冲区的前 4 个字节获取，转化为 int 后放入 PcapDataHeader 的 times 字段。其他字段如法炮制。PcapDataHeader 的定义如下。

```
public class PcapDataHeader {

    private int timeS;  //时间戳（秒）：记录数据包抓获的时间，记录方式是从格林尼治时间的1970年1月1日00:00:00到抓包时经过的秒数（4个字节）
    private int timeMs;    //时间戳（微秒）：抓取数据包时的微秒值（4个字节）
    private int caplen;//数据包长度：标识所抓获的数据包保存在 pcap 文件中的实际长度，以字节为单位（4个字节）
    private int len;  // 数据包实际长度：所抓获的数据包的真实长度（4个字节）   如果文件中保存不是完整的数据包，那么这个值可能要比前面的数据包长度的值大。
```

以太帧头：PcapEthernetFrame，IP 报头：IPHeader 也是类似的方式。对于 IP 报头中的变长字段，可以通过 IP 报头的标志位进行判断，根据不同情况进行处理。本例中报文是固定的一条，所以使用固定大小的缓冲区进行解析。

在前面的示例中，我们将原始的一条报文 SNMP Trap 报文的数据复制为了两条，通过当前的解析，可以得到更新的内容。通过如下代码可以看到解析的文件对比。

```
//打开原始报文文件
PcapFileReader pcapFile1 = new PcapFileReader("1_LinkDown_Appear.pcap");
System.out.println(pcapFile1.toString()+"\n\n");
//打开复制的报文文件
PcapFileReader pcapFile2 = new PcapFileReader("1_LinkDown_Appear-bak.pcap");
System.out.println(pcapFile2.toString());
```

运行结果如下。

```
PcapFileHeader [magic=d4c3b2a1, magorVersion=512, minorVersion=1024, timezone=0, sigflags=0, snaplen=400, linktype=1000000]
[PcapDataHeader [timeS=3613285d, timeMs=0, caplen=266, len=266]
PcapDataFrame [dstMax=000c2902db1a,srcMac=1c1b0d2bd777,frameType=2048]
IPHeader [varHLen=69@version=100-headerLen=20, tos=0, totalLen=252, id=-3805, flagSegment=0, ttl=64, protocol=17, checkSum=-15959, srcIP=222.3.0.1, dstIP=172.16.61.15]
Payload(TCP/UDP+Data) [Len=232,
Contents=00bc00a200e86ead3081dd020101040e747261702063 6f6d6d756e697479a781c70202017602010002010030 81ba301006082b0601020101030043040794 6ff43017060a2b0601060301010401000 6092


PcapFileHeader [magic=d4c3b2a1, magorVersion=512, minorVersion=1024, timezone=0, sigflags=0, snaplen=400, linktype=1000000]
[PcapDataHeader [timeS=3613285d, timeMs=0, caplen=266, len=266]
PcapDataFrame [dstMax=000c2902db1a,srcMac=1c1b0d2bd777,frameType=2048]
IPHeader [varHLen=69@version=100-headerLen=20, tos=0, totalLen=252, id=-3805, flagSegment=0, ttl=64, protocol=17, checkSum=-15959, srcIP=222.3.0.1, dstIP=172.16.61.15]
Payload(TCP/UDP+Data) [Len=232,
Contents=00bc00a200e86ead3081dd020101040e747261702063 6f6d6d756e697479a781c70202017602010002010030 81ba301006082b0601020101030043040794 6ff43017060a2b0601060301010401000 6092
PcapDataFrame [dstMax=000c2902db1a,srcMac=1c1b0d2bd777,frameType=2048]
IPHeader [varHLen=69@version=100-headerLen=20, tos=0, totalLen=252, id=-3805, flagSegment=0, ttl=64, protocol=17, checkSum=-15959, srcIP=222.3.0.1, dstIP=172.16.61.15]
Payload(TCP/UDP+Data) [Len=232,
Contents=00bc00a200e86ead3081dd020101040e747261702063 6f6d6d756e697479a781c70202017602010002010030 81ba301006082b0601020101030043040794 6ff43017060a2b0601060301010401000 6092
```

五、创建自定义网络报文

有了基础格式的定义和解析，再结合实际业务的测试需求，就可以完成自定义网络报文的创建了。以本项目通过一条 SNMP Trap 创建 1W 条为例进行说明。通过实际的设备上报 SNMP Trap 报文完成抓包，记实际设备的 IP 为：IP_ Physical，需要将抓包数据转换为对模拟的 1W 个网元的 SNMP Trap，记模拟设备的 IP 为：IP_Simu_1 到 IP_Simu_10000。整体流程如下。

连接数据库获取待修改的 IP 列表。

打开原始的 PCAP 文件，实例化为 PcapFileReader。

根据 IP 列表和 PcapFileReader 对象生成最终的网络报文 PCAP 文件。

即如下代码的实现。

```
//连接数据库，获取网元IP地址列表，IP地址作为报文修改的目的地址
DBConnection dbConn = new DBConnection("ip","port","user","password");
List<Map<String, String>> ipList = dbConn.getResNeIpList();
dbConn.closeConnection();
//打开原始报文文件
PcapFileReader appearFileContent = new PcapFileReader("1_LinkDown_Appear.pcap");
//生成新的报文文件
PcapFileReproducer reproducer = new PcapFileReproducer(appearFileContent);
reproducer.produce4MultiSrcIp("MultiSrcIp1.pcap",ipList);
```

下面具体说明每一步的实现。

由于模拟设备存储在数据库中，通过数据获取待修改的 IP 列表，一方面节省了手工编写代码生成 IP 的时间，同时也减少了逻辑出错的可能。通过 Java 获取数据库中的内容有很多现成的代码。本例主要代码如下。

```
public List<Map<String, String>> getResNeIpList() throws SQLException{
    ResultSetMetaData resultMetaData;
    String query = "select ipaddress from [redacted] where ipaddress like '222.%';";
    ResultSet rs = dbStat.executeQuery(query);
    resultMetaData=rs.getMetaData();
    List<Map<String, String>> queryRstList = new ArrayList<Map<String, String>>();
    while(rs.next())
    {
        Map<String, String> queryOneRow = new HashMap<String, String>();
        queryOneRow.put(resultMetaData.getColumnName(1), rs.getString(resultMetaData.getColumnName(1)));
        queryRstList.add(queryOneRow);
    }
    rs.close();
    return queryRstList;
}
```


解析原始的 PCAP 文件。这一步在前面解析 PCAP 文件中已经详细说明了，这里就是对只有一条 SNMP Trap 的报文文件进行解析，存储在一个 PcapFileReader 对象中。

根据 IP 列表和 PcapFileReader 对象生成最终的网络报文 PCAP 文件。这里定义了一个 PcapFileReproducer 对象，通过原始的 PcapFileReader 进行实例化。

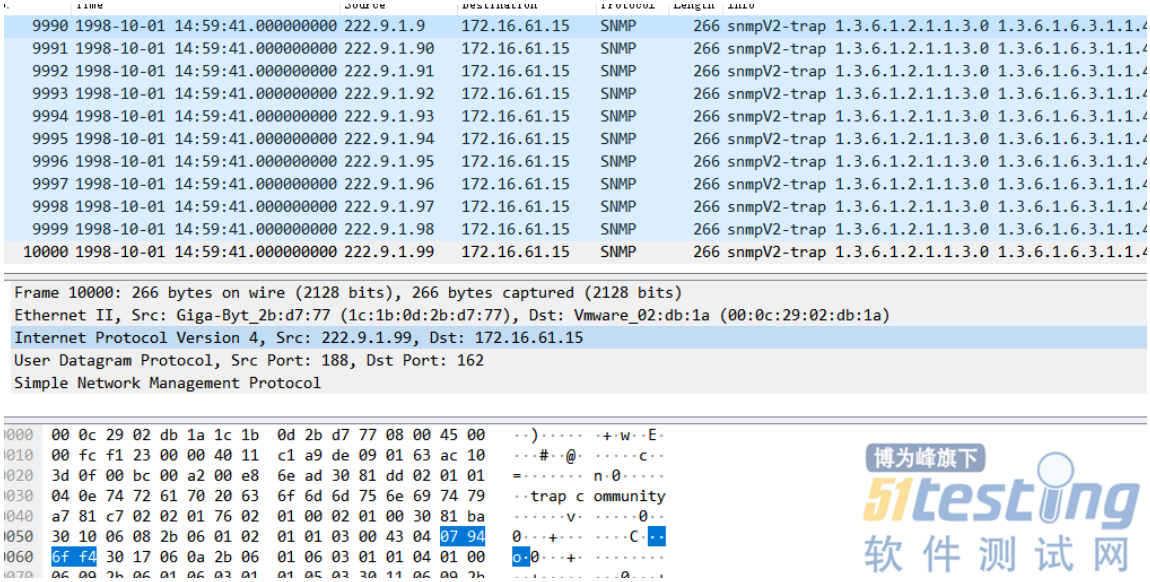
```
public class PcapFileReproducer{

    private PcapFileReader pcapSeed;
    public PcapFileReproducer(PcapFileReader pcapSeed){
        this.pcapSeed = pcapSeed;
    }
```

之后提供各种方法适应不同修改报文需求的场景。本例中实现 produce4MultiSrcIp 函数完成对原始 PCAP 文件修改为列表中 IP 的需求，具体实现如下。

```
public void produce4MultiSrcIp(String descendantFileName, List<Map<String, String>> ipList) throws IOException{
    File file = new File(descendantFileName);
    FileOutputStream fos = new FileOutputStream(file, false);
    fos.write(pcapSeed.getFileHeader().toByteArray());
    for(Map<String, String> ip:ipList){
        for(PcapPacketData packetData:pcapSeed.getPacketDataList()){
            packetData.setSrcIp(ip.get("ipaddress"));
            fos.write(packetData.toByteArray());
        }
    }
    fos.close();
}
```

该方法以目标 PCAP 文件和待修改的 IP 列表为入参，首先打开目标文件作为文件输入流。之后写入与原始 PCAP 文件一致的文件头 PCAP Header。之后循环 IP 列表，针对每个待修改的 IP 地址，将原始 PCAP 文件的每个 Packet 的 IP 报头的源 IP 字段修改为待修改 IP，之后将该 Packet 写入目标文件。最后关闭文件流。

以上述一个 SNMP Trap 报文的文件，修改后得到的 1W 个报文数据通过 WireShark 打开结果如下。



原始报文中的 IP_ Physical 均转化为 IP_Simu_1 到 IP_Simu_10000，得到了 1W 个符合要求的 SNMP Trap 报文。

从这里我们可以看到，PCAP 的格式校验不是特别严格。上述文件头 PCAP Header 的长度字段并没有根据增加的报文进行更新，所以肯定是不对的。但 WireShark 同样可以解析。同时，在各个格式（如 IP 报文头）中均有校验和字段，也没有根据更新情况进行修改。实际测试中可以根据具体的需要进行检查。通过服务端报文解析的日志等线索进行微调。

六、总结

在上述过程中，如果我们简化一下过程可以发现，实际定义 PcapFileReader 等结构化对象，并不是必须的。如果按照特定字节位置的方式也可以实现，只是可读性差，扩展性低。同时，WireShark 还提供的合并报文的工具 mergecap.exe 也非常有用，可以配合完成报文修改的任务。该工具接受合并后的目标 PCAP 文件和待合并的若干个 PCAP 文件，完成报文的合并工作。格式如下:

mergecap.exe -w d:\port1.pcap d:\input_section1\new_1_LinkDown_Appear_*

值得注意的是，该工具对于待合并的文件个数有限制，实际测试发现 1W 个文件无法处理，5000 个可以处理。**这样我们可以得到另外一种解决方案:**

1) 获取原始报文。

2) 针对固定字节进行修改，一次修改生成一个文件，一共生成需要个数的文件。

3) 分批次使用 mergecap 工具得到目标文件。

当然这种方法操作比较繁琐，不如都通过程序的方式简单。

对于多条报文的情况，本例也是适用的。本例中对 IP 数据部分的读取是通过 IP 报头的字段进行的动态获取，所以对每个 Packet 的读取都是完整的。多条报文的修改可以自动适用本例。

对于 IP 报头有其他选项（非 20 字节）的情况，需要进行调整。通过在解析 IP 报头的前面，增加 IP 报头数据的解析，根据选项字段进行不同的解析。如果存在 IP 分片的情况，也能够自动适用。

UDP 的场景由于是无连接的，可以直接进行模拟报文的发送，完成测试。如果是 TCP 的情况，需要将报文模拟的流程嵌入到设备模拟的代码中，完成功能。即在 TCP 连接测试的代码中，增加模拟报文的生成逻辑，将 PCAP 文件的解析、修改作为设备模拟代码的一部分，以完成测试需求。

浅谈测试工程师的产品思维

◆作者：咖啡猫

题记：
“作为一个产品经理，你应该具备的基本素质就是拥有产品思维！”
“作为一个互联网 er，你应该具备产品思维！”
“作为本次线上活动的运营方，你应该具备产品思维！”

“产品思维”这个词似乎随处可以听到或者看到，我相信对广大测试人员来说也并不陌生，今天咖啡猫就和你聊聊测试工程师的产品思维。

一、什么是产品思维

既然是聊产品思维，那我们就先来看看产品思维的定义吧：“从用户心理需求出发，结合自身公司的能力以及市场情况，制定的面向市场的可以使商品价值最大化的方案计划的思维体系。”

哈哈，相信不少同学看了这么一长串定义就头大。

简而言之，产品思维的本质就是洞察人性+利用人性！

举个例子吧：项目组接到一个新任务：优化一款背单词 APP。技术思维是这样的：如何对单词库进行压缩存储？怎么样根据用户偏好决定单词出现的先后顺序？如何让 PC 端和手机 APP 端内容同步？但是产品思维是：怎么让更多的用户参与到背单词活动中来？如何让用户愿意分享自己的单词清单？如何将背单词与线上付费课程联系起来？

通过这个小例子，我们看到了技术思维和产品思维之间明显的差异：技术思维更加关注产品的功能价值、性能指标等；产品思维则更关注用户痛点和用户价值等。

二、测试工程师为什么需要产品思维

都说互联网创业者要过三大关：产品关、市场关和管理关。而这三关中最最根本的还是产品关，毕竟打铁还需自身硬才行啊。就好比你开一家服装店，地理位置再好、客流量再大、员工管理再现代化……但是你的衣服质量很差，相信你的服装店开不了多久就要关门大吉了。所以从这个角度上说产品思维决定了一家公司的生死存亡。

看到这里，也许有小伙伴会说：“我知道产品思维很重要啊，但那是产品经理需要考虑的事，我一个小测试，只保证软件没 bug 就好了啊。”

事实并非如此：就像我们在推行敏捷开发理念时，曾提到过的---软件质量不是测试工程师一个人的事，而是整个团队的事。同理，产品思维也并不是产品经理一个人的事，也是整个团队的事。如果你是一个初入行的测试人员，你可能只需要了解产品功能、搭建测试环境、编写用例、执行用例、提交测试报告即可……但是如果你是一个有了三四年经验的测试人员，你的思维模式就要从技术模式转向产品模式，因为在一家公司里任何技术最终的目标还是服务用户的，思维模式不转变的话就很可能在互联网的浪潮里被淘汰掉了。

比如 Leader 让你测试一个注册页面，初级测试人员只需要照着需求文档编写用例然后执行即可，比如测试每个输入框的正常异常输入情况，多个输入框的排列组合情况，输入完成后的信息能否顺利写入数据库，最多支持多少用户同时注册等等……但是作为一个中级以上测试人员，你可能就会在测试过程中，更加注重用户体验，也会注意到一些初级人员注意不到的问题，比如：所有个人信息都放在一个页面的话，会让用户产生畏难心理，甚至会放弃注册；支持使用外部账号先登录再注册，方便一些新用户可以先体验产品……

三、如何培养自己的产品思维

关于如何培养自己的产品思维，我想无非五点：多观察、多学习、多体验、多交流、多实践。

多观察

产品思维来源于生活，又高于生活。

比如去超市买东西，结账的时候总是要排很长的队；比如去医院看病，传统的流程会让你楼上楼下跑断腿；又比如，去大排档撸串，你发现服务生经常上错菜……

这时候，如果你不仅仅是抱怨商家的不给力，而是开始思考出现这些问题背后的原因是什么，有什么方法可以优化他们原有的业务流程，提高商家服务效率，进而提高客户满意度。这既是一个在生活中发现痛点，进而想办法解决痛点的过程；又是一个培养产品思维的好机会。

多学习

活到老，学到老。

当今社会，学习并不仅仅是学生时代的事情，终身学习才是时代的主旋律。不要给自己找借口说自己很忙没时间看书什么的。见过太多的人手机刷朋友圈可以刷上一上午，买的书没翻几页就开始犯困了。而且个人觉得把一本经典书读上 100 遍，胜过随意浏览 100 本书，“书读百遍，其义自见”就是这个道理。推荐几本培养产品思维的经典好书给大家:《浪潮之巅》《人人都是产品经理》《设计心理学》。另外，除了利用整块的时间读书之外，还可以利用零散时间看一些业内有名的公众号推文。

多体验

所谓多体验，就是多用别人的产品，边使用边思考这款产品的优劣之处。

下载别人做的 APP 之前，你可以先了解那家公司的信息、产品定位、目标用户群体、商业模式和核心卖点。打开 APP 将其核心功能流程过一遍，想想这款 APP 主要解决了用户的那些需求，还有那些地方可以优化。然后可以用 XMIND 软件把这款 APP 的思维脑图画出来，想想为什么设计者这样设计产品的各个模块。再从外观上，看看它的设计风格是不是符合目标用户的主流审美观。最后思考一下，如果你是这款 APP 的测试人员，你会从哪几个方面进行测试，你觉得哪些地方最有可能出现 BUG，哪些地方还可以进一步优化……

多交流

“思想碰撞出智慧的火花”“听君一席话，胜读十年书”

不管是和测试同行交流，还是和产品、设计、开发交流，还是同客户交流，其实都能学到不少的东西。

比如你和测试交流的时候，你可以了解他们用了哪些最新的测试理念和测试技术，这些理念 or 技术可以解决哪些工程上的难题；你和产品 or 设计交流的时候，你可以学到他们的产品思维 or 设计思路，了解他们设计产品背后的逻辑是什么？他们期望产品最终达成的效果是什么？了解这些可以帮你更好地设计测试用例；你和开发人员交流，可以了解更多技术细节，分析技术风险，甚至提前判断出哪里会出现 BUG；你和客户交流，可以了解到他们的核心需求是什么，哪些地方应该是你重点关注的，需要多加一些测试用例，哪些地方是重要性没那么高的，这样方便你合理分配测试资源。

多实践

“纸上得来终觉浅，绝知此事要躬行”“实践出真知”　　“光说不练假把式”

产品思维如果不付诸于实践，不融合到我们日常的测试工作中去，它将始终是空中楼阁。

你可以把你观察到的、学习到的、体验到的、交流到的产品思维，那怕只是很小的一点思路，运用到日常测试工作中。比如你开始关注用户注册的复杂程度了，你可以把这个作为一个用例去执行.度量一下注册需要填写多少条信息、整个注册流程需要花费多长时间，注册过程需要手工设置哪些权限，同时容许多少用户注册等等……

总而言之，产品思维是一种不同于技术思维的思维模式，这种思维模式也是广大测试工程师们需要学习的。因为它能帮你开拓视野，跳出技术思维的局限性；它能帮你更好地理解用户需求，更好地保证产品质量；它能帮你成为一名优秀的测试工程师！

软件定义世界，质量保障未来

◆ 作者：合肥人真帅

只要是人们提出来的想法和需求，软件都可以实现，只是时间和成本的问题，一旦实现了，那么质量就是衡量你工作能力的标准。对于企业来说，只有高质量的产品才能保障企业的竞争力。IT 行业在众多人的眼中，是一个熟悉又陌生的行业。熟悉是因为日常生活中我们经常使用它，例如微信支付、百度搜索等。陌生是因为对软件的理解仅仅只限于应用层。其实软件和我们日常看见的交通工具，家用电器都有着类似的生产过程。其中一个环节就是质量检测。软件测试就是保证质量的一个重要手段。

软件测试没有技术性？软件测试收入偏低？软件测试前景堪忧？大部分人都这么认为。笔者也曾经迷茫、徘徊过，但最终也是坚定了方向。我将以实际经验和大家探讨一下上面提出的 3 个问题。希望可以让迷茫、徘徊的朋友们，找到方向。

一、软件测试没有技术性？

大多数公司的老板、领导，以及身边了解软件行业的朋友，同学。嘴上都会说软件测试很重要，但打心底都会认为软件测试毫无技术可言。甚至有些软件测试工程师自己也这么认为，软件测试纯粹就是点点点。对于创业公司，为了节省成本，甚至可以舍去测试。那么软件测试是否重要？我举个例子大家就明白了。一辆汽车，所有的零件没有经过质量检测。组装成整车后，也没有检测过。那么你得知这些信息后，还敢买这辆车吗？当然不敢，万一在开车过程中爆胎怎么办，刹车失灵怎么办。软件也是一样。

软件测试是否真的没有技术可言？首先我来带大家看一下软件水平考试。

什么是软件水平考试？计算机技术与软件专业技术资格（水平）考试（以下简称软件水平考试）是原中国计算机软件专业技术资格和水平考试（简称软件考试）的完善与发展。这是由人力资源和社会保障部和工业和信息化部领导下的国家级考试，其目的是，科学、公正地对全国计算机与软件专业技术人员进行职业资格、专业技术资格认定

和专业技术水平测试。

说白了，软件行业不像会计、建筑、消防那样有从业资格证，更加没有注册资格证。如果有，那么这个考试的初级就相当于是从业资格证，高级就是注册资格证。

	计算机软件	计算机网络	计算机应用技术	信息系统	信息服务
高级资格	信息系统项目管理师　系统分析师　系统架构设计师　网络规划设计师　系统规划与管理师				
中级资格	软件评测师 软件设计师 软件过程能力评估师	网络工程师	多媒体应用设计师 嵌入式系统设计师 计算机辅助设计师 电子商务设计师	系统集成项目管理工程师 信息系统监理师 信息安全工程师 数据库系统工程师 信息系统管理工程师	计算机硬件工程师 信息技术支持工程师
初级资格	程序员	网络管理员	多媒体应用制作技术员 电子商务技术员	信息系统运行管理员	网页制作员 信息处理技术员

没错，你没有看错，程序员是初级，软件评测是中级。这是国家级考试，给出的等级划分。很多人乍一看觉得不合理，程序员技术含量应该比测试要高啊。但是细细想来，这个划分是正确的。站在国家的角度，他要求的不仅仅是黑盒测试，而是需要了解软件的原理，对被测软件给出一个合理的评价。单纯的黑盒无法评价软件的性能，无法评价软件的稳定性，无法评价软件的安全性。大部分企业为了节省成本，软件测试都是执行黑盒测试，因为这样成本最低，产出最高。久而久之大家就会产生软件测试没有技术可言的误解。

那软件测试的技术性到底体现在什么方面？首先黑盒测试是软件测试的基本功，是软件测试工程师必须精通的技能，包括测试流程，测试用例设计等。然后是专项测试，自动化、性能、安全等等。专项测试可以继续细化，例如自动化可以分为 UI 自动化、接口自动化等。专项测试的技术包括但不限于：编程语言，数据库，操作系统，数据结构，网络环境，系统部署，持续集成等等。成为一个高级软件测试工程师或测试专家，所需要的技能，一点不比其他 IT 岗位少。

二、软件测试收入偏低？

接下来是大家比较关心的问题——收入问题。目前我国从事软件测试的人员中，能力参差不齐。初级软件测试工程师收入很低，因为纯黑盒而言，任何一个可以熟练操作键盘鼠标的人，经过一段时间的培训都可以达到大部分企业对初级测试的要求，并且这

个培训时间不会太长。可替代性强的岗位，薪资不会高。擅长专项测试中的一项或多项，收入就比较可观了。

这是 2018 年 51Testing 调查获取的软件测试从业人员的工资分布图和工作城市分布图，受访者 60%来自于一线城市。北京上海 2018 年平均工资都是 7800， 7900 不到。从图中可以看出，有 45%的软件测试人员收入是高于当地城市平均工资的。这样看来，

你还觉得收入低吗？

三、软件测试前景堪忧？

第一个问题，软件测试是否会被人工智能机器人取代？近几年，随着人工智能的火热，很多行业的很多岗位都受到了人工智能的冲击。例如亚马逊的人工智能监督系统，物流行业的智能机器人代替人工分拣等等。随着人工智能的发展，我认为一部分简单的测试工作，会被人工智能取代。但软件测试中，对于软件的审美、易用、测试用例设计所需的创造性，创新能力，是无法被替代的。第二个问题，软件测试发展前景如何？近几年，部分 211、985 大学已经在本科段和硕士段开设了软件质量保障这个专业，说明国家已经在重视培养这方面的人才。目前国内还是比较缺少软件测试人员的，尤其是专项测试。软件测试的发展历史不过短短的 40 年，走入我国百姓的视野也不过就十几年，是一个非常年轻的行业。人类对于品质的追求是极高的，软件测试作为提高软件质量的重要手段，发展前景自然不会差。

四、总结

随着我国电子信息产业的发展，目前对于中高级人才的需求量极大。若想在 IT 行业站稳脚，在软件测试岗位继续深入下去，那么编程，数据库，操作系统是必不可少的技能。即便是做管理岗，不做一线技术人员，对于这些技能也要达到熟练的程度。无论什么产品，质量都不会缺席。希望迷茫的朋友可以找准自己的方向，用双手创造高质量的产品，保障未来，也许下一个改变世界的就是你。

商业银行测试数据安全管理实践

◆作者：孙绍伟

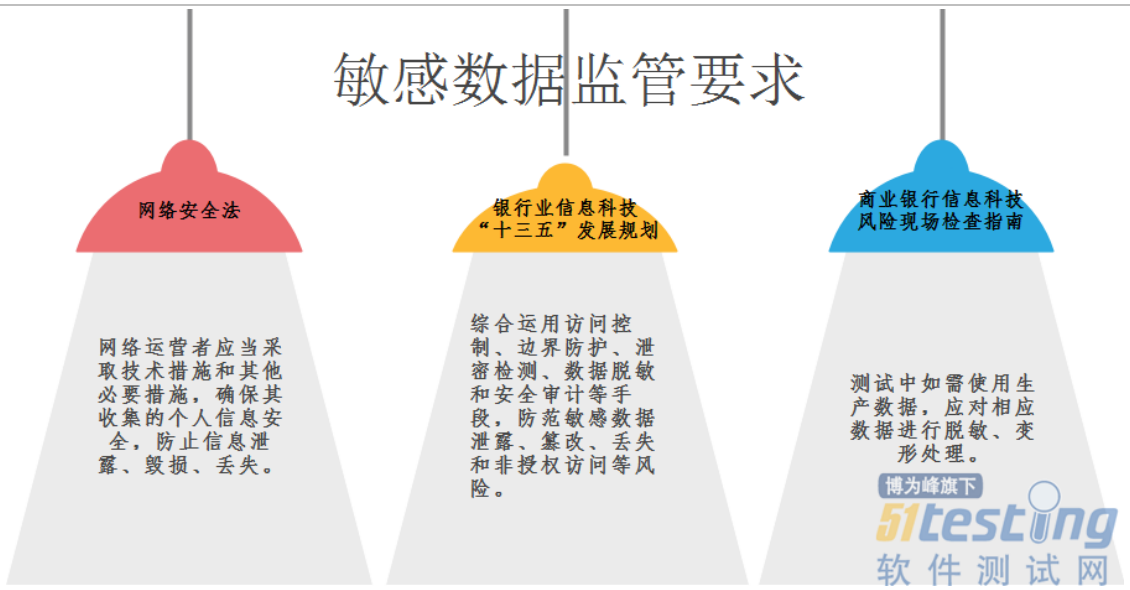
信息化时代，数据是企业的核心资源，是企业运营过程中积累的宝贵财富。随着银行业务的快速发展，业务系统积累了大量包含账户等敏感信息的数据，作为企业既要保证数据在产生、交换、共享等场景下的高效可用，又要保护敏感信息不被泄露。

软件研过程中需要使用大量的测试数据进行功能和性能的测试工作，银行一般采取数据脱敏技术和安全的数据管控手段为开发和测试人员提供非生产用数据，有效保护银行客户的敏感信息不被泄露，从而达到数据安全管控的目标。

一、为什么要数据脱敏

银行积累的大量数据如果发生泄露，不仅会给银行带来直接的经济损失，而且会给银行的声誉带来重大负面影响。以 2018 年著名的 Facebook 泄露 8700 万用户数据为例，公司股价一度蒸发 590 亿美元，而且欧盟、美国、英国等先后对 Facebook 公司提出调查，导致 Facebook 面临失去用户信任的重大声誉危机。

除了需要承担市场、社会等外部风险外，同时我们还面对诸多的的监管合规性要求。目前国外有欧盟 GDPR（《通用数据保护条例》），国内有《网络安全法》以及与金融业密切相关的《商业银行信息科技风险管理指引》。同时我国正在积极制定《中华人民共和国个人信息保护法》。各商业银行也先后出台了《客户信息保护管理办法》、《信息系统数据管理办法》等制度。



二、敏感信息判定标准

当前，某商业银行数据敏感性界定和判断标准主要依据《非生产环境数据敏感性分类与脱敏方法》，该标准主要是从密钥密码、客户信息、员工信息、账户信息、交易信息、经营管理信息角度进行敏感信息分类。近年来随着外部监管要求不断提高，同时结合金融业信息系统特点，今年又进一步将敏感信息细分为关键敏感信息和重要敏感信息。

关键敏感信息为能直接识别到特定数据主体的信息，如客户名称、证件号码、移动电话号码（客户）、邮箱（客户）、车牌等信息。此类信息实施脱敏后，其它信息无法与数据主体相互关联。

序号	参考信息项	敏感类别
1	客户名称	客户信息/员工信息类
2	证件号码（身份证、军官证、护照、驾驶证、工作证、社保卡、居住证、组织代码等）	客户信息/员工信息类
3	移动电话号码	客户信息类
4	车牌	客户信息/员工信息类
5	邮箱	客户信息类

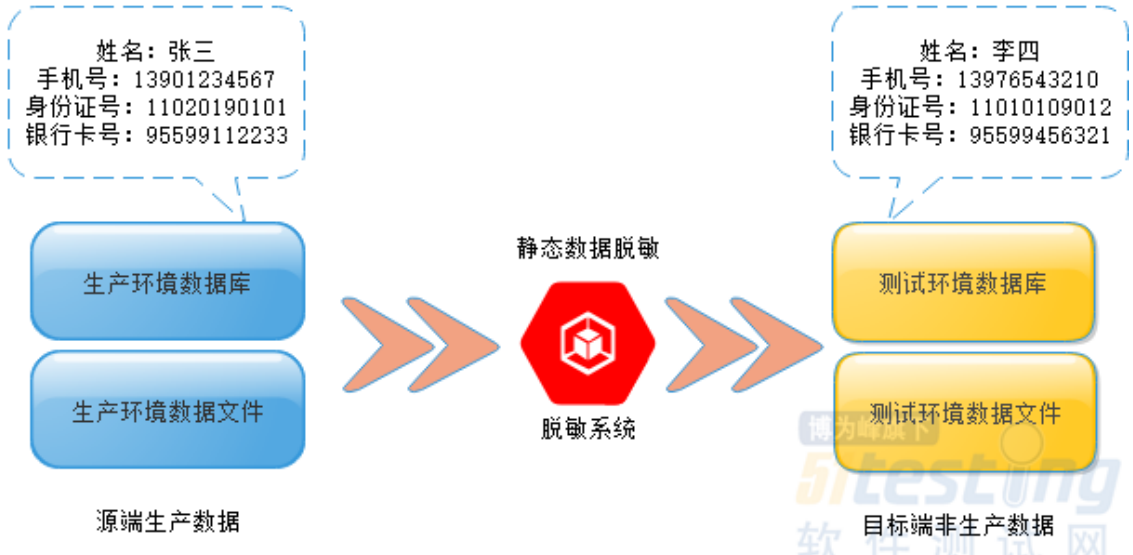
重要敏感信息的界定思路和信息参考项为发生客户交易的一个或多个组合信息项，客户和行内用户密码密钥，存储生产系统的 IP、用户、密码等信息项;

序号	信息参考项	敏感类别
1	银行卡号、有效期、CVV 码（组合）	账户信息类
2	资产兑换码	
3	银行账号、卡号与之有关鉴别口令、密码、密钥、数字证书等（组合）	账户信息/密钥密码类
4	系统账号、邮箱地址与之有关的密码、口令、口令保护答案，token、用户个人数字证书等网络标识与认证信息（组合）	
5	系统账户与预留印鉴	
6	系统账户与指纹、虹膜、声纹、面部识别特征等客户生物特征（组合）	
7	卡密钥	密钥密码类
8	生产系统数据库/系统/中间件等用户及密码（组合）	经营管理/密钥密码类
9	生产系统的 IP、端口等对外服务接口信息	经营管理类
10	尚未公开或披露的财务、监管等管理数据	
11	基金/第三方托管/养老金等未披露信息	客户信息/账户信息类

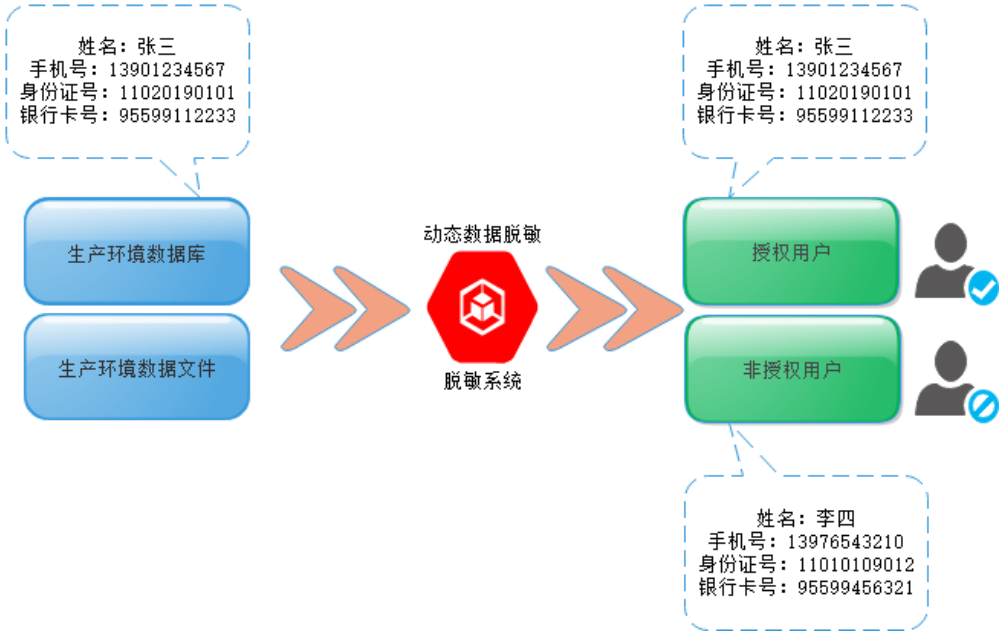
三、数据脱敏技术实践

数据脱敏技术按照应用场景的不同分为静态脱敏和动态脱敏:

静态数据脱敏（SDM）一般用在非生产环境，在敏感数据从生产环境脱敏完毕之后再在非生产环境使用，一般用于解决测试、开发库需要生产库的数据量与数据间的关联，以排查问题或进行数据分析等，但又不能将敏感数据存储于非生产环境的问题。



动态数据脱敏（DDM）一般用在生产环境，在访问敏感数据当时进行脱敏，一般用来解决在生产环境需要根据不同情况对同一敏感数据读取时需要进行不同级别脱敏的问题。



受应用场景和脱敏技术的限制，当前主要采取静态数据脱敏技术为非生产环境提供数据，主要包括:

删除: 是将数据中的部分或全部敏感信息直接删除的处理方式。删除处理应在分析对应用逻辑的影响后进行，经过删除处理后的数据在数据量上仅为原数据的子集。为保证数据可用性，某些情况下部分删除可能需配合修改应用程序，对应用逻辑完整性和一致性的影响应进行评估。

简单置换: 是将数据中的部分或全部敏感信息，通过简单的映射规则，替换为一组或者几组对应数据的处理方式。经简单置换后的数据能保持原数据量，保证应用逻辑的完整性和一致性，但不能体现原数据业务特征和分布特征。

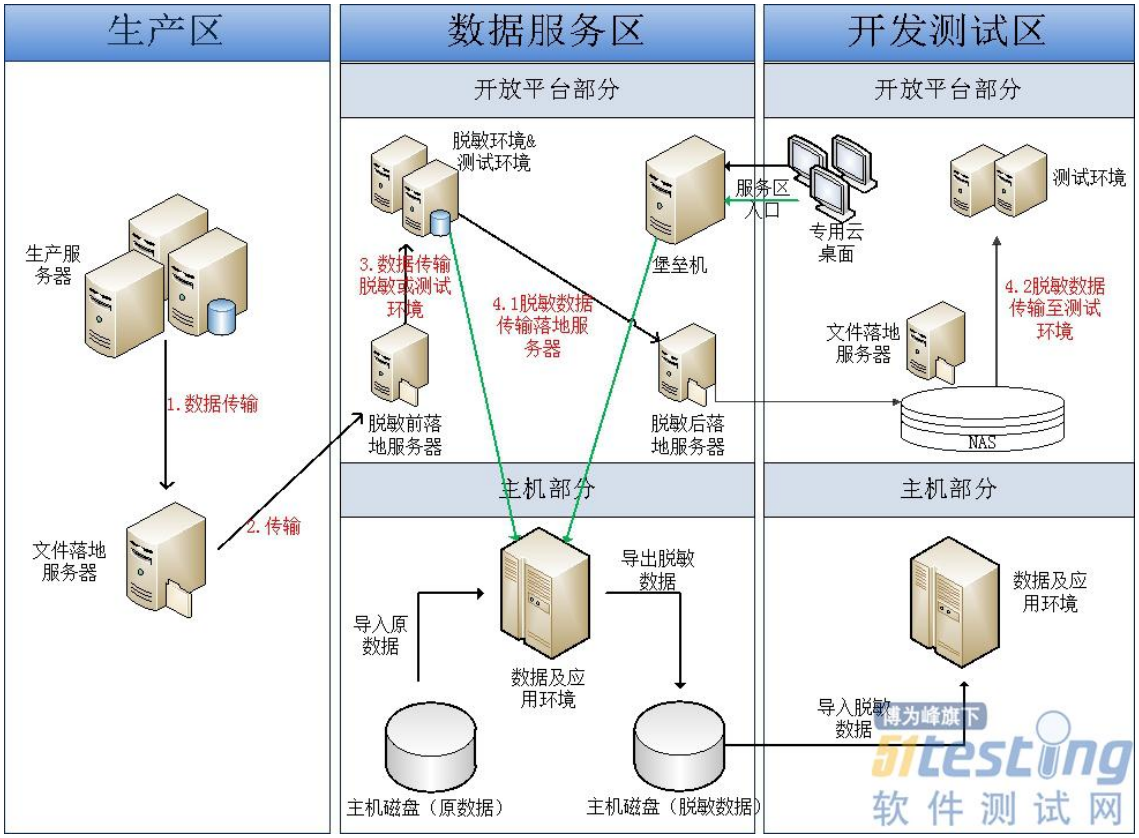
仿真置换: 是将数据中的部分或全部敏感信息，通过预先订制的规则，替换为对应数据的处理方式。经仿真置换后的数据能保持原数据量，保证应用逻辑的完整性和一致性，保持原数据业务特征和分布特征。

四、规划建设数据服务区

在采用静态数据脱敏的情况下，如何在信息不泄漏的前提下完成各项工作任务一直

是行内各部门面临的问题。为此，某商业银行于 2017 年搭建数据服务区，旨在通过网络隔离、审计等方式加强敏感信息的保护，确保数据脱敏，业务测试与分析场景中数据安全可控。

数据服务区通过专用虚拟桌面、网络访问控制等方式构建一套专用的数据安全防护区域。简单来说，数据服务区是参照生产网建立的一套管理体系，与传统的生产、开发测试网相互隔离，构成一个单独的区域。



数据服务区自 2017 年投入使用以来，已累计处理 1000 多单开放平台测试数据申请，领域覆盖产品与服务域、业务管理域、企业数据服务域等六大领域，有效支持了综合财会平台、智慧大脑等多个全行重点项目的测试数据申请，全力保障测试工作正常有序进行。

目前在数据服务区已经搭建多套常备数据脱敏和测试环境，涵盖 Oracle、Sybase ASE、Sybase IQ、Gbase 等多个数据库产品与版本，基本可以涵盖大部分开放平台测试数据的申请需求；同时正在实践主机脱敏系统纳入数据服务区的工作，日后随着其他主机系统的纳入，将实现一整套全量开发、测试环境，满足大部分测试环境在服务区内测试的需求。

近年来，随着银行应用系统的建设步伐逐步加快，数据脱敏需求也正同步快速增长，持续改进测试数据服务各环节的工作方法，快速响应测需求，有效地满足了开发和测试等部门的需求。后续，将在自动化脱敏、测试数据安全监控等方面加快步伐，为研发工作提供高效、专业的数据服务。

7 年测试工作让我体会到倾听的重要性

◆ 作者：小鱼儿

今天读到一篇文章，文章题目为《7 年软件测试工程师，如何突破职业瓶颈？》里面提到倾听的重要性，在我们的日常生活与工作中倾听真的很重要，{学会倾听。生活中我们往往会主动寻找倾诉对象，做一个倾诉者，而回避做一个倾听者。在人与人的交往中，倾听是一个不可缺少的方面，有时它甚至比交流还要重要。学会倾听，能修身养性，陶冶性情；学会倾听，能博采众长，弥补自己考虑问题的不足；学会倾听，能使人萌发灵感，触类旁通；学会倾听，能养成尊重他人的良好品质，创造一个良好的人文环境；学会倾听，能体会一种默默无语的关心和体贴，赢得亲情、爱情和友情；学会倾听，还能在矛盾激化时，让对方从另一个角度重新认识自己，打开僵持的局面……}

但是很不幸运的是我就是那个倾听能力极为薄弱的人，我会不自觉地打断别人在说的话，有时候甚至会抢着去说自己的想法，不等别人说完，不等自己去理解别人的对话内容。当在 2019 年 10 月 8 号即将到来的日子里，在自己在测试这个行业工作 7 年之久我也深刻体会到了倾听的重要性。

我是一名工作将近 7 年的测试人员，从最初的实习生测试服务器，到现在去当 12 位测试同事的测试主管，我感觉我不能再是那个火急火燎的性子，工作需要我更多的去理解工作内容，理解同事的想法，我不能将自己的想法强加到别人身上，因为我知道当有人这样对我时我是多么的讨厌这个人，所以在日常的工作和生活中我希望更多的去倾听别人的所思所想，让同事，朋友，亲人去表达他们的想法，无论当你听完这个想法之后觉得是有多么的不可取，那也是有价值的，因为倾听其实也是对人的尊重，因为谁都不想当那位别人不愿意让你表达想法的人。

2015 年我在福州的一家外包公司上班，入职的乙方公司有一位小姑娘，因为她比我早入职，况且也是甲方职员，只能说比我更加熟悉那个产品，但是那段短暂的 2 个月的

工作经历让我是筋疲力尽，原因主要有三，一是我的 baby 还在哺乳期；二是工作地点距离我居住的地方有将近 2 个小时的公交路程，况且我还晕车；三是这位员工脾气很大，虽然年纪很轻，但是职场的老套和坏脾气却是我工作到当时唯一让我深刻的。其实这也是职场的一种通病，往往很年轻作为技术管理或者人员管理的管理者有着相对的技术能力但是缺乏为人处世的技巧，他们的理解里职员除了工作还只是工作，忘却了身为人这一个单个个体的我们还有着很强烈的生活属性，还有着很多的面子等等我们其实不必须但是又要命的东西。那段工作经历是在那位小姑娘的指责，谩骂还有每天的晕车晕吐当中度过的，2 个月在拿到 1k 左右的薪资当中离职了，说来真的很不值得，但是现在想想也是一段比较特殊的体验。

后来我想想如果那位小妹妹在当时如果能给我营造一个相对轻松的工作环境，一个相对轻松的工作氛围，我是不是会呆的时间长很多呢？因为我记得我妈妈和我姐姐说过：“人其实身体累点没有关系，心里不累就可以，毕竟这个世界累不死人”。

或许她没有这样的能力，公司的企业氛围也影响了她。或许那位小姑娘的老大之前就是这样对她的，而且当时的她才那么年轻，怎么能去奢求她能体会和她一样年纪但是已经身为人母的我的处境呢？其实那家甲方公司除了那位员工之外还是有一位比较理解我的人，同样身为 mom 她比较理解我的难处，她也是面试我进入那家公司的面试官，我应该感谢她因为那是我在上海工作了 3 年之后去到一个陌生的城市面试了将近 20 几家当中唯一一家给我机会的公司，虽然对现在的我而言那样的工作机会毫无任何诱惑可言，但是在当时在我那个工作年限，那个二线城市，那个处境给了我唯一一点安慰。直到现在我能脸不红心率不起伏地描述那段经历。

2017 年我的生活轨迹变化了一点，在来到杭州的 1 年半之后我辞职去到一家外包公司，只能说这家外包公司毕竟也上市了，去到了乙方一家国有商业银行，中国四大商业银行之一的某银行，在这家老牌国企干的工作还是测试，带我的那位组长比我年龄小，虽然是国企但是企业氛围节奏很快，每个人都是心浮气躁，要的东西很快，给你思考的时间很少，企业也很形式化，领导氛围很重，测试一个模块你需要给出很多 PPT，PPT 的准备很费时间也是我最不愿意写的东西，这期间和这位年轻的组长之间有过很大冲突，这让我在这段工作经历当中可以说有不下几百次想辞职的冲动，终于在 19 年元旦离开了那里。这期间发生了很多事情，我的老大换了一任又一任。其中也有一位我至今见过的能力强脾气极好的老大，如果不是因为她的调动可能我还会继续在那个组呆下

去，只是她调到别的组也于今年顺利诞下她的 baby，很是为她高兴和感动，前几个月我和之前的同事还一块去看了她，至今我们还保留着之前交流的微信群，大家还是像老同事老朋友一样联系着，虽然这期间我们五个人发生了很多的变化：有一位美丽的小女孩在很早之前就回到了她的老家西安发展；一位高瘦的小伙子去了华为外包；一位有趣的小伙子去了蚂蚁金服外包；而我去到一家电力公司，逃离了外包的行业。但是我们似乎还是在一起着，像是老朋友一样。

现在我的工作发生了很大的变化，我成为了 12 个小伙伴的老大，成为了测试组的主管，日常工作中我发现我有时候甚至很多时候会保留着那些浮躁的坏脾气，很少会认真倾听别人的想法，我总是把我的想法强加于他们身上，让工作氛围变得紧张，让大家感觉工作很累，这时候我依稀能在我自己的身上找到我当时那位坏脾气老大的影子，这时候情不自禁的让我想起了在电视剧里面的那句经典的台词，“我终于成为了我最讨厌人的模样”。

意识到这些之后我发现我也在逐渐的发生改变，我会适当的放慢脚步，不会再催的那么紧，我会适当的放手，让大家自己做主，不在是以一位领导者下达命令的口吻，更多的时候会尊重员工的个性，我发现我的这些适当改变让大家工作氛围变得没有那么紧张，大家也更加愿意和我共事，现在我们在一起相处的很愉快。

未来我只想成为这样的人：当落千年之后当他们在想起他们这位曾经的老大时脑子里不在只是那个发号司令颐指气使的我，而是一位有工作能力有为人处世技巧相处愉快的我；当他们回忆起这段工作经历时不是不开心和满脑的抱怨而是轻松愉快未来还想拥有这样体验的工作回忆的我；当他们离开时不是逼不得已而是因为未来的工作有更大的发展空间，去追逐更好的工作阅历。

我希望当你看到这篇文章时能有所共勉，如果你是一位初入这个行业的测试人员当你看到这篇文章时你就能理解你为啥会有这样一位老大，或者很幸运至今你都没有碰到过这样的人，你的工作伙伴很好那就当看这篇文章有不一样的体会，更加轻松吧；如果你初为测试组长，测试老大，测试主管，测试经理，当你看到这篇文章时当你曾经和我一样为人难以相处时我希望你看完这篇文章之后可以意识到那样的我们其实很不受欢迎，并希望你我为此在未来发生适当的改变；如果你是一位入行很多年，很有经验和心得体会的老朋友，那我很期待再未来的你能适当的分享一些此类体会让我们共勉

自动化测试：预防还是治疗？

◆ 译者：咖啡猫

一、简介

很多团队都倾向于认为自动化是一种加速软件交付的方式，因为这是团队内部经常能够感知到的瓶颈。但是，如果他们将自己的开发实践过程当作一个整体来深入研究的话，那么他们会得到更好的结果。

二、预防 BUGS

测试，尤其是 UI 层面的自动化测试，通常会被安排在软件交付周期的最末端，通常会试图发现一些可能进入生产环境并给最终用户造成不良影响的 bug（这些 bug 就如同细菌一般）。在这种情况下进行的测试可以探测出 bug 的症状，开发人员进行的修复就是治疗。这就如同我们在等待我们的系统生病并为此采取一些措施。

这种方法对于团队来说是有一定的效果，但是，现在的工作环境迫使我们用更少的人做更多的事并且要比以往更快地完成任务。因此，这种方法并不能长期可持续地运转。这时基于预防的方法而非治疗的方法就横空出世了。

通过对如何构建系统作出调整，我们就可以在故障发生之前探测出问题，或者更好的是，让他们从一开始就不那么容易产生 bug(这就是所谓的“预防”)。这就意味着我们在 bug 产生前就预防它的发生而非试图在 bug 产生后再治疗它。古语有云：预防胜于治疗！

三、我们的测试自动化之旅

在我刚开始加入移动团队（该团队负责创建并管理公司的点播产品）时，我们所有的测试都是手工执行的，平均每年我们只在每个平台上发布 2 到 3 次。我们知道如果想要加快速度，最明显的瓶颈就是测试。在没有发现新 bug 的情况下，每个回归测试周期

要花费将近两周的时间。如果发现了新 bug，那么开发团队需要时间理解并确认 bug，确定一个修复方案，然后再应用这个方案。这会导致已经执行的任何测试都变得无效，因此需要重新启动测试过程，最终导致测试周期花费两倍以上的时间。

因此，我们开始着眼于将更多的 UI 测试进行自动化处理。我们总是从小的方面开始改变看看这样的尝试能否将我们引入我们想要的正确的方向上，并且我们只选择对新功能进行自动化测试。这种尝试一旦被证实是有效的，我们就会将自动化测试引入到现存系统的其他领域或者已知的问题领域。

我们组织 3 个朋友作为一个团队来理解我们想要构建什么，以及该特性的关键接受标准应该是什么。这为我们提供了一个起点，让我们了解如何分解该特性以及哪些用户场景需要自动化。

在此基础上，我们确定了可以用来自动化测试的工具(Calabash 和 Appium)，并在实际环境中应用它们。对我们来说，这是在真实的手机上进行测试，而不是模拟器，为此我们建立了自己的设备测试场以更好地利用我们的移动设备，同时也允许它扩大规模以期能够在整个组织中得到应用。

四、测试自动化带给我们的好处

起初，自动化给我们带来了很大的帮助：通过自动化我们可以快速可靠地运行简单场景的测试用例并迅速得到我们想要的反馈结果。但是随着时间的推移，在最初的一批 bug 被捕获之后，自动化测试很难再发现新的 bug 了，除非我们手动地修改自动化用例代码。

同时，我们也注意到由于某些场景我们无法自动化导致一些 bug 依然存在于系统之中：比如，任何与可用性相关的功能都不得不进行手工测试。因此，我们最终采取了一种混合的解决方案---自动化用来快速跑一些关键场景，让团队知道自动化测试没有明显地破坏任何东西，以及如果合适的话对任何新功能进行的探索性测试也可以自动化处理。因此，尝试自动化测试的过程是曲折的，我们在尝试测试时很容易出错，又或者手工测试花费的时间太长。

带给我们自动化之旅的一个意料之外的好处就是我们开始更快地发布产品，自动化让我们更加专注于我们正在尝试完成的工作事项。这就让我们可以将每一个新特性新功能拆分成更加细小的分支（可以独立运行的分支）并将这些分支进行自动化测试。这使

得我们可以更快地将各个分支发布到生产环境中并从最终用户处得到实时的反馈。起初这一好处并不明显，因为我们仍在尝试着识别自动化场景。只有在事后，团队才能看到这是他们无意中所做的事情带来的价值。简单地说，我们开始将工作分解为一小批最终用户价值。

五、研究我们的开发模式

我们开始意识到 UI 自动化测试并没有真正带给我们想要的回报。正因为如此，我们开始研究开发过程中的其他领域，看看是否可以做出些许改进。但作为一个团队，我们的问题之一是，我们太过接近流程，以至于无法客观地看到哪些是有效的，哪些是无效的。为了克服这个问题，我们请来了一位敏捷教练来帮助我们的团队。事实上，我们引入了两位教练：一个是帮助团队理解他们正在使用的流程，另一个是帮助我们更好地理解我们实际上是如何构建我们的系统的。

敏捷教练都来自于团队外部，因而他们可以质疑我们系统的任一部分，而不用担心冒犯任何人。他们会问一些简单的问题，让我们了解我们工作方法背后的原因，并将我们从“我们一直都是这样做的”循环中解脱出来。例如，用于管理我们工作的 stand up board 通常有 backlog、next、dev、wait - For -test、in test and done 等列，但是我们从来没有想过要问为什么我们有 next 和 wait - For -test 列。我们的教练能够提供帮助的是：我们为什么要将软件开发工作分解成这些小项，为什么开发和测试被视为两个不同的活动。教练的方法并不是简单地改变我们的开发过程，但他们能帮助我们明白问题的根源在哪(未发布的功能卡放在任务看板的 next 和 wait-for-test 列)，让团队看到通过消除开发和测试列（next 和 wait-for-test），取而代之的是一个简单的正在进行（in-progress）的列，很多工作依然可以顺利进行。您可以在我的文章 In test colum 中找到更多关于使用这种方法的好处.

六、我们学到了什么

我们发现的最大问题是：就敏捷开发实践而言，我们的团队中存在“形式主义”。仅仅因为我们有 standups（站会），以小团队的形式工作，并在 sprint 结束时发布东西，但并不意味着我们实际上是敏捷的。这只是意味着我们有一些仪式，让我们看起来像“敏捷”。事实证明，并不是每个人都很清楚我们为什么要这么做，甚至不清楚我们这样做的好处是什么。我们所做的第一件事就是澄清什么是敏捷：它更多的是基于客观反馈的可持续软件交付过程，而不是试图尽可能快地发布你所能发布的任何东西并期望最好

的结果。我们通过读书俱乐部促进团队讨论来实现团队内部对于“敏捷”一词达成一致的理解。这有助于团队成员更好地掌握敏捷实践背后的原则，并在他们的工作中做出更好的决策。

我们还开始研究我们实际上是如何在代码级别构建系统的，并试图将开发人员在如何提交代码、提交的频率和提交的大小这些内容可视化。这并不是试图让开发人员难堪，而是帮助他们理解作为一个团队，他们是如何影响代码库的，并试图鼓励开发人员养成更高效的习惯；比如对于较小的、有重点的任务进行提交，而不是在一天结束时的一次性提交大量的代码。如果他们确实做了大量的提交，那也可以，但是要让其他开发人员知道他们为什么这样做，来促进团队成员间相互学习。

我们对团队最大的改变之一是鼓励结对编程，因此没有一个开发人员单独开发一个特性。这加快了代码评审的速度，但他们也不太可能相互推卸责任。比起在某些项目中仅由资深成员来审查代码而言，结对编程还有助于更快地提高我们团队中较年轻成员的技能和知识，使他们能快速地提高生产力。

七、为了拥有更加高效健康的开发模式，我的一些建议

在团队工作中，需要在这个团队中确定一个高效、健康的开发流程。一个行之有效的方法是建立一个团队视频俱乐部。这允许成员从日常活动中抽出一些时间来学习构建软件的新工具或新方法。在每次会议结束时，团队讨论都由会议负责人(项目经理、技术负责人或将想法带给团队的人)推动，来探索如何使用这些概念来帮助团队尝试新事物。

然后选择一个概念，并对结果应该是什么有一个清晰的想法。我们以更好的单元测试为例，单元测试对团队意味着什么?更好的单元测试会给团队带来什么？一旦你有了这些答案，你就可以想出多种方法来达到这个目的，这样你就可以选择一个可以让你快速而客观地见证结果的方法。你需要弄清楚，你所使用的新过程或新技术是否真的帮助你在规定的时间内实现了你的目标。如果是这样，那就太好了。如果没有，你需要停止吗？做更多调整吗？您还需要决定谁将实际进行这一尝试，以及他们将如何与团队的其他成员进行沟通。

记住，如果你想让任何新流程或想法在团队中站稳脚跟，那么团队中每个人都需要参与其中；否则，遇到的第一个困难就是，这个想法将停止或缓慢执行，因为只有参与尝试的人才能从中有所收获。

Postman 进阶之变量&Collection runner

◆译者：桃 子

我们都知道 Postman 是接口测试工具，接口测试位于测试金字塔模型中的第二层，一般接口比较稳定，对接口进行测试效益最大。

本篇文章我将从以下几个方面进行介绍并和例子结合讲解:

一、学习 Postman 的常用资料

下载地址：https://www.getpostman.com/downloads/

官网文档：https://www.getpostman.com/downloads/

postman API 文档：https://docs.postman-echo.com/

二、Postman 的四个常用变量

什么是变量?

变量简单来说就是可以发生变更的值，比如登录功能每个用户都有属于自己的用户名和密码，这个用户名和密码我们就可以通过变量的方式进行更改操作

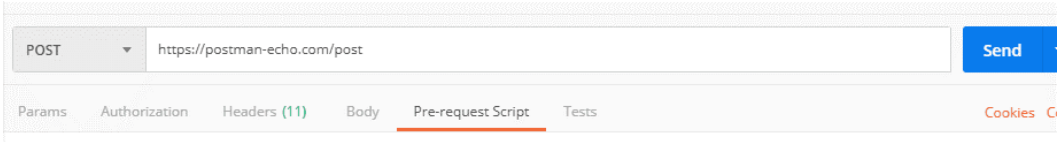
1、本地变量

本地变量: 就是在一个 url 内，设置 x 个变量为参数

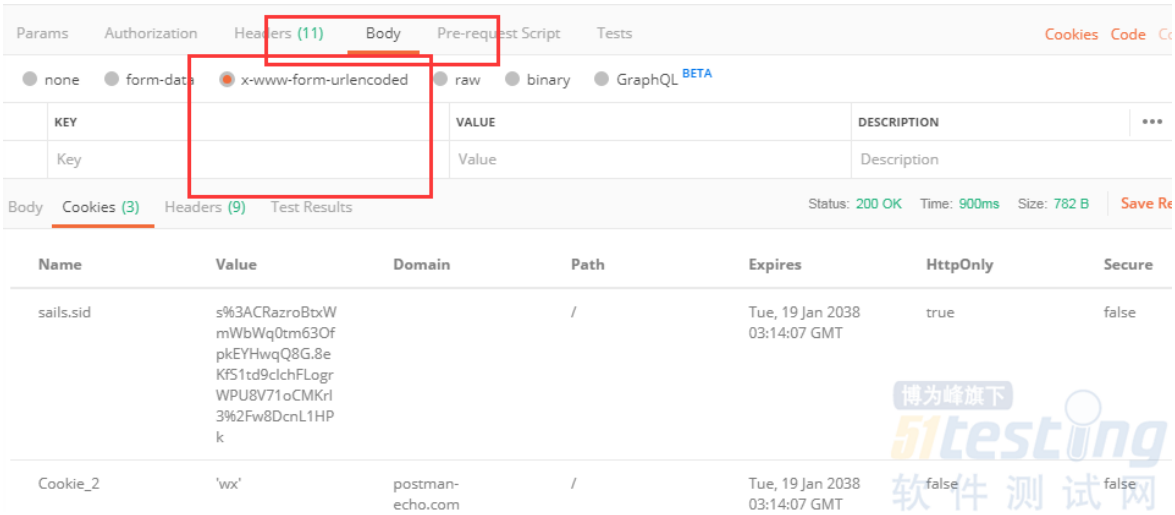
设置格式: {{变量名}}

案例一：本地变量的使用

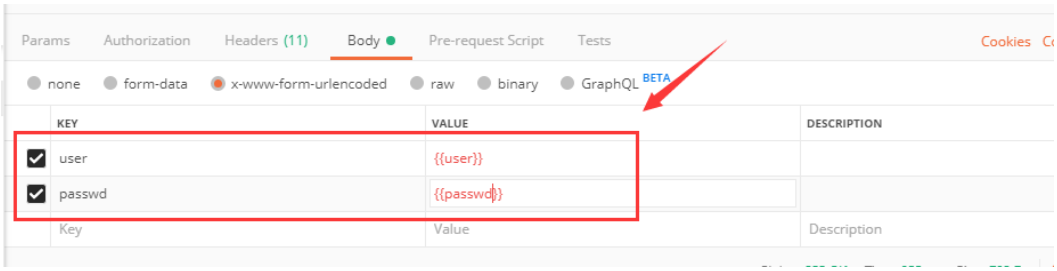
步骤 1.打开 postman url 处输入路径地址：https://postman-echo.com/post



步骤 2.切换 body-x-www-form-urlencoded 下



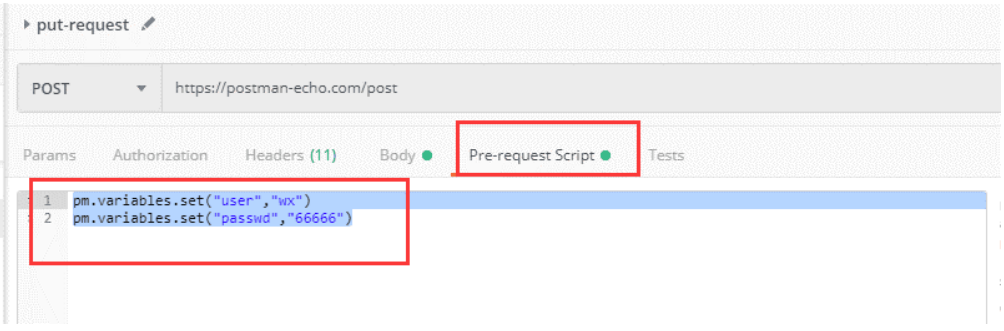
步骤 3.填写 key 及 value 值，注意内容要相同



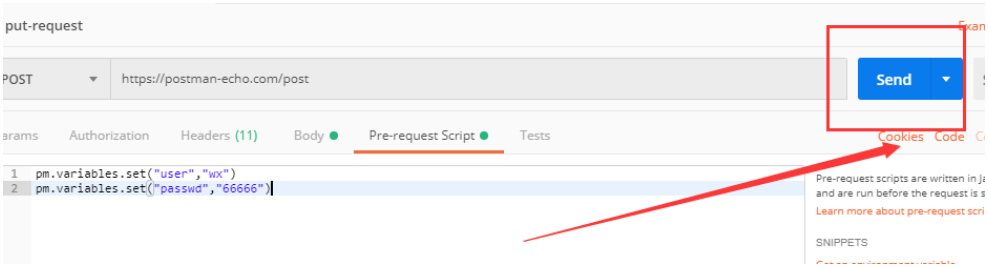
步骤 4.切换 pre-request script 下，填写变量内容

pm.variables.set("user","wx")

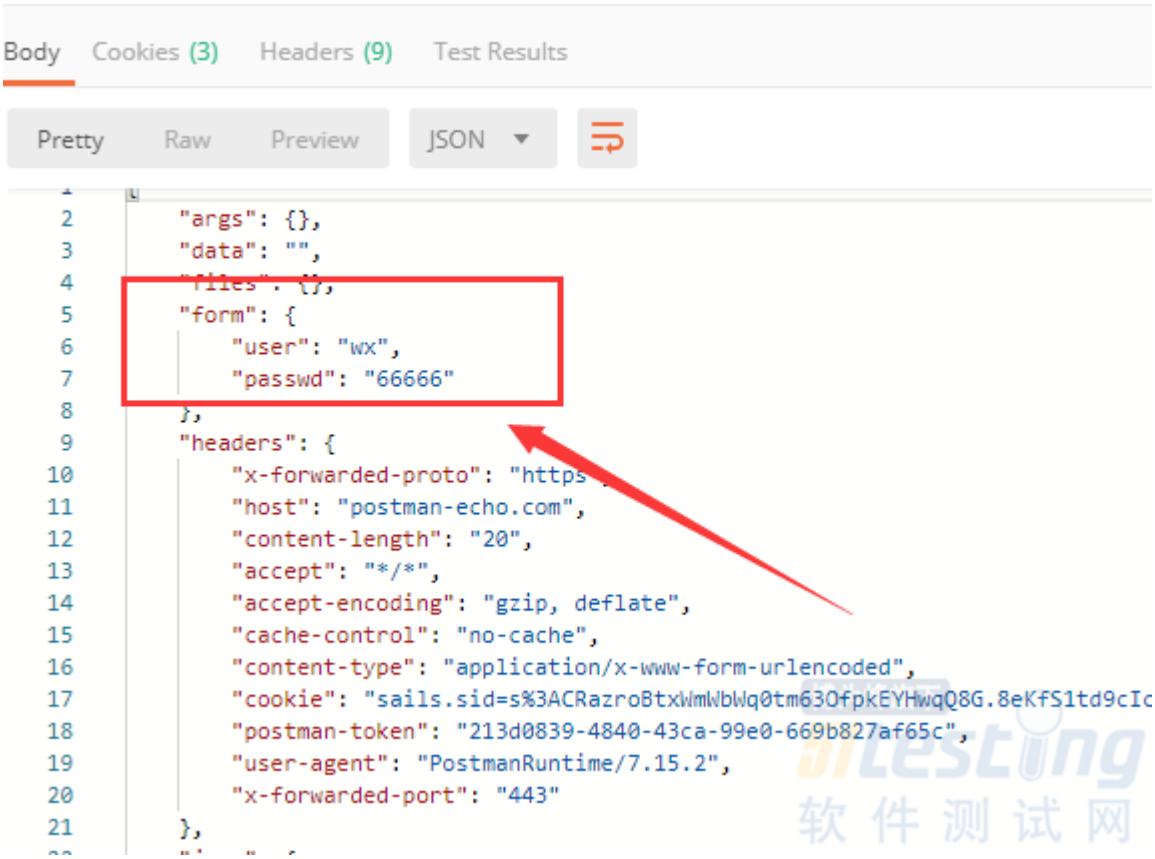
pm.variables.set("passwd","66666")



步骤 5.点击 send 按钮



步骤 6.查看结果显示刚刚添加的变量信息



2、全局变量

全局变量：适用于整个环境，适用于集合中所有请求

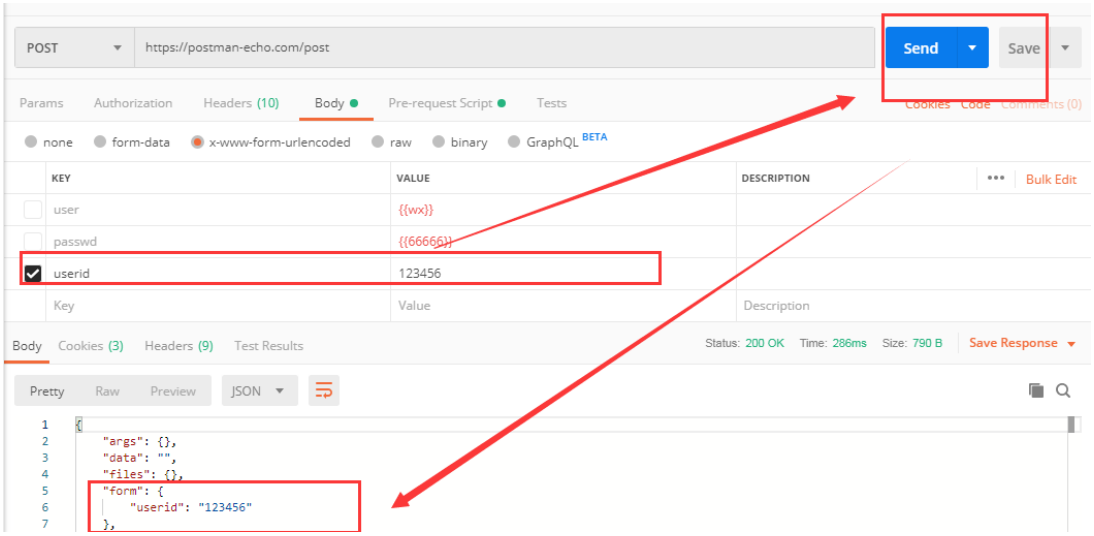
如何设置：在 body 和 test 进行更改

使用场景：a 接口的返回值，是 b 接口的请求参数

注意：当环境变量和全局变量名称相同时，环境变量会覆盖全局变量

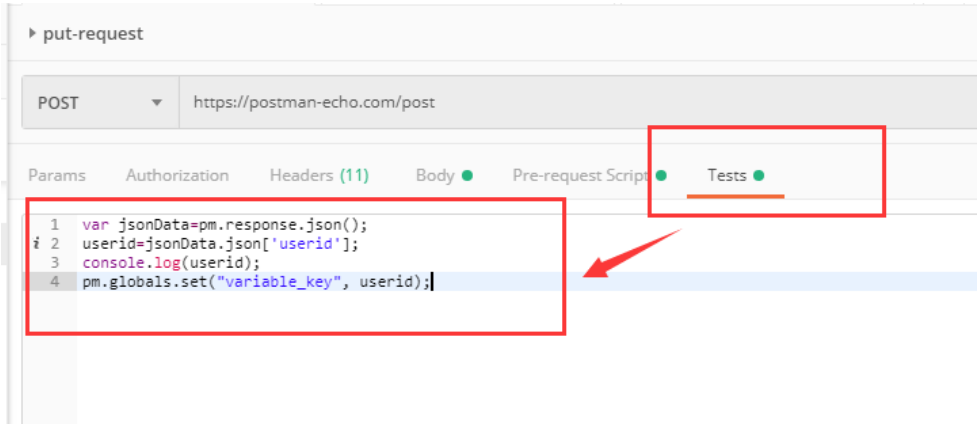
案例二：全局变量的使用

步骤 1.在 body-x-www-form-urlencoded 下填写 key 及 value 值

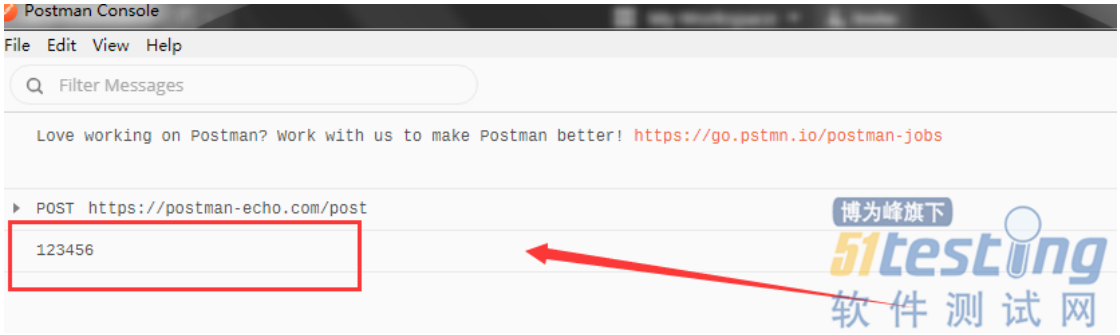


步骤 2.test 标签下编写脚本获得 userid 返回值

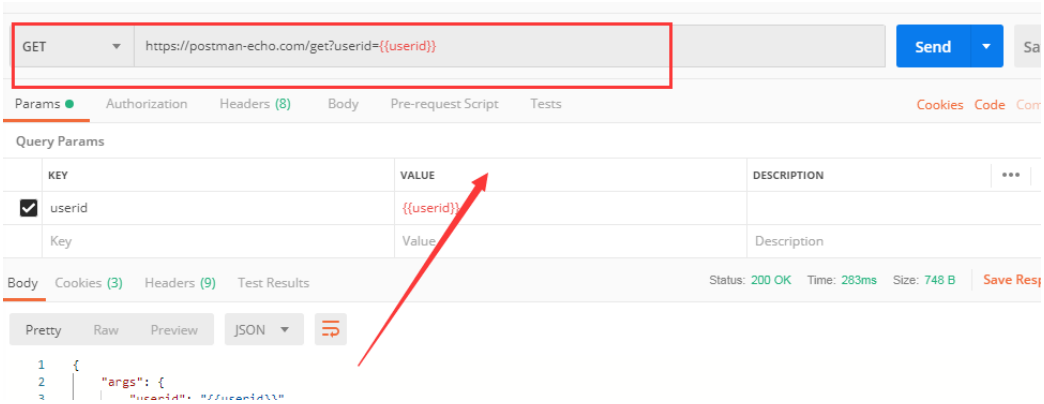
```
var jsonData=pm.response.json();
userid=jsonData.json['userid'];
console.log(userid);
pm.globals.set("userid ", userid);//设置全局变量 userid
```



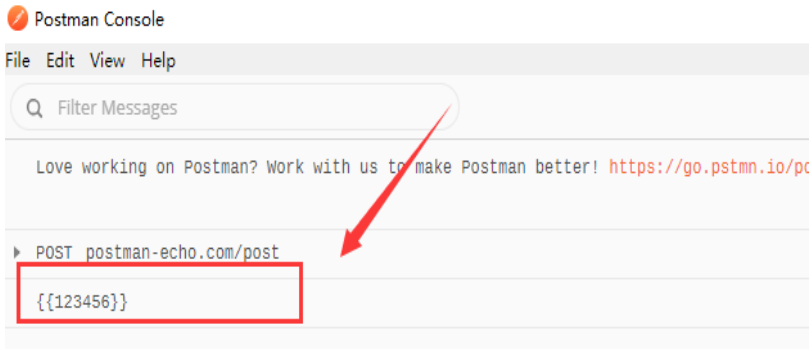
步骤 3.点击 send，在 console 中查看返回结果



步骤 4.B 接口调用 A 接口查看返回值，新建 get 请求



步骤 5.发送 send 按钮，console 查看请求结果



3、环境变量

环境变量：可以理解为不同的环境需要有不同的变量，比如生产环境和测试环境，2 套环境需要的变量也不相同

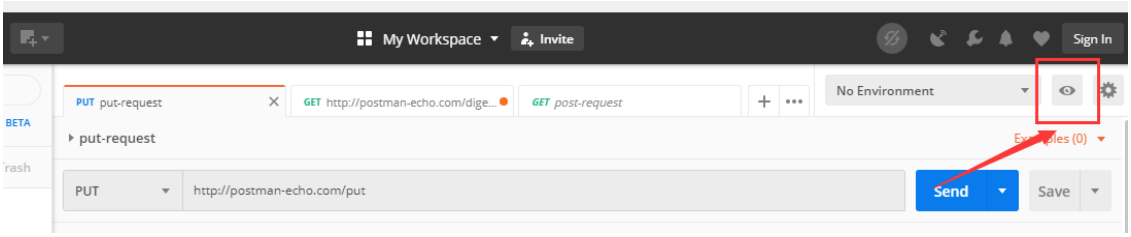
如何设置：在 host 中进行设置，点击右上角的眼睛图标可以设置环境变量和全局变量

使用场景：生产环境 host：postman-echo.com

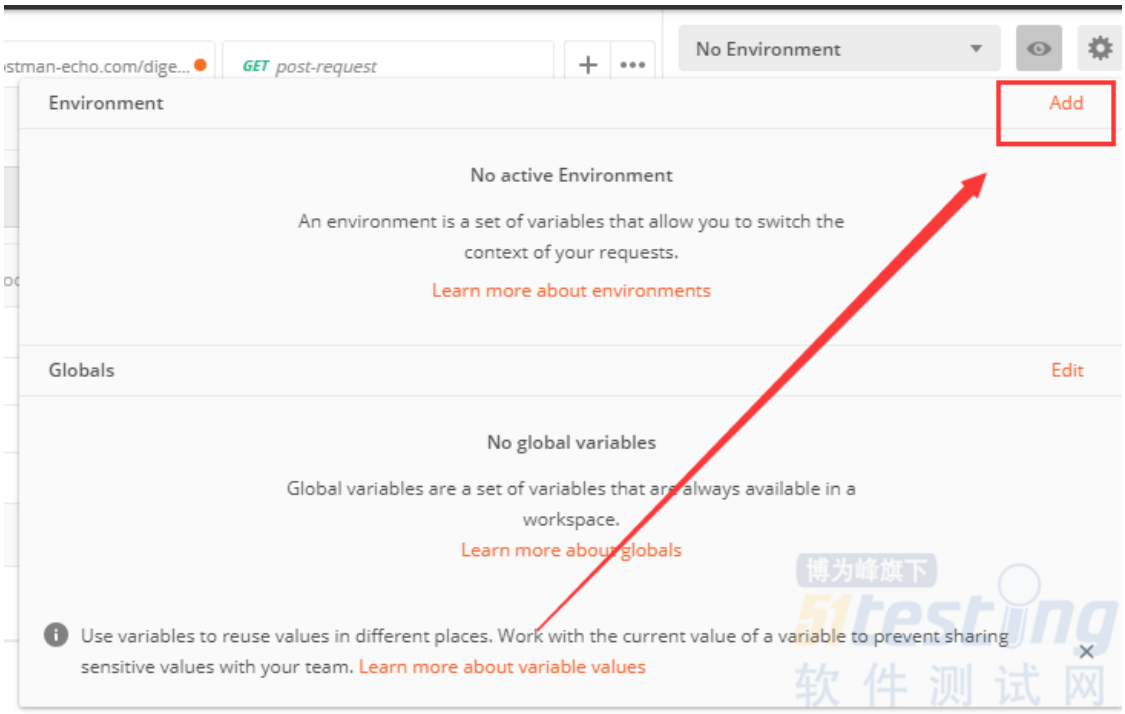
测试环境 host：dev.postman.com

案例三

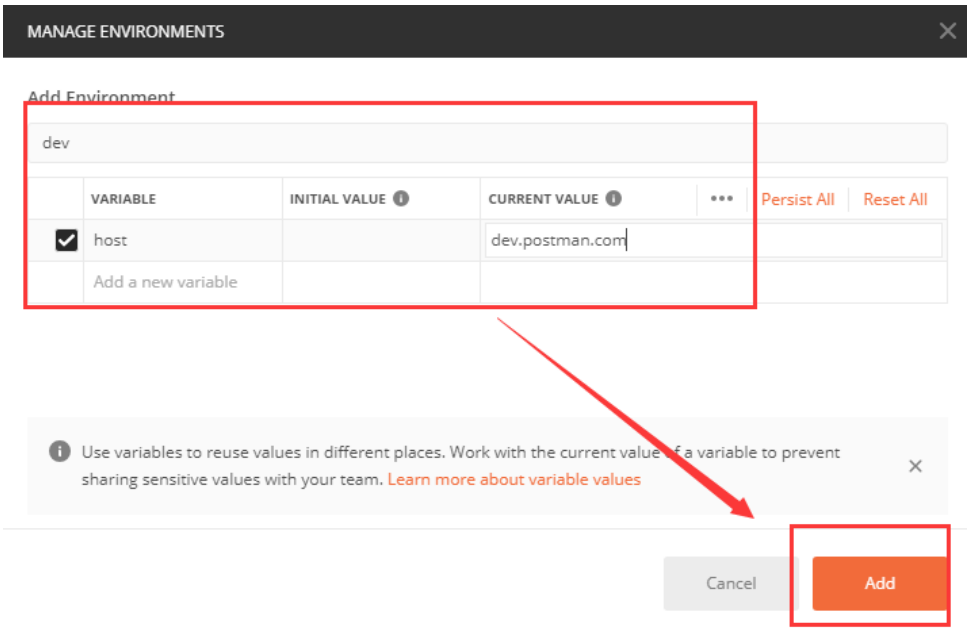
步骤 1.点击眼睛图标



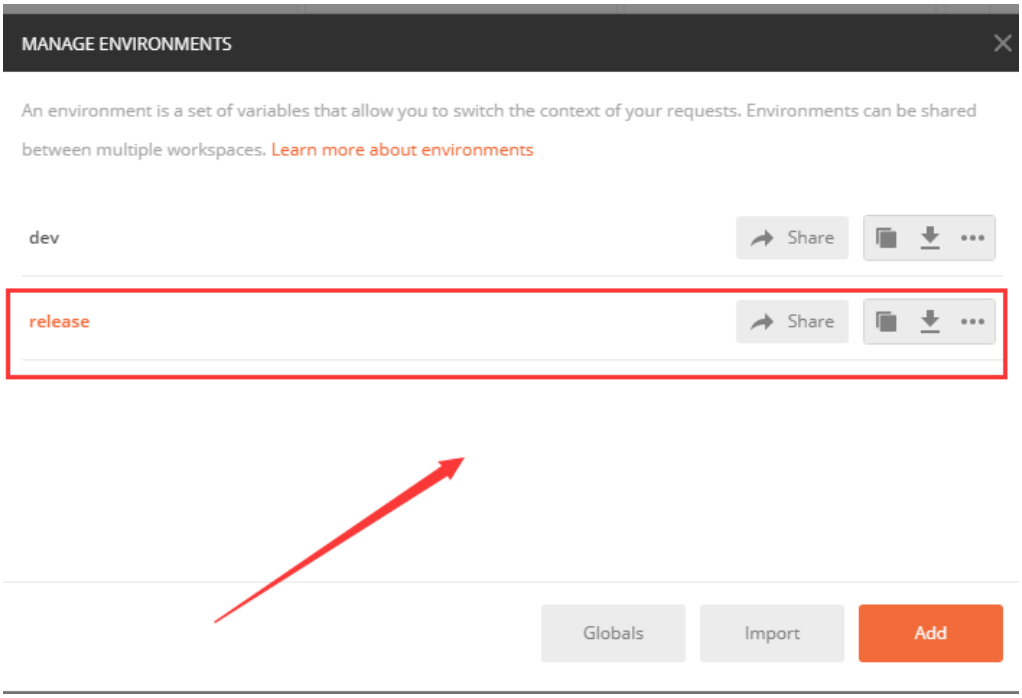
步骤 2.点击 add 添加按钮



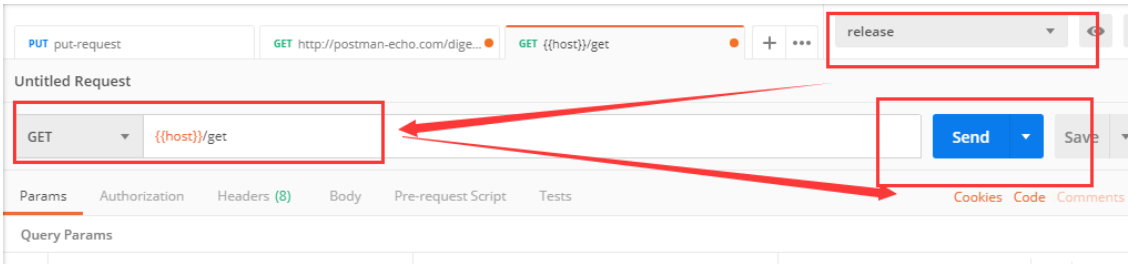
步骤 3.输入环境变量名称和值，点击 add 添加按钮



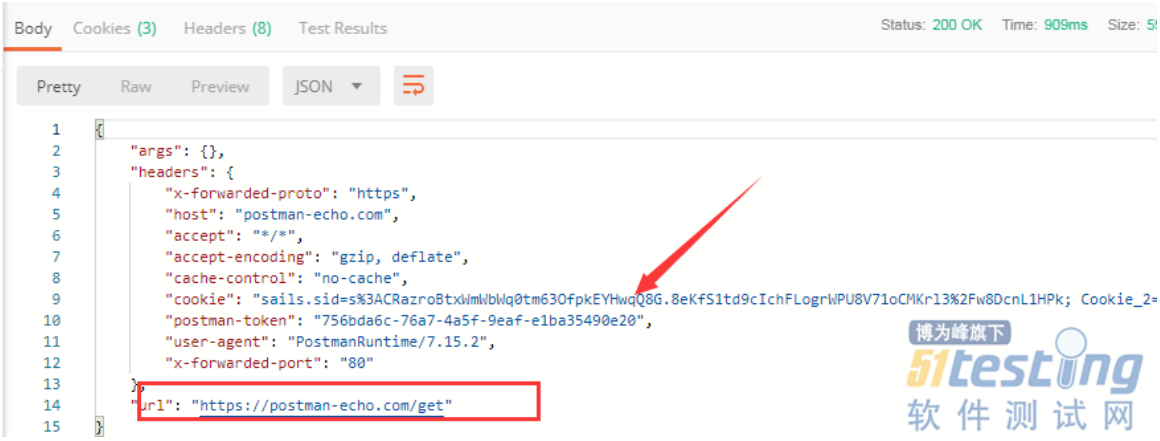
步骤 4.执行步骤 3 添加生产环境变量 post



步骤 5.environment 选择环境变量如 release，ur 填写 {{host}}/get，点击 send 按钮



步骤 6.查看执行结果，具有 relese 环境地址



4、数据变量

数据变量：通过导入外部文件（json 或 csv 文件）来获取变量

如何设置：在 collection 中设置

使用场景：创建 data.json 文件，配合数据驱动功能使用

案例四：参见 postman 数据驱动内容

三、Postman collection runner 的使用

1、Postman 断言

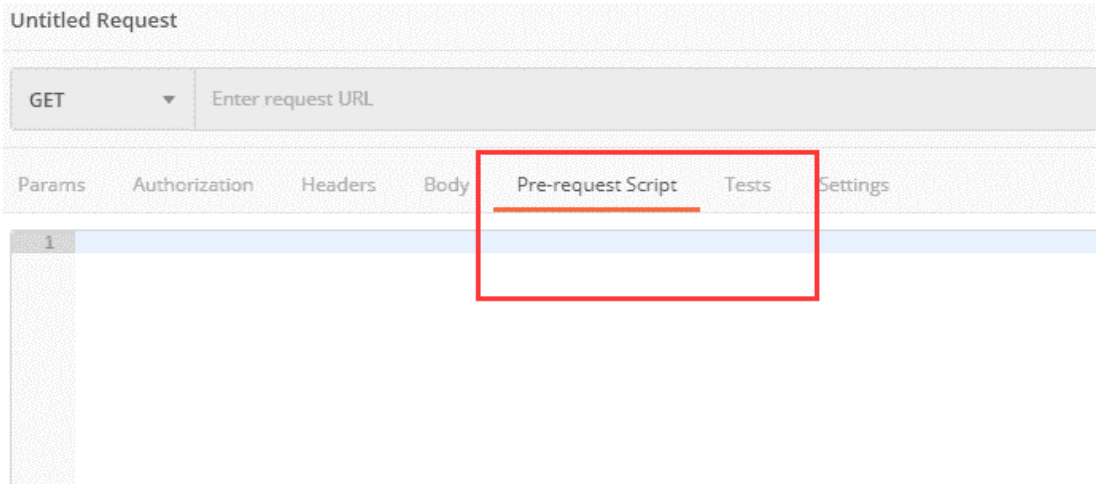
什么是断言：判断程序执行结果是否符合我们的预期就是断言

接口测试根据什么判定断言：一般根据程序返回状态码和响应返回数据

postman 在什么地方设置断言：在 pre-request script 和 tests 内设置

pre-request script（预置脚本）：在请求发送之前执行

tests script：在接到响应之后执行脚本



案例五：如何设置断言过程

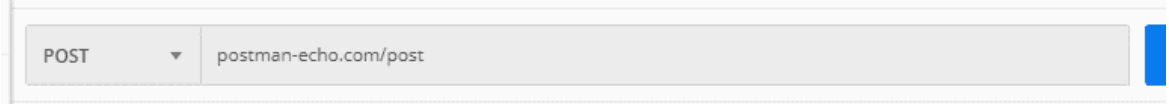
Url 地址：postman-echo.com/post

断言规则：

- 相应状态码：200
- 相应内容：返回的 user 参数值和定义的一致
- 相应时间：小于 0.5s

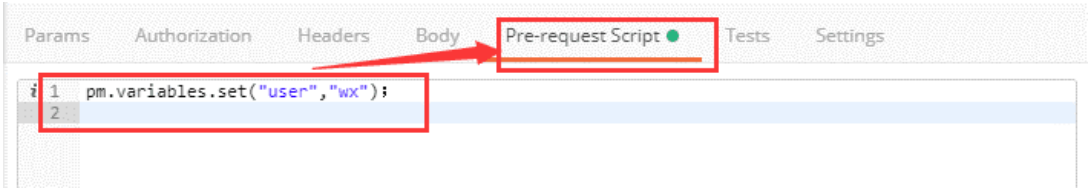
脚本设置过程：

步骤 1.地址栏输入地址

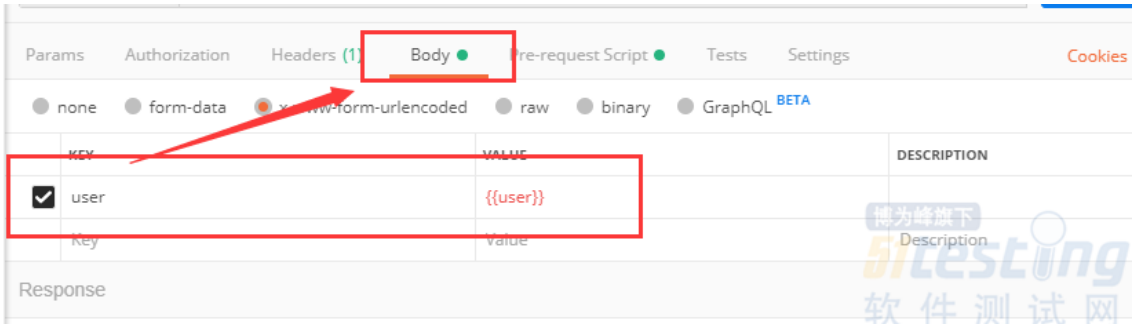


地址栏输入地址

步骤 2.pre-request script 内定义变量



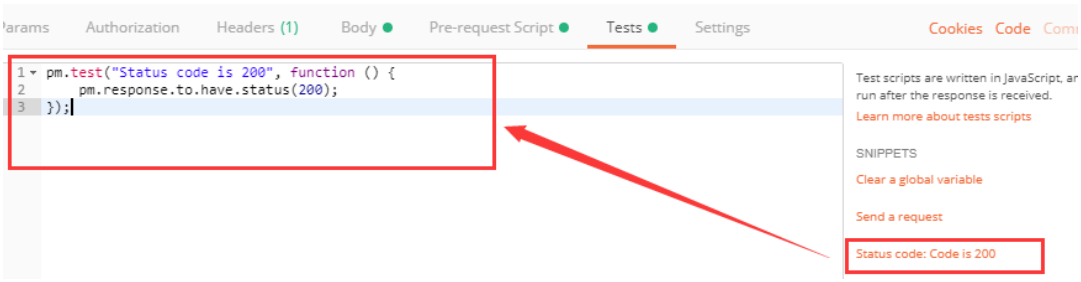
步骤 3.body 下定义本地变量



步骤 4.test 下设置断言的处理

- 相应状态码：200

点击右侧 code is 200 在编辑区域会自动生成相应代码



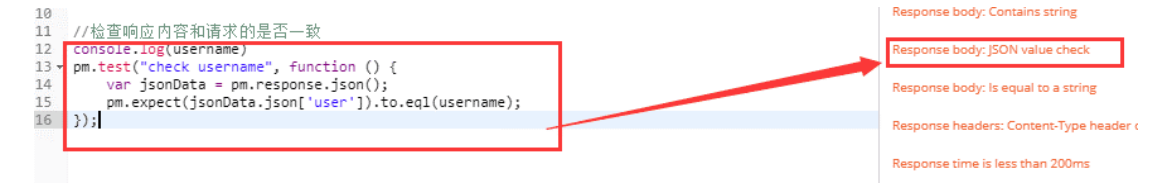
- 相应内容：返回的 user 参数值和定义的一致

获取参数值

点击 get an environment variable ，再修改里面的变量值为 user

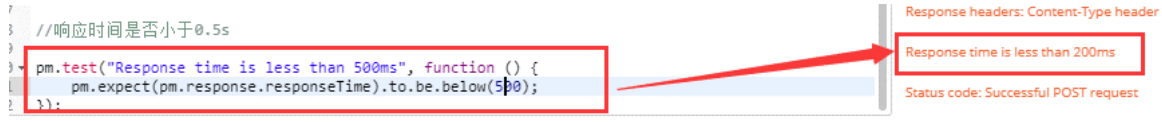
- 检查响应内容和请求的是否一致

点击 response body: json value check，修改相应参数值



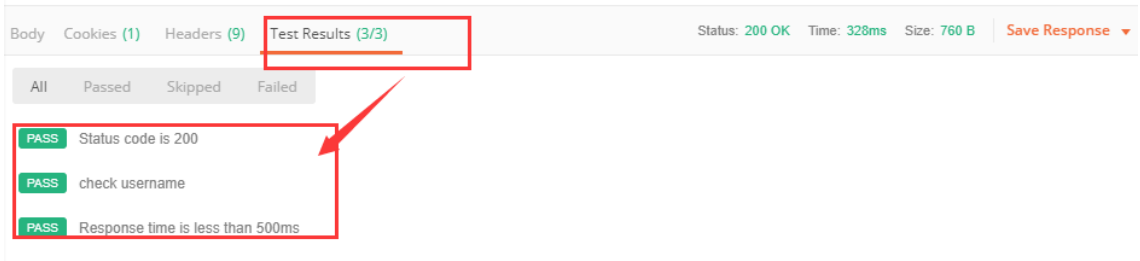
- 相应时间：小于 0.5s

点击 response time is less tahn 200ms，修改相应参数值为 500



步骤 5.运行脚本

在 test results 下查看运行结果



脚本代码:

```
pre-request script ：
pm.variables.set("user",'zxw');
test script:
//判断响应状态码
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});
//获取参数值
username=pm.variables.get("user");
```

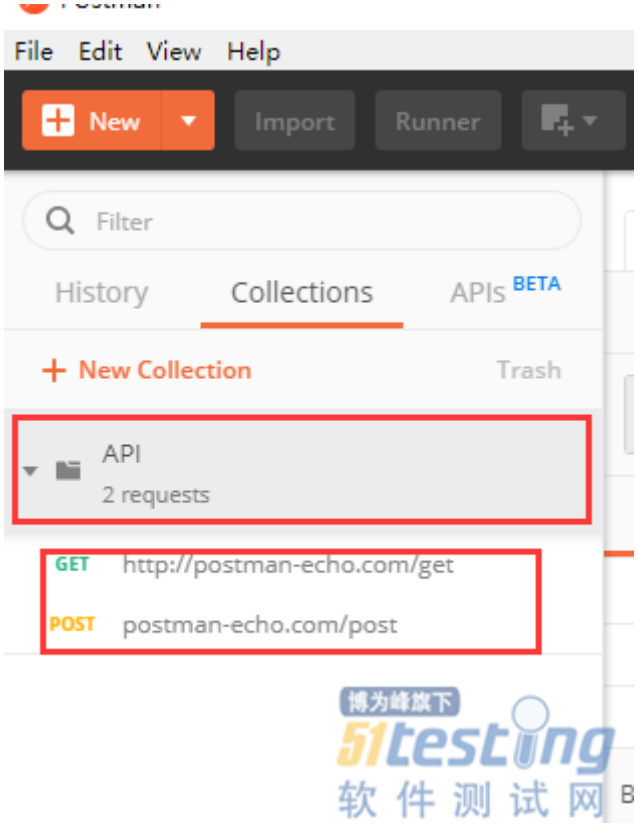
```
//检查响应内容和请求的是否一致
console.log(username)
pm.test("check username", function () {
var jsonData = pm.response.json();
pm.expect(jsonData.json['user']).to.eql(username);
});
//响应时间是否小于 0.5s
pm.test("Response time is less than 500ms", function () {
pm.expect(pm.response.responseTime).to.be.below(500);
});
```

2.postman 批量执行

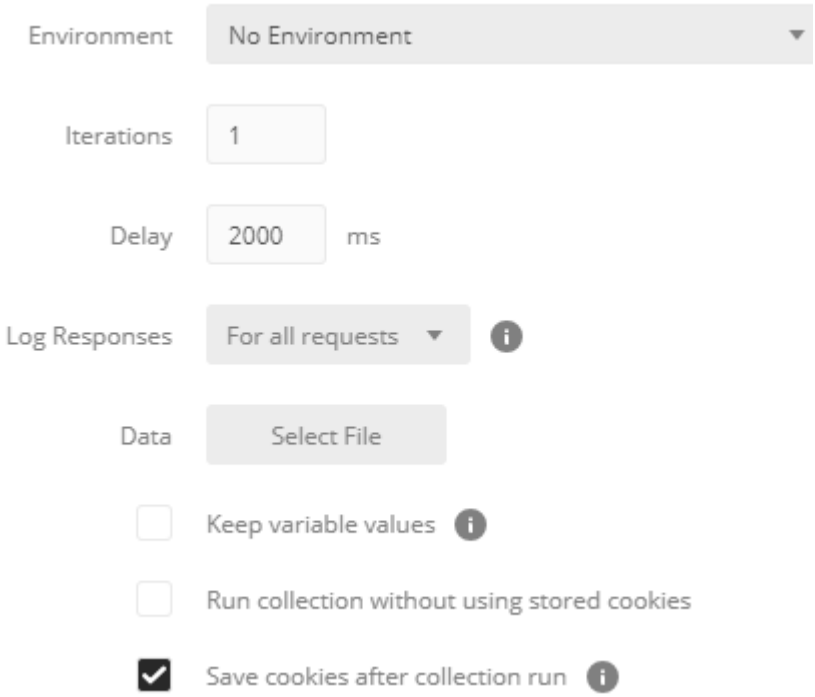
适用场景：某个集合有多个 API 时，可以通过在 collection runner 中设置来批量运行所有的 API

案例六：批量执行集合中所有请求

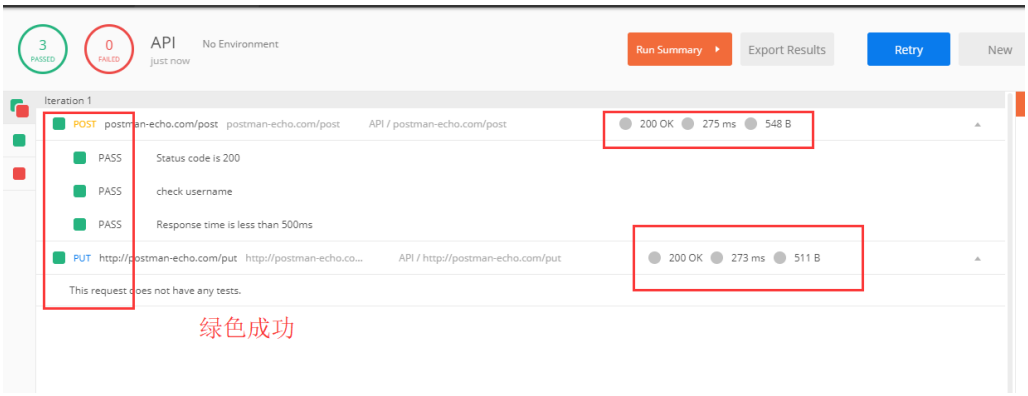
步骤 1.API 集合下有 2 个接口请求



步骤 2.点击标题栏下方 runner 按钮，对运行界面进行设置



步骤 3.Run API



3.postman 数据驱动

测试场景

比如登录接口，需要输入不同的用户名、密码验证登录功能

什么叫数据驱动?

通过导入外部文件对接口进行参数的操作叫做数据驱动

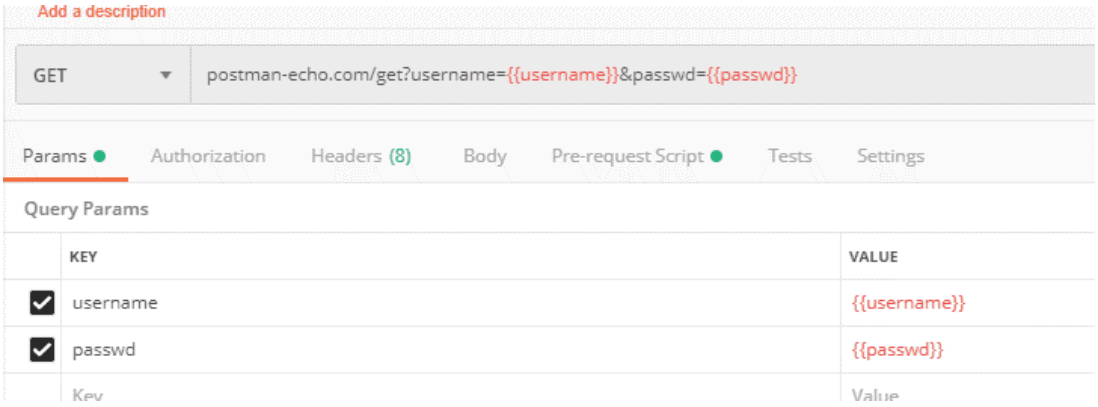
案例七：导入包含用户名、密码的 json 文件

步骤 1：编写好 json 参数文件 data.json

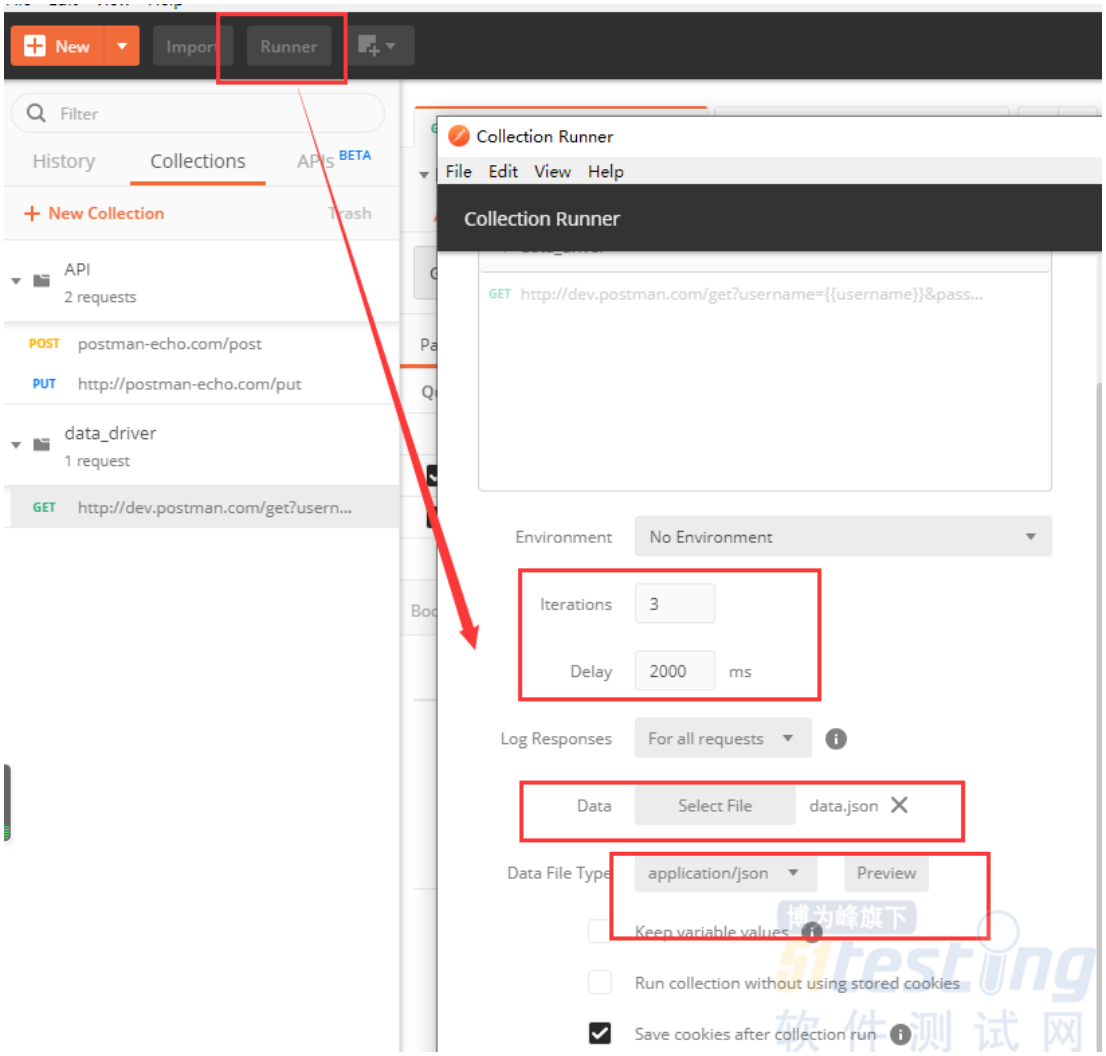
[{

```
    "username":"jack",
    "passwd":"6666"
    },{
        "username":"wx",
         "passwd":"123"
        },{
            "username":"dfsfs",
            "passwd":"55555"
           }
]
```

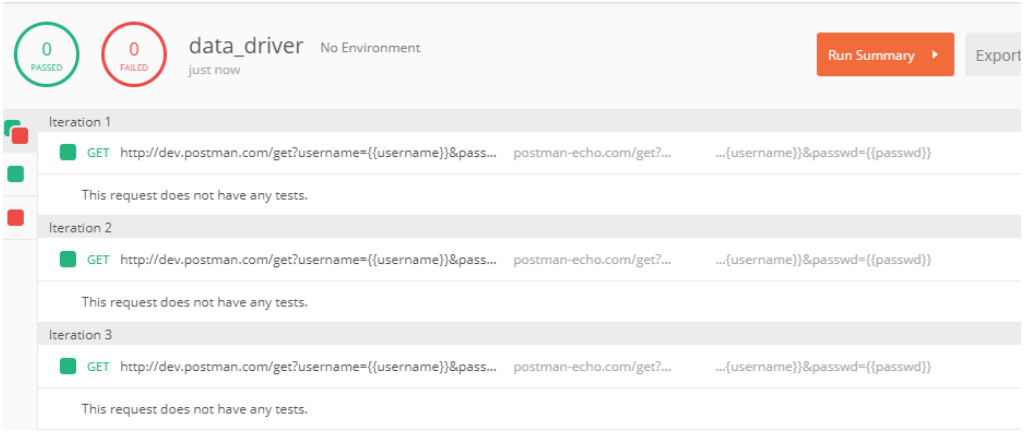
步骤 2：填写路径:postman-echo.com/get?username={{username}}&passwd={{passwd}},点击保存



步骤 3：设置 runner 内容，因为有 json 文件有三个参数，所以迭代次数为 3，时间 2s，选择事先准备好的 data.json 文件

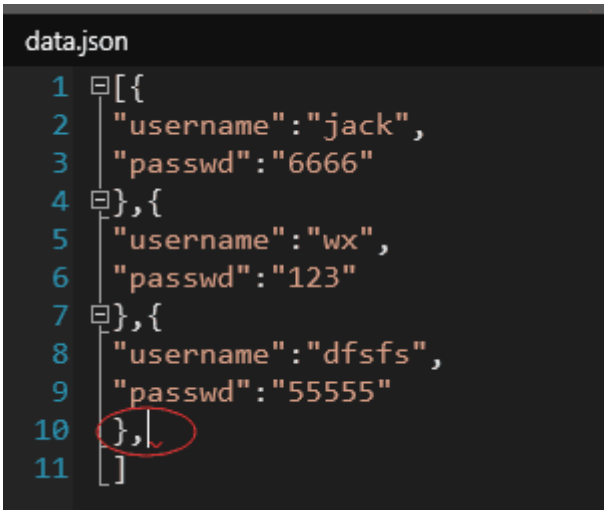


步骤 4：设置完成后点击 run 按钮，查看执行成功结果



错误 1：Error reading data file: SyntaxError: Unexpected token] in JSON at position 128

将 json 文件最后一个，去掉



错误 2：There was an error running your collection: getaddrinfo ENOTFOUND dev.postman.com dev.postman.com:80

出现这个问题的原因很有可能你的 url 接口就没有调通，尝试把 get 后面内容去掉直接调用试一试，调好接口后在进行参数化

4.postman 构建工作流

测试场景

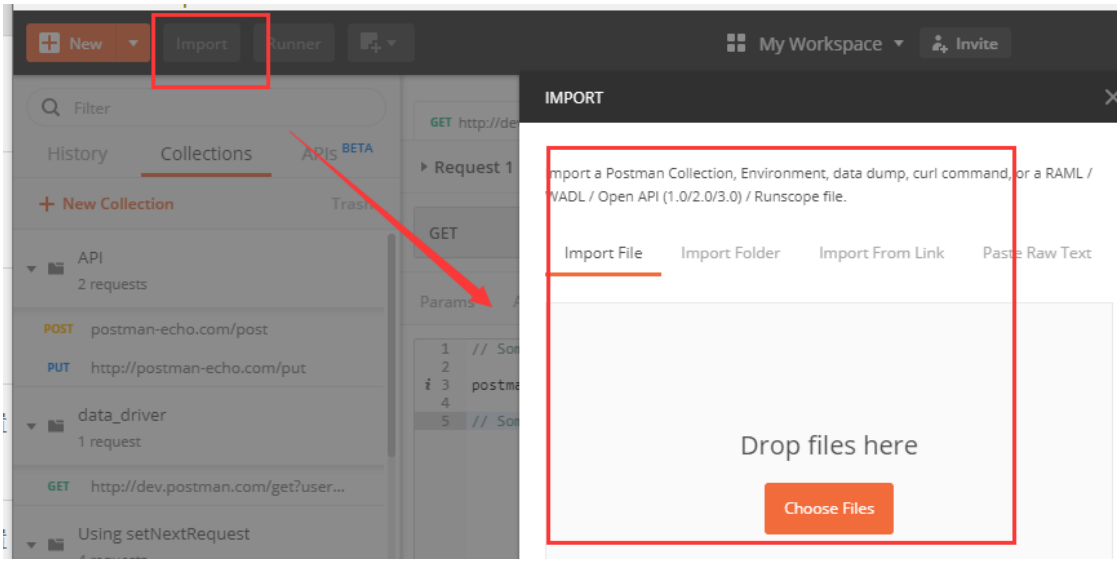
当集合中有很多多个 API 时，默认 collection runner 会按从上到下的顺序执行，我们可以通过构建工作流的方式切换 api 间的执行顺序

设置方法

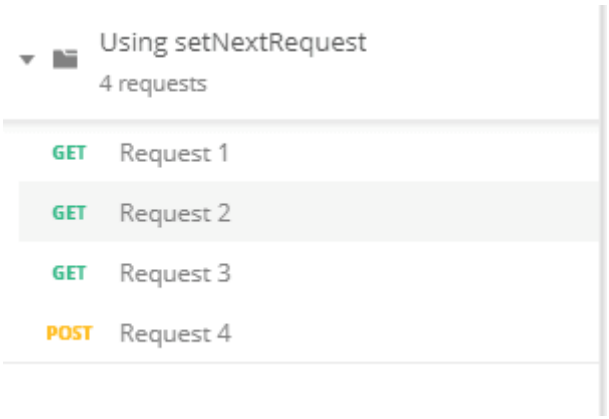
案例八：实现按照 1->4->3->2 顺序执行

步骤一：下载案例文件 collection.json 文件

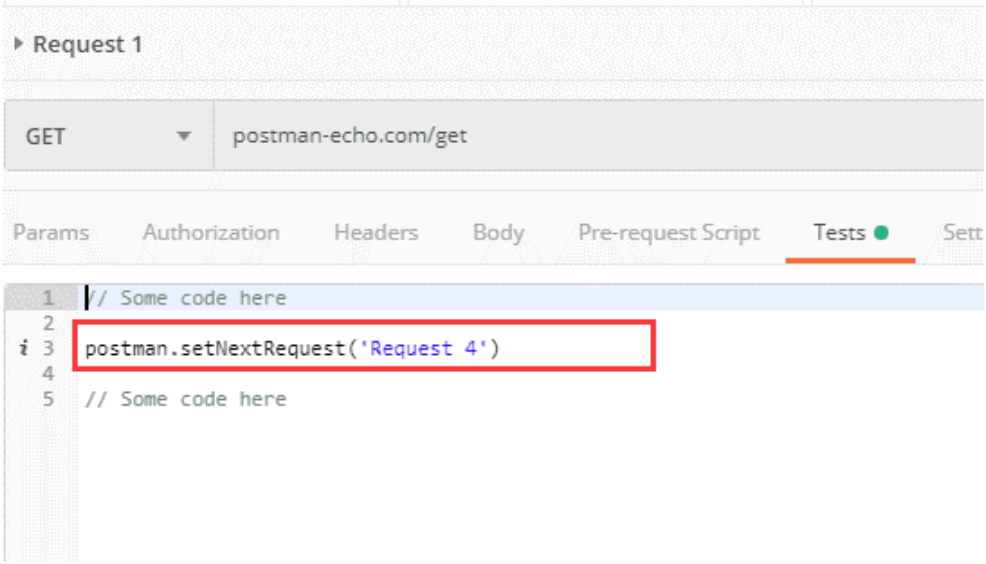
步骤二：将文件导入到 postman



导入后效果:



步骤三：在 request1 test 下填写如下语句，点击 save 保存



postman.setNextRequest('Request 4')

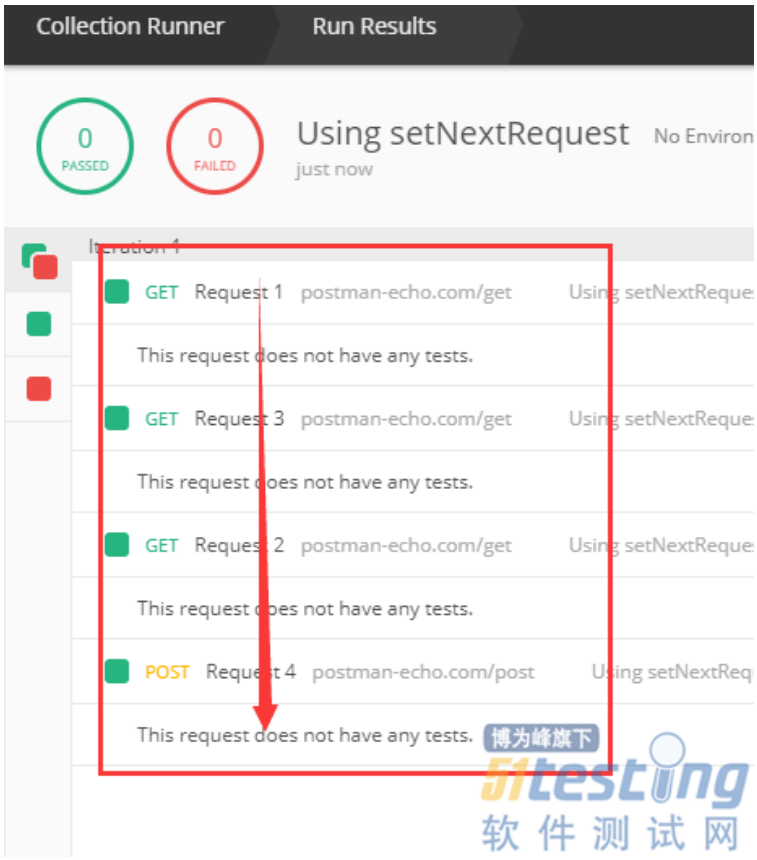
在 request4 test 下填写如下语句

```
postman.setNextRequest('Request 3')
```

在 request3 test 下填写如下语句

```
postman.setNextRequest('Request 2')
```

步骤四： 设置 runner 内容，点击 run 执行，查看执行顺序



注意每次更改完成后都要点击 save 按钮，否则仍按照原来顺序执行

知识点总结

1.本地变量：作用域针对于单个集合

2.全局变量：作用域针对于所有集合

3.环境变量：侧重于不同环境间切换，便于管理不同接口。当环境变量和全局变量名称相同时，环境变量会覆盖全局变量

4.数据变量：主要和数据驱动配合使用，常常通过导入外部文件（json 或 csv 文件）来获取变量

5.断言：我们首先需要知道断言的规则是什么样的才能根据规则设置断言脚本。一般根据程序执行的结果设置断言，比如程序响应的时间是否在 2s 内，程序返回的用户名是否和传入时一致等。至于在 test 编辑区域如何编写，我们可以通过点击的命令行获取响应的代码

6.批量执行：在 collection runner 界面进行设置，注意 pass 个数为执行断言的个数

7.数据驱动：配合数据变量联合使用，导入外部文件。在 collection runner 界面基本上设置迭代次数、时间、选择外部文件、及文件格式就可以了。

8.构建工作流：也是在 collection runner 界面操作，其中主要不同于 test 代码编辑区域使用命令 postman.setNextRequest('Request 3')指定下一个执行文件是哪一个。注意每修改一次都要保存文件再执行。

《51 测试天地》(五十五) 上篇 精彩预览

- 测试女巫紧跟时代脉搏 AI 之三种聊天机器人
- 基于三层结构的自动化测试数据准备技术研究
- 安卓 app 兼容性测试之 targetSdkVersion
- 你是如何汇报工作的？——软件测试的价值论述
- 高效的移动应用程序测试项目的三大策略
- API 测试该了解的一些技术细节
- Python 多线程在爬虫中的应用
- 一个好测试，首先得是一个好管理
- 银行线上信贷系统接口自动化测试探索

马上阅读