目 录

(五十期下)

为了可持续的测试自动化，透过表面看本质.....01

使用 Jmeter 进行性能测试实战.....06

测试工程师的捉虫历险记.....35

项目经理眼中的项目计划.....46

CentOS 上 nginx 服务器安装 phpmyadmin.....50

有了 XPath，再也不用担心元素定位了.....52

为了可持续的测试自动化，透过表面看本质

◆ 作者：zhengchenhappy

当提到可接受的测试自动化，最重要的一步是在适当的位置有一个适当的测试自动化团队框架。这篇文章对一些不同的自动化测试适用场景有一些已证明的项目——由一个自动化或者回归团队主导，以敏捷的适应性——帮助组织享受长期的测试自动化的成功。

公司发起一项新的测试自动化倡议——任何销售人员设计销售相关的工具——倾向认为他们的成功取决于完美的上线。作为一个测试自动化顾问，我喜欢提供一个真实的基于我在领域里所见的检查。如果你没有准备好，最初的上线可能是坎坷崎岖的路，但是在长期中，那不是要制造而是破坏你的测试自动化倡议。

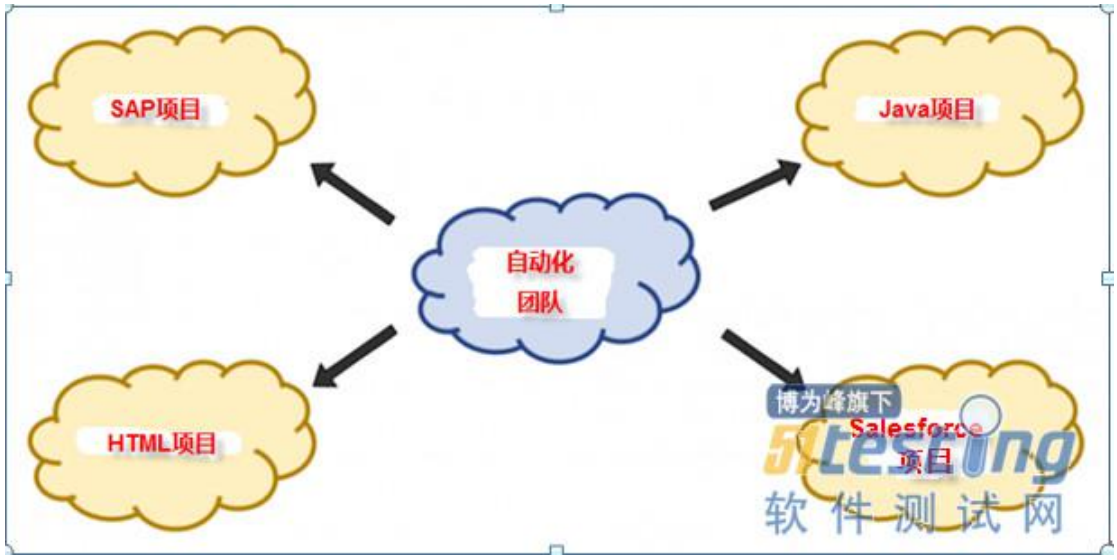
近来我亲眼看到一些公司没有准备就开始出现测试自动化。执行这个“策略”的风险是测试者们可能要因为这些原因坚持测试自动化：

- 他们觉得没有它生活得很好
- 他们不想要（或没有时间）学习新工具
- 他们担心它太复杂而且需要获取跟上进度的努力超过了感知到的好处

一个不充分的计划——或者完全不计划——上线很明显不是最好的发起一个测试自动化倡议的最好方法。无论如何，我发现了从坚固的上线中恢复比没有一个恰当的测试自动化退队结构而去获得持续的测试自动化来得简单些。

我喜欢去为一些不同的测试自动化采用场景去分享一些已证明的实践，帮助了很多机构享受长期的成功——甚至当最初的上线不理想时。

自动化团队正一个接一个项目地上线我们的测试自动化



我能提供的最好一条建议是从小事开始，然后做大的。已证明的第一个实践是从创建自动化团队开始。

这个团队将从不同的项目迭代并且开始从最重要的测试用例起自动化。这让你证明在倡议中的进度并且让你看到了一些应用程序的改变如何影响主应用程序功能性。

我建议从最有影响力的测试用例开始：覆盖你的头等业务风险的那些。这创造了强有力的回归文件夹的建立，它提供了最快的应用程序变化是否先破坏了正在工作的功能的反馈。

自动化团队应该包含至少一位测试设计专家，他通过回顾需求并使用测试用例设计方法制定出应该被自动化的测试用例。然后，测试设计专家或者两个或三个自动化专家中的一位能评估是否需要测试数据管理，然后他们能合作指定测试用例。同时，测试设计专家将关注下一个项目。

取决于你想要测试的软件以及你定制它多少，一个自动化工程师是必要的。他们将确保常规的控制被创建然后被维护。

团队的大小将会随着扩大，取决于测试用例和团队处理的项目的数量。这些建议只是为了最初的上线。一旦上线开始进行，你们将会想开始招募尽可能多的人员。

一旦测试用例被自动化，他们能通宵在虚拟机上运行，然后在日常基础上报告执行结果。为使测试自动化加入与运转的最重要的事情是确实在执行测试用例！假如你不是正执行你的用例，你不会从它们得到任何价值，不论你的测试集设计得如何好并且它如何好地覆盖你的主要业务风险。

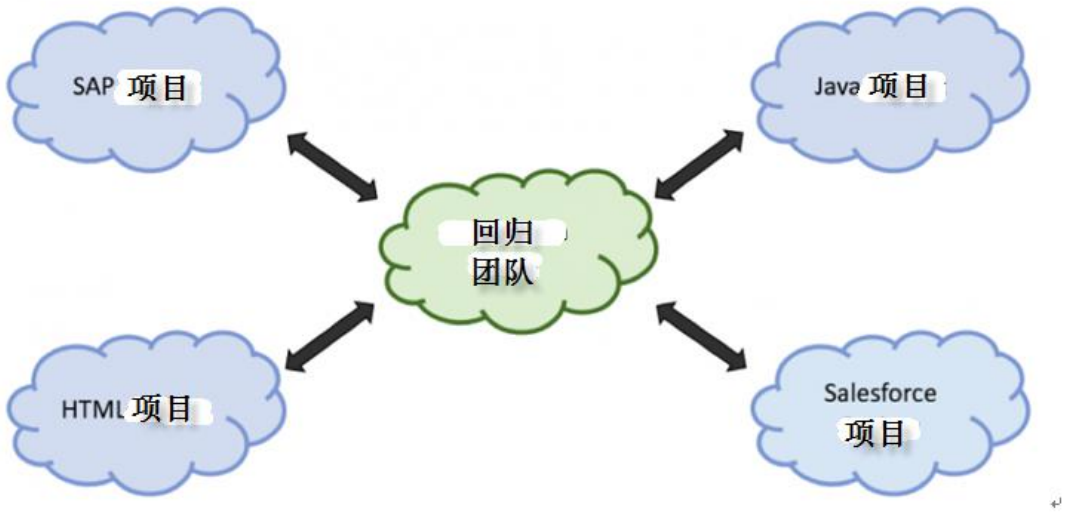
取决于你的公司架构（敏捷或不是，测试团队或者独立的测试员们），自动化团队可能主要以教导每个人关于上线自动化最佳实践的实施以及当他们开始他们自己的测试自动化时的需要考虑的项目多样性为主。

在测试人员开始采用测试自动化之前，必须做个决定：谁负责这些测试用例？这是主要的。假如自动化团队（或者后来回归团队）是负责的，然后他们不得不与测试人员

们和商务方面做更多的沟通。假如它是测试员们的的责任，他们需要足够的时间去保证所有的测试用例在执行并且符合业务需求。每一个组织需要去决定什么是对他们最好的，取决于他们的架构和工作模式。

不管你负责哪个方向，确保节约足够的时间来沟通。一方面，新的测试用例需要被回归团队审核然后为执行而创建。另一方面，回归测试结果（尤其失败的测试用例）需要被共享使得负责的团队成员或者测试员们能审查它们并必要时更新测试。至少，确保测试员们独立地检查回归结果并做合适的调整，这样回归团队能从共享文件夹里推出新的测试用例，自己审核它们。

回归团队在每个项目上领导和指导测试员们



我所见过的最有效率的设置是建立一支初始的自动化团队，它包含一个回归团队，一旦测试自动化基金会被成立，其他就要准备驱动它向前。

回归团队仍然为在最初的项目里创建测试用例负责，维护存在的测试用例，执行测试用例，保护测试架构，并且监督拓宽的测试自动化建立和项目。你能把这个团队叫做优秀测试中心。

关于这个建立，可能有项目和更大的改变由测试设计专家指导着，他关注正确的格式并提供它应该看起来像什么的预览。他们就像软件艺术家（或者以我们为例，测试艺术家）。

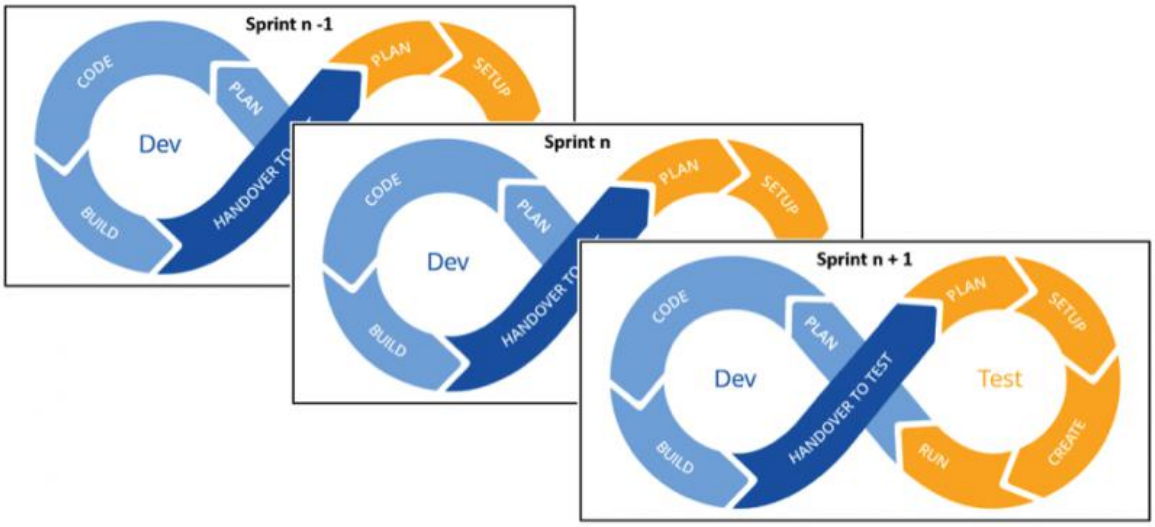
回归团队是“定位群”，假如你有关于问题的疑问或者你想要引进新的创新。所有者和责任应该应该是回归团队的，因为他们有测试文件夹的最好概览。

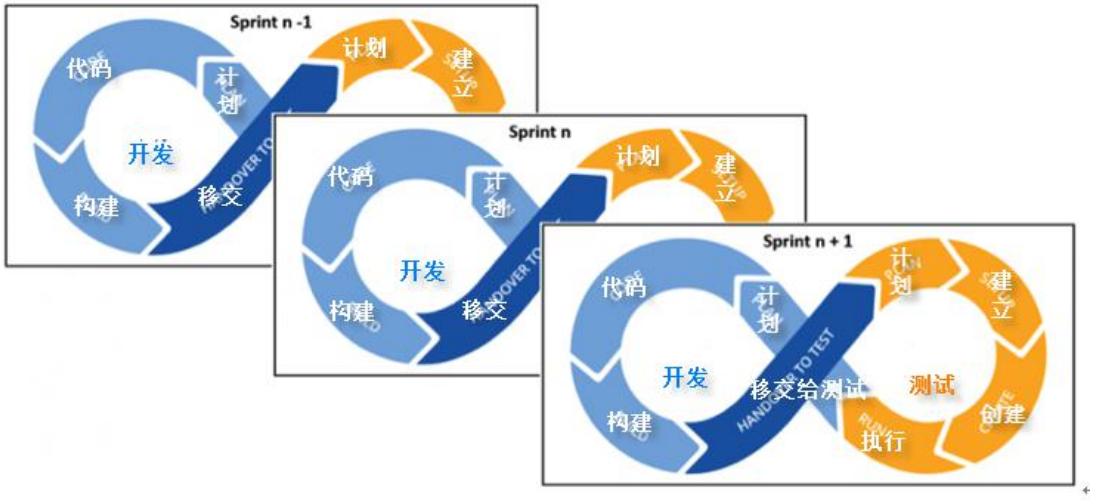
从一个资源的角度，你在回归团队里需要高技能的人，但是你能侥幸逃离经验少的作为测试员的团队成员。假如测试员们需要比培训提供的更多的知识，他们能经常问回归团队，他们然后能通过共享他们建立起的知识提供帮助。

敏捷环境的适应

对于敏捷团队，测试自动化被团队里的敏捷测试员驱动。这个人应该比一个自动化专家更先进——他们需要理解测试用例设计、需求、测试用例创作和执行，以及核心的测试最佳实践，并且他们需要一个测试什么以及如何有效测试它的好的理解。

我建议一个开发和测试从一个 Sprint 到另一个 Sprint 持续的建立：





当开发结束了 Sprint n-1 的特性，测试开始。同时，开发能持续特性 2（Sprint n），等等。只是确保为修复 bug 和更深入改进保留时间。

我们分离开发和测试吗？

最后，当谈到架构，我想要给频繁发生的一对问题增加我的两分钱：我们的开发需要测试吗，或者我们需要分离开发和测试吗？我们在测试中使用软件开发工程师（简称 SDETs）吗？

理论上，你也可以通过有开发测试来节约时间和金钱因为开发最知道代码。但是实话实说：开发从没有足够的时间去完成每个人想要完成的开发任务。假如他们也为测试

自动化负责，那意味着他们甚至没有时间去完成那些任务。而且即使开发们最知道代码，你确定他们能检查边界条件吗？预计风险并尝试询问合适的问题？探究系统并把它推向失败？

我的意见是有些人不写代码更适合做这个。那些其他人关注于通过应用程序获取实际的业务目标并尝试思考每一个能导向系统一个错误的指导的可能性。从我所在领域里见过的，那通常不是一个开发的强项。

旋转桌子

所有这些建议是基于我的帮助用户适应测试自动化的个人经验。我知道每个读者可能有关于这的不同观点——而且我鼓励你在下面去分享关于它的想法。什么帮助你最多了？你如何在你的公司建立测试自动化？你面对的挑战是什么？

使用 Jmeter 进行性能测试实战

◆ 作者：晴空

一：实施背景

1.1 任务目标

钉钉方对此次压测任务解释如下：

1.稳定性专项的目的是希望和服务商一起保障服务稳定可用，为用户提供更可靠的服务；主要包括以下内容：

2.系统架构，包括服务器、数据、容器等

3.监控配置，报告系统监控（cpu、内存等）和业务监控（QPS、RT 等）

4.首页性能情况

5.压测情况

6.降级方案，用于流程突增时，系统及时处理保证服务运行

（1）提供单链路或全链路压测数据及最高支持的 QPS 上限，各业务场景下列各项值：

已上架	查看监控线上最高 QPS 为 xxx(可查询近 2 个月最高 QPS 等)	要求压测至少支持： 峰值 QPS 的 3~5 倍

a.峰值 RT<=100ms

b.峰值错误率(统计非 200)<=0.1%

c.峰值总 cpu 利用率<=70%

d.峰值 load1<=cpu 总核数-0.5

e.峰值内存利用率<=80%

（2）ISV 服务端授权激活场景，服务端响应时间整体小于 3s

这样一来，我们的目标很明显，就是验证 QPS 是日常峰值 3~5 倍情况下，服务器的资源占用，RT，QPS 是否满足要求。

1.2 调研选型

由于时间仓促，我在选型压测工具时只对比了自己比较熟悉的工具 Jmeter 和 LR，而 LR 只能使用破解版的(HP 和微软这些厂商很鸡贼的，你想用破解版的那就用吧，等把你养肥了啃你一口大的，他们的策略也很套路 就是广撒网，总有那么几家会上钩的，对吧)，并且钉钉方明确建议使用 Jmeter，那就没啥好说的了，直接上 Jmeter 吧。

二：环境配置&安装目录解释

2.1 Windows 环境下安装 Jmeter

Step1：配置 Java 开发环境，话不多说。

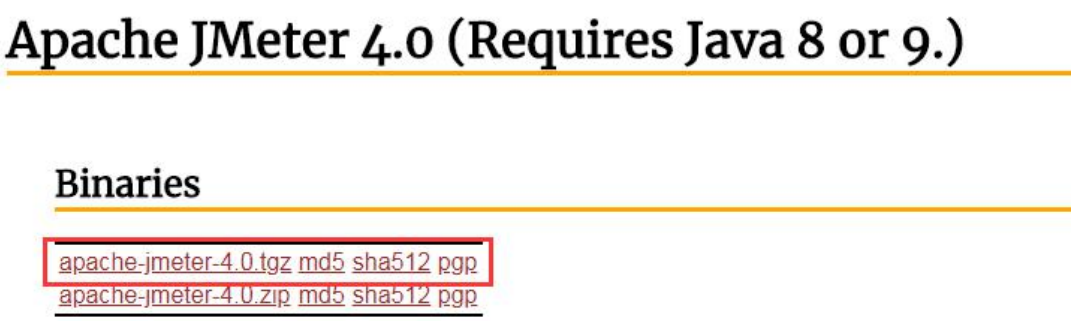
Step2：下载 Jmeter：https://jmeter.apache.org/download_jmeter.cgi

解压后设置环境变量 JMETER_HOME(设置为 Jmeter 的安装目录)

2.2 Mac 下配置 Jmeter

Step1：配置 Java 开发环境。

Step2：下载 Jmeter 并解压。



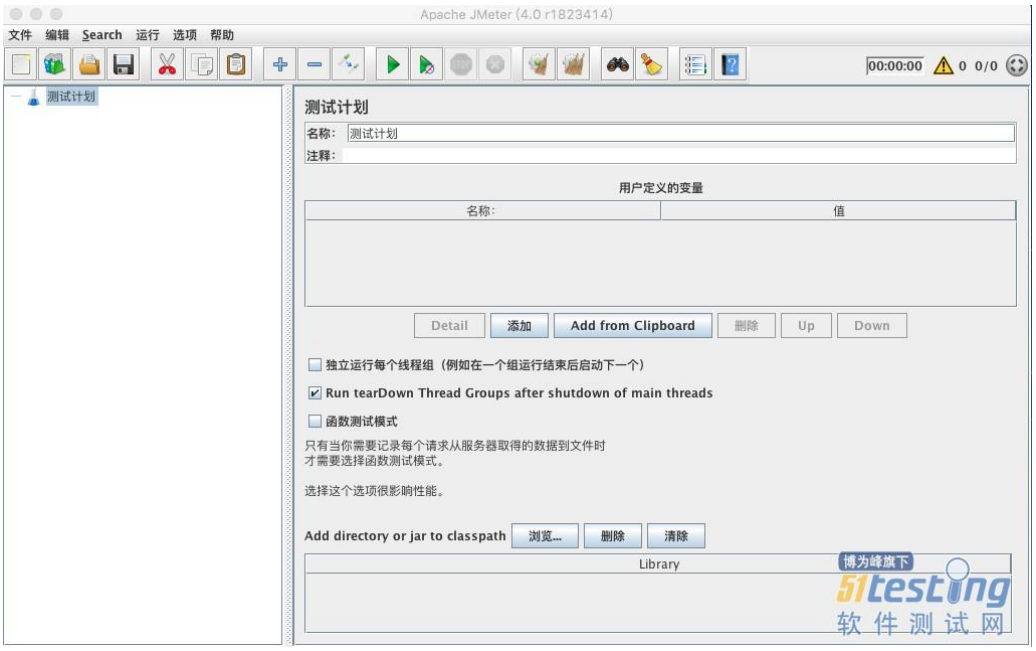
Step3：配置 bash_profile(切换到当前用户主目录即:cd，然后查看隐藏文件 ls -al 即可看到这个文件，若无，自己建)，加入以下内容。

export JMETER_HOME=/Users/mc/Applications/apache-jmeter
PATH=$JMETER_HOME/bin:$PATH:.

export PATH

然后执行，source .bash_profile 即可，我们输入 jmeter.sh 验证下结果如下：

```
mcdeMacBook-Pro:~ mc$ vi .bash_profile
mcdeMacBook-Pro:~ mc$ source .bash_profile
mcdeMacBook-Pro:~ mc$ jmeter.sh
================================================================================
Don't use GUI mode for load testing !, only for Test creation and Test debugging.
For load testing, use NON GUI Mode:
   jmeter -n -t [jmx file] -l [results file] -e -o [Path to web report folder]
& increase Java Heap to meet your test requirements:
   Modify current env variable HEAP="-Xms1g -Xmx1g -XX:MaxMetaspaceSize=256m" in the jmeter batch file
Check : https://jmeter.apache.org/usermanual/best-practices.html
================================================================================
```



2.3 目录解释

bin：可执行文件目录。以下文件是我们经常会用到的。

jmeter.bat 可以设置 jemeter 使用的内存(ps：建议配置为负载机物理内存的 1/4~1/2)

jmeter.sh：Linux 和 Mac 下启动 Jmeter GUI

jmeter-server(.bat)：jmeter 联系负载设置文件。

jmeter.properties：jmeter 的 80%以上的配置项均在该文件中配置；一旦该配置文件被改动，只有重启 jmeter 才生效。

docs：jmeter API 文档

extras：Jmeter 拓展。

lib： jmeter 启动时默认的 classpath。使用 jmeter 进行测试时所有需要 import 的包和

类必须存在该目录下。

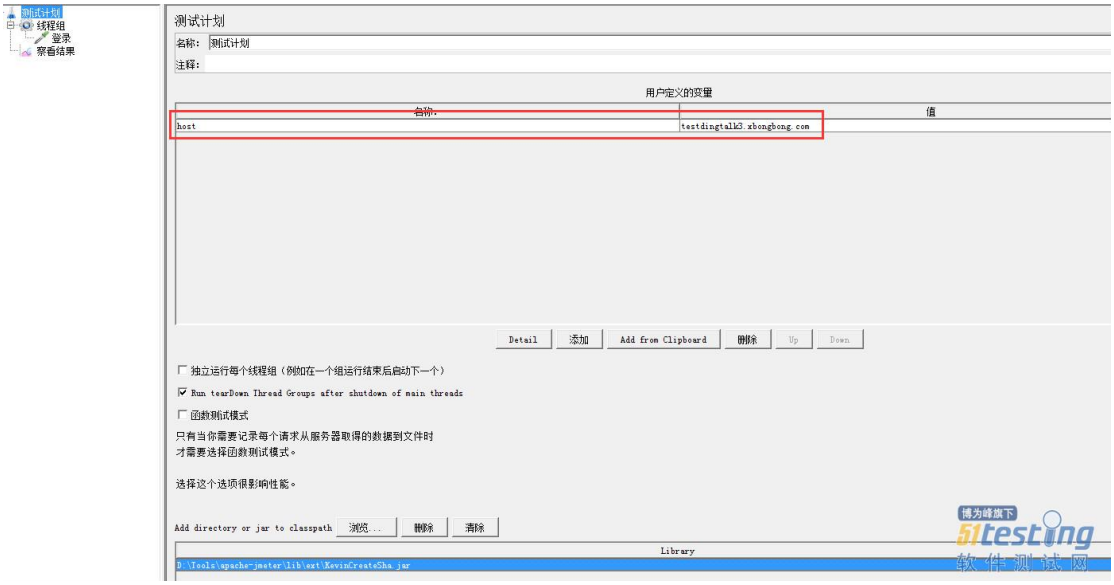
lib/ext：存放 jmeter 的组件/插件，第三方组件和插件也要放置在该目录下。

所有图形化 GUI 中可见，可使用的部分必须放置在 lib/ext 目录下。

printable_docs：jmeter 官方帮忙文档。

三：Jmeter 常用的元件

3.1 测试计划



测试计划中可以做以下事情：

1：定义全局变量 2：控制线程组的执行方式 3：引入外部拓展的 jar 包。

如图：定义全局变量 host，后续的请求中可以通过${host}来使用这个变量值。

最下方蓝色框中是需要依赖的外部 jar 包，这个包是生成 sha-256 加密字符串用的，后续在 bean shell 前置处理器那里我会详细说明。

注意：对于<独立运行每个线程组>这个选项，如果一个测试计划中有多个线程组，设置此项可生效，不设置时：每个线程组同时运行。

3.2 线程组

3.2.1 线程组的类别

Jmeter 中的线程组有 3 中，分别是：thread group，setup thread group，tearDown thread group。

手动划重点

三种线程组无本质区别，都可以实现多线程的效果。如果在测试计划下只存在一种的话，是没有区别的。区别在于若 3 中线程组同时存在于一个测试计划下时会存在执行先后的区别：setUp 先执行，然后再执行 thread group；最后执行 tearDown 线程组。

1：线程数=虚拟用户数。

2：Ramp-up Peroid：启动所有线程所需的总时间。

ps：Jmeter 中，线程启动的方式采用平均时间计算，线程的最小单位是 1；最终效果即：1 线程/N 秒。N=线程数/Ramp-Up Peroid。

线程组只能指定线程第一次启动时的间隔时间，不能控制之后的循环过程中的线程的间隔。

3：循环次数 每一个线程执行线程组内的组件的次数。

4：Delay Thread creation until needed：

5：调度器：设置线程组计划的启动时间和持续时间。

5.1：调度器是在点击启动后生效。

5.2：启动延迟的优先级别高于启动时间。

5.3：持续时间的优先级高于结束时间。

5.4：线程的停止条件是-->循环次数或持续时间满足设置。

3.2.2 线程组各配置项的意义

1：线程数就是允许当前线程组下脚本的线程数，等效于 LR 中的 Vusers，即：虚拟用户数。

2：Ramp-Up period：在多久时间内启动指定的线程数。

3：循环次数是指虚拟用户循环多少次线程组内的所有请求。

3.3 配置元件

用来配置脚本运行时所需的一些环节值，配置原件是全局的，是在 Sampler 运行之前编译执行的。

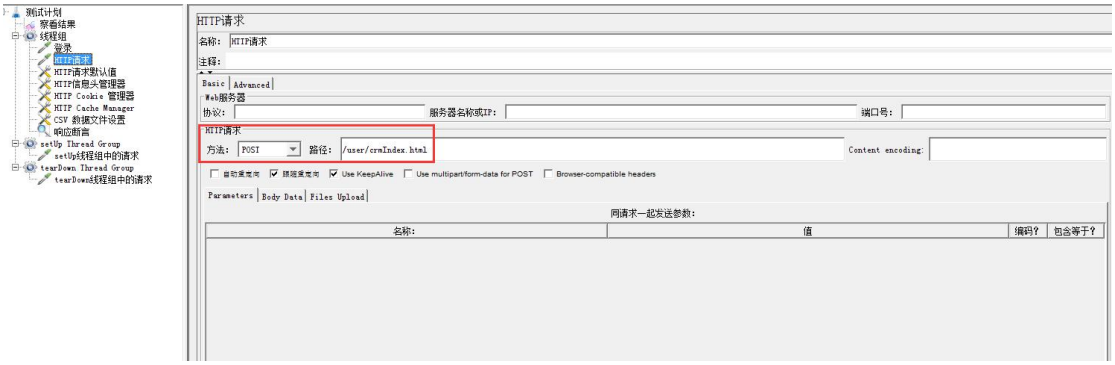
3.3.1 HTTP 请求默认值

实际项目中，我们的请求肯定有很多是公用是部分，比如：服务器名称，编码，协议。

我们可以把这部分内容提取出来做封装，当然，Jmeter 为我们提供了 HTTP 请求默认值。

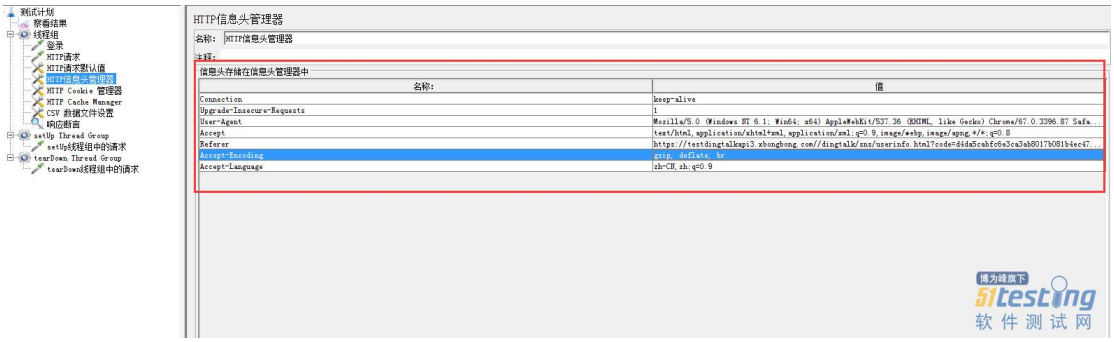
如上图，我将每个请求的协议，服务器名称，编码三项配置在 HTTP 请求默认值中，后续的 HTTP 请求就可以不用填写这些内容了。

后续 HTTP 请求如下：



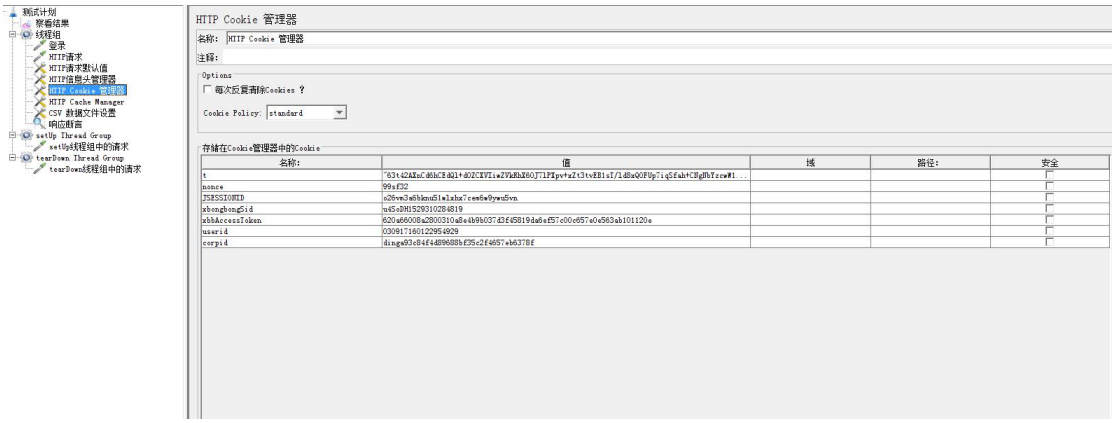
如上图，协议，服务器名称，Content Encoding3 项可以为空，因为 Jmeter 会使用 HTTP 请求默认值中的配置，如果在 HTTP 请求中配置了这 3 项则会覆盖 HTTP 请求默认值的配置。

3.3.2 HTTP 信息头管理器



顾名思义，就是将请求的头部信息集中管理起来。

3.3.3 HTTP Cookie 管理器



1. Cookie 管理器就像一个 web 浏览器那样存储并发送 cookie。

如果你有一个 HTTP 请求，其返回结果里包含一个 cookie，那么 Cookie 管理器会自动将该 cookie 保存起来，而且以后所有的对该网站的请求都使用同一个 cookie。每个 JMeter 线程都有自己独立的"cookie 保存区域"。因此，如果你在测试网站的时候使用了 Cookie 管理器来存储 session 信息的话，那么每个 JMeter 线程将会拥有自己独立的 session。**注意这些 cookie 不会显示在 Cookie 管理器里，你可以通过察看结果树来对

其进行察看。

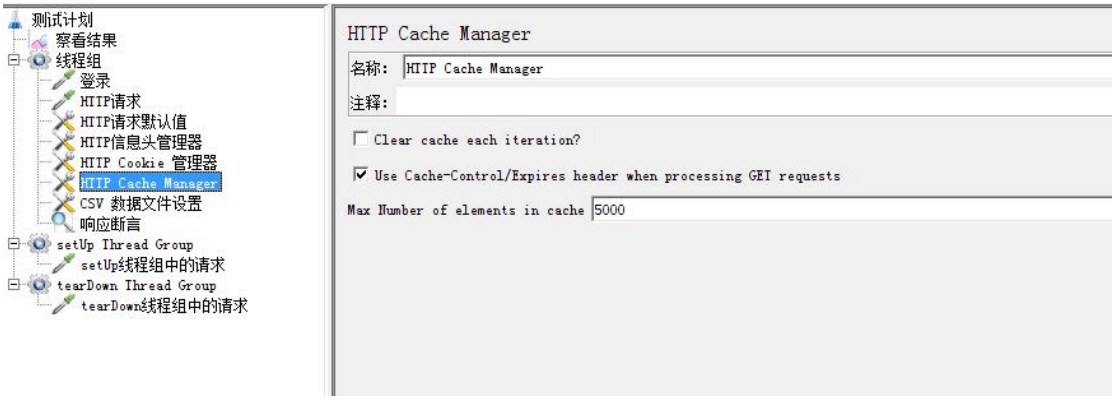
2. 接收到的 cookie 数据可以作为 JMeter 线程的参数进行存储

要将 cookie 存储为参数，定义属性"CookieManager.save.cookies=true"。cookie 在被保存之前会在名字上加上 "COOKIE_" 前缀(避免和本地参数重复)。设置好一会名字为 TEST 的 cookie 可以用 ${COOKIE_TEST} 进行引用。如果不希望这个前缀可以对属性 "CookieManager.name.prefix=" 进行定义。

3. 手工添加一个 cookie 到 Cookie 管理器

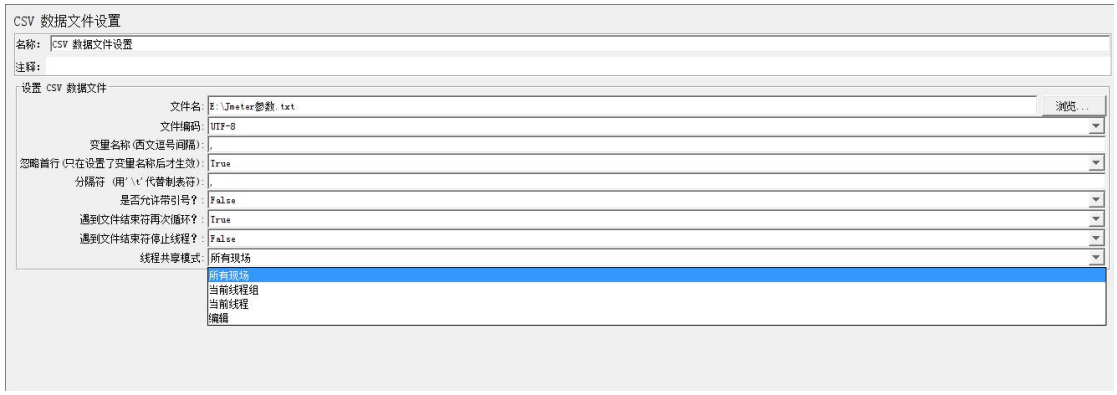
注意如果你这么干了，这个 cookie 将被所有 JMeter 线程所共享。这种方式用于创建有很长过期日期的 cookie。

3.3.4 HTTP Cache Manager



管理线程组下所有请求的缓存。

3.3.5 CSV 数据文件设置



文件名：参数文件的地址，可以是相对路径，也可以是绝对路径。此外，也可以使用 Jmeter 的用户自定义变量来参数化参数文件的路径。

注意：相对路径的根目录是 Jmeter 的启动目录(即：%JMETER_HOME%\bin 或 ${JMETER_HOME/bin})。

文件编码：读取参数文件使用的编码格式，此处一定要和参数文件的编码格式一致，强烈建议使用 UTF-8 格式保存参数文件。

变量名称：定义的参数名称，用逗号隔开，将会与参数文件中的参数对应；如果此

处参数个数比参数文件中的参数列多，多余的参数取不到值；反之，参数文件中的部分列将无参数对应。

忽略首行：是否忽略第一行，跟进实际情况定，如果首行是你定义的列名，可以设置为 True。

分隔符：用来分割参数文件的分隔符，默认为逗号；如果参数文件中用 tab 分隔，此处应为”\t”。

是否允许带引号？：如果设置为 True，则允许分隔完成的参数里面有分隔符出现。

遇到文件结束符再次循环？：设置为 True，则参数文件循环遍历；设置为 False，则参数文件遍历完成后不循环(Jmeter 在测试执行过程中每次迭代会从参数文件中心取一行数据，从头遍历到尾)。

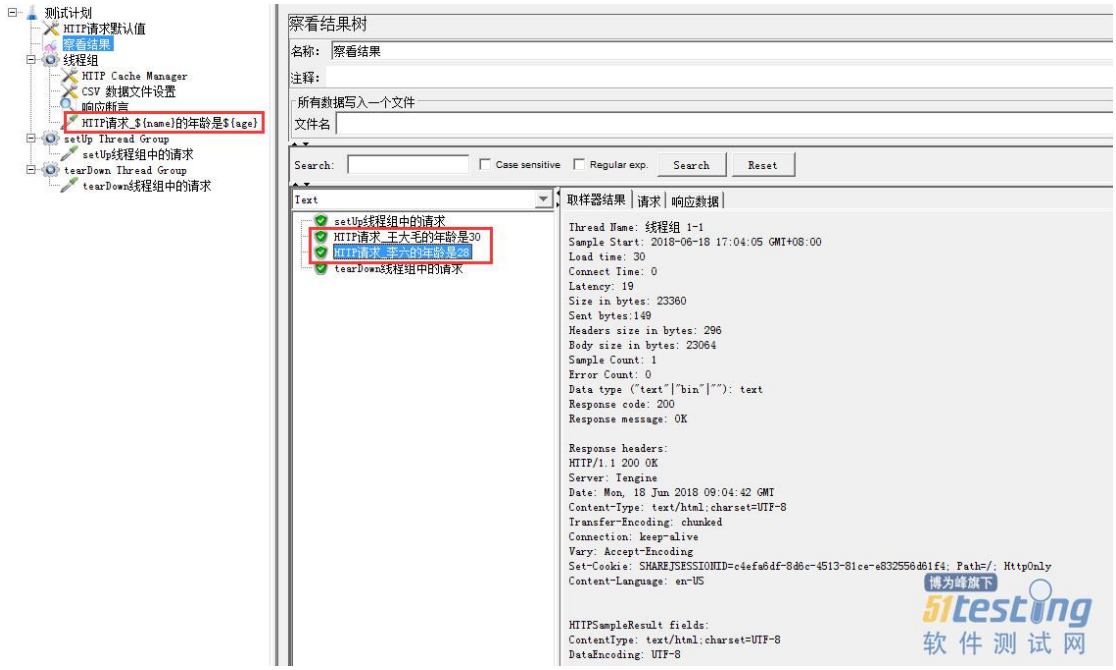
遇到文件结束符停止线程？：和<遇到文件结束符再次循环>设置为 False 时复用，设置为 True 则停止测试；设置为 False 则不停止。

线程共享模式：

1 所有线程：参数文件对所有线程共享，包括同一测试计划中的不同线程组。

2 当前线程组：值对当前线程组中的线程共享。

3 当前线程：仅当前线程获取参数。



3.4 定时器

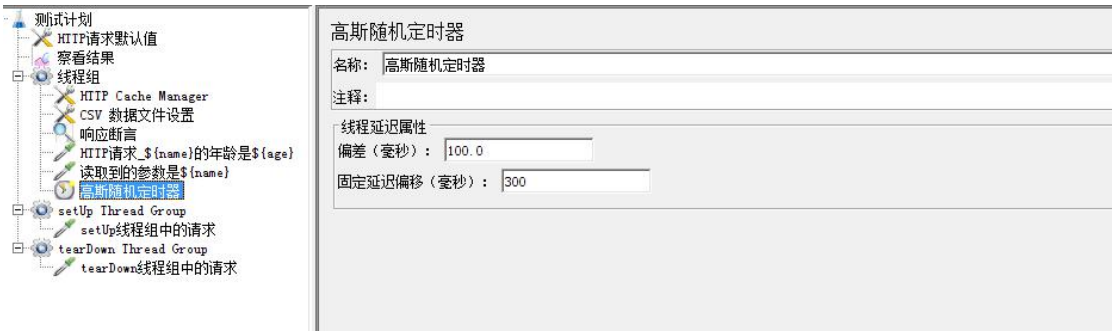
运行在作用域内的每一个请求之前，和组件本身的先后次序无关，而且运行次数等于作用域内的请求数量。

3.4.1 固定定时器

固定定时器可以用来模拟用户思考时间。

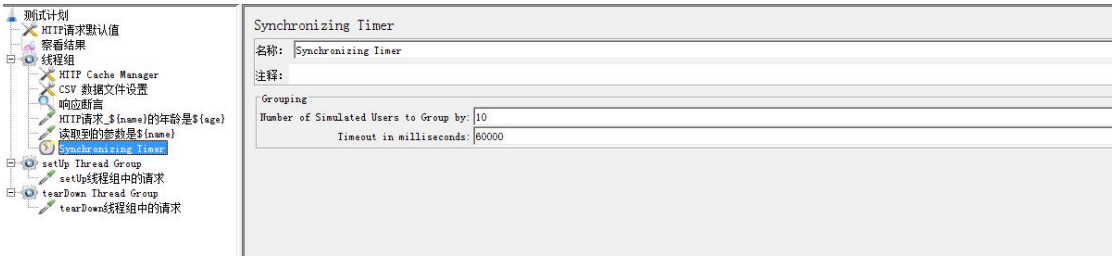
定时器放在不同的组件下，其作用域不同。如果放置在线程组下，则线程组内每个请求间的间隔都会是这个设置固定定时器延迟时长。

3.4 2 高斯定时器



固定延迟是指设置的固定思考时间，实际线程运行中的思考是时间是以固定延迟为基础浮动一定的偏差，高斯定时器也用来模拟用户思考时间。

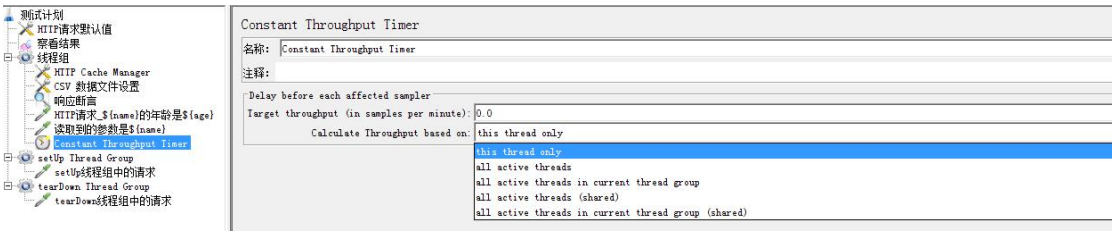
3.4.3 Synchronizing Timer



Synchronizing Timer 用来模拟集合点

注意：这里的超时设置的是到达集合点的第一个虚拟用户的等待时间。和 LR 不一样，LR 中设置的超时是 Vuser 之间的等待时间。

3.4.4 Constant Throughput Timer



吞吐量定时器，可尽量保持吞吐量在一定范围内。

吞吐量计算模式有 5 中

1 This thread only：仅针对当前线程，即：线程间互不干扰。

2 All active threads：针对所有线程，把所有线程的吞吐量合在一起作为因子计算。

3 All active threads in current thread group：针对当前线程组中的所有线程。

4 All active threads(shared)：线程延迟计算是基于任意一个线程上次运行的时间，也就是随便获取一个线程的运行时间来进行计算。

5 All active threads in current thread group(shared)：在当前线程组中任取一个线程的上

次运行时间来计算延迟，和 4 相近。

3.5 前置处理器

运行在作用域内的每一个请求之前，和组件本身的先后次序无关，而且运行次数等于作用域内的请求数量。

3.5.1 BeanShell PreProcessor

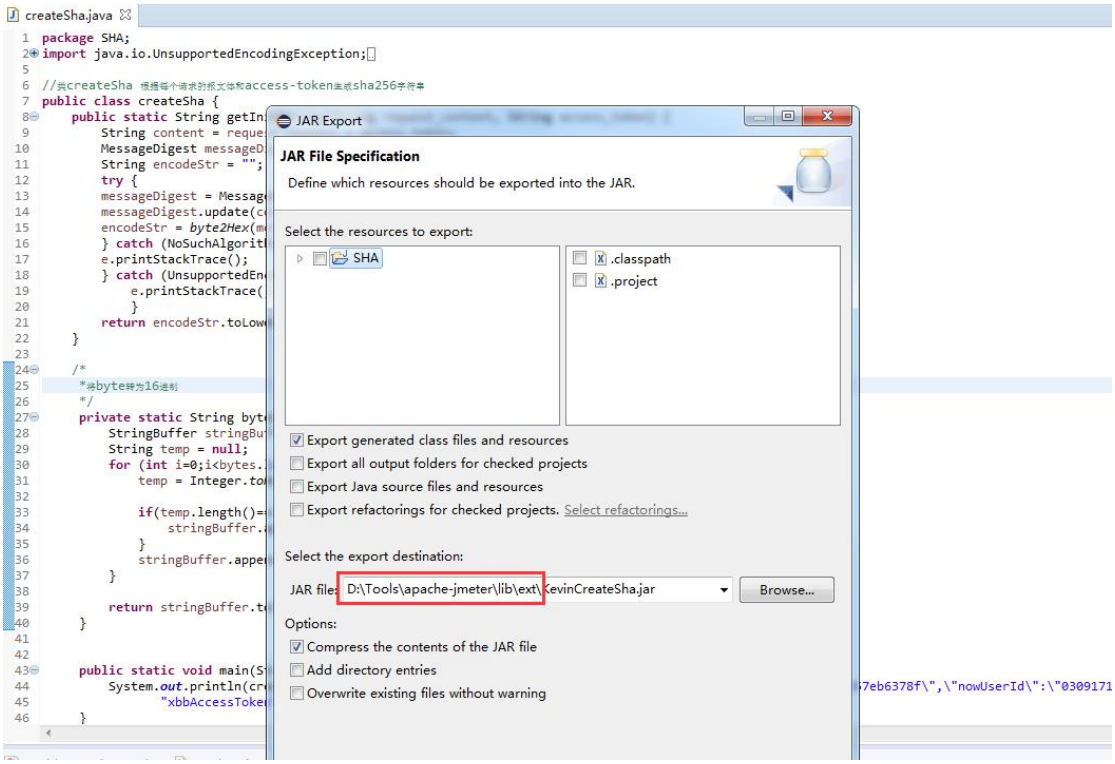
这里以实际例子来说明吧，我们这里的每个请求都会将请求的 param 和 accessToken 组成的字符串进行 sha-256 加密，然后作为 sign_code 成为每个请求中的一部分，所以我们这里使用 BeanShell 前置处理器先把每个请求的 sign_code 生成。

Java 源码如下：

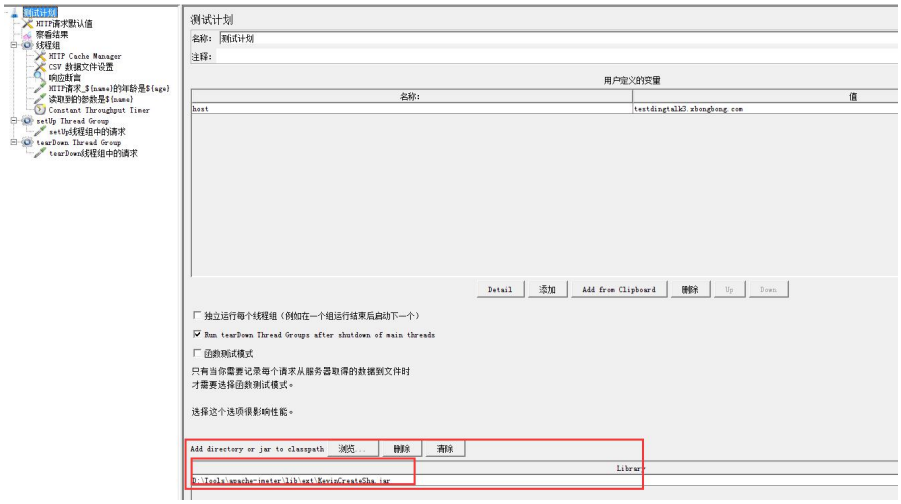
```
package SHA;
import java.io.UnsupportedEncodingException;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

//类 createSha 根据每个请求的报文体和 access-token 生成 sha256 字符串
public class createSha {
public static String getInitString(String request_content, String access_token) {
String content = request_content + access_token;
MessageDigest messageDigest;
String encodeStr = "";
try {
messageDigest = MessageDigest.getInstance("SHA-256");
messageDigest.update(content.getBytes("UTF-8"));
encodeStr = byte2Hex(messageDigest.digest());
} catch (NoSuchAlgorithmException e) {
e.printStackTrace();
} catch (UnsupportedEncodingException e) {
e.printStackTrace();
}
return encodeStr.toLowerCase();
}

/*
*将 byte 转为 16 进制
*/
private static String byte2Hex(byte[] bytes) {
StringBuffer stringBuffer = new StringBuffer();
String temp = null;
for (int i=0;i<bytes.length;i++) {
```

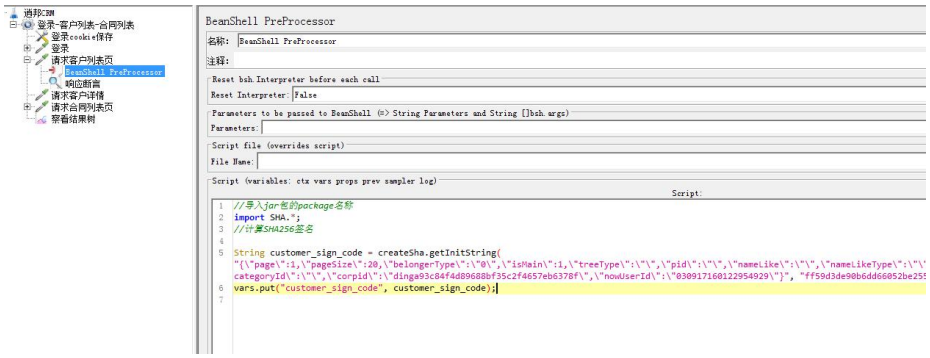
```
temp = Integer.toHexString(bytes[i] & 0xFF);

if(temp.length()==1) {
stringBuffer.append("0");
}
stringBuffer.append(temp);
}

return stringBuffer.toString();
}
}
```

然后将工程导出 jar 包放置在 Jmeter 安装目录下的 lib/ext 目录中。

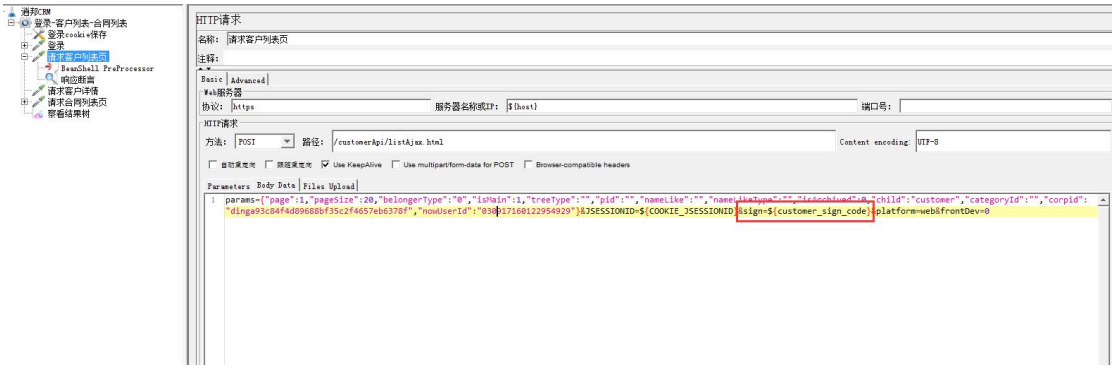


在 Jmeter 中引用 Jar 包：





```
//导入 jar 包的 package 名称
import SHA.*;
//计算 SHA256 签名
String customer_sign_code = createSha.getInitString("{\"page\":1,\"pageSize\":20,\"belongerType\":\"0\",\"isMain\":1,\"treeType\":\"\",\"pid\":\"\",\"nameLike\":\"\",\"nameLikeType\":\"\",\"isArchived\":0,\"child\":\"customer\",\"categoryId\":\"\",\"corpid\":\"dinga93c84f4d89688bf35c2f4657eb6378f\",\"nowUserId\":\"030917160122954929\"}", "ff59d3de90b6dd66052be25524444cb208e652482880040e28fe0eb26334f28b");
vars.put("customer_sign_code", customer_sign_code);
```



3.6 后置处理器

运行在作用域内的每一个请求之后，和组件本身的先后次序无关，而且运行次数等于作用域内的请求数量。

3.6.1 正则表达式提取器

引用名称：输出的参数名称；参数的值受到正则表达式匹配内容的影响。

正则表达式：自己多多摸索，这是必备技能。

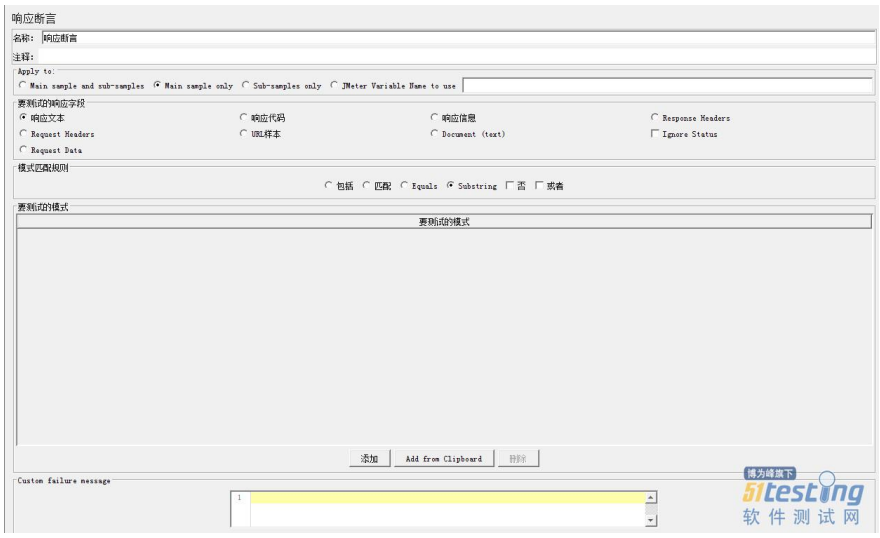
模板：常量最后引用名称就会获取常量的值，位置变了N：表示将匹配到的第 N 个内容的值赋值给引用变量。

匹配数字：正整数：将第 N 次的模板指定的括号的值传递给变量。数字 0：随机将匹配的数据传递给变量。

缺省值：如果正则表达式匹配不到数据，则会使用缺省值，一般留空即可。

3.7 断言

最常用的是<响应断言>



APPly to:适用范围

- Main sample and sub-samples:作用于父节点取样器及对应子节点取样器
- Main sample only：仅作用于父节点取样器
- Sub-samples only:仅作用于子节点取样器
- JMeter Variable:作用于 jmeter 变量(输入框内可输入 jmeter 的变量名称)

- 要测试的响应字段：要检查的项
- 响应报文
- Documeng(text)：测试文件
- URL 样本
- 响应代码
- 响应信息
- Response Headers:响应头部
- Ignore status：忽略返回的响应报文状态码
- 模式匹配规则：
- 包括：返回结果包括你指定的内容
- 匹配：（好像跟 Equals 查不多，弄不明白有什么区别）
- Equals：返回结果与你指定结果一致
- Substring：返回结果是指定结果的字串
- 否：不进行匹配

要测试的模式:即填写你指定的结果（可填写多个）,按钮【添加】、【删除】是进行指定内容的管理

3.8 逻辑控制器

控制 Jmeter 中各种组件的执行逻辑。

3.8.1 ForEach Controller(循环控制器)

ForEach 控制器一般和用户自定义变量一起使用，其在用户自定义变量中读取一系列相关的变量。该控制器下的采样器或控制器都会被执行一次或多次，每次读取不同的变量值。如下图：

参数:

- Input Variable Prefix：输入变量前缀
- Output variable name：输出变量名称

- Start index for loop(exclusive)：循环开始的索引（这里如果不填写，默认从 1 开始，如果没有 1 开始的变量，执行时会报错）
- End index for loop(inclusive)：循环结束的索引
- Add”_”before number：输入变量名称中是否使用“_”进行间隔。

3.8.2 Once Only Controller

作用：在测试计划执行期间，该控制器下的子结点对每个线程只执行一次，登录场景经常会使用到这个控制器。

注意：将 Once Only Controller 作为 Loop Controller 的子节点，Once Only Controller 在每次循环的第一次迭代时均会被执行

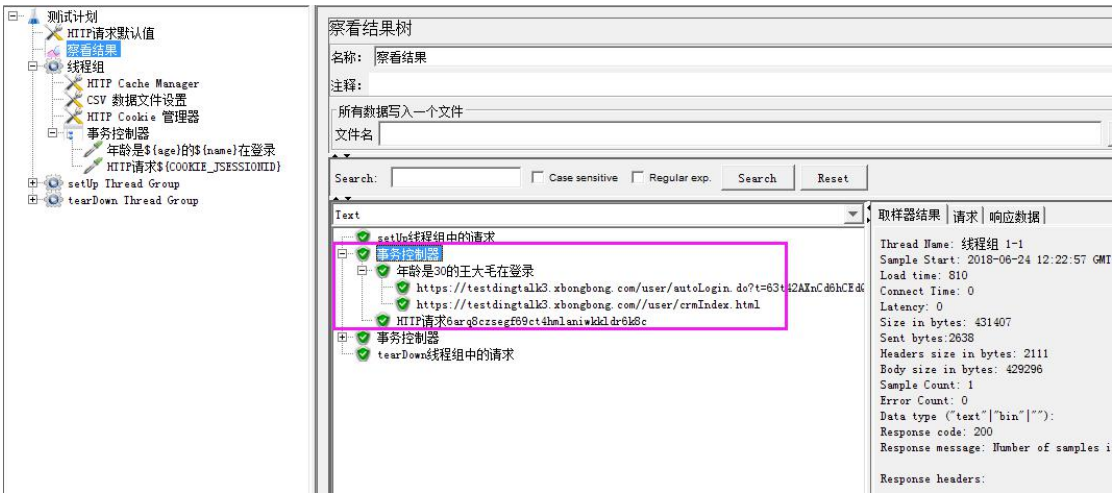
3.8.2 Transaction Controller(事务控制器)

Jmeter 中默认每个请求是一个事务；类比 LR 中每个步骤是一个事务。

如果想把多个请求作为一个事务，使用逻辑控制器-事务控制器元件。

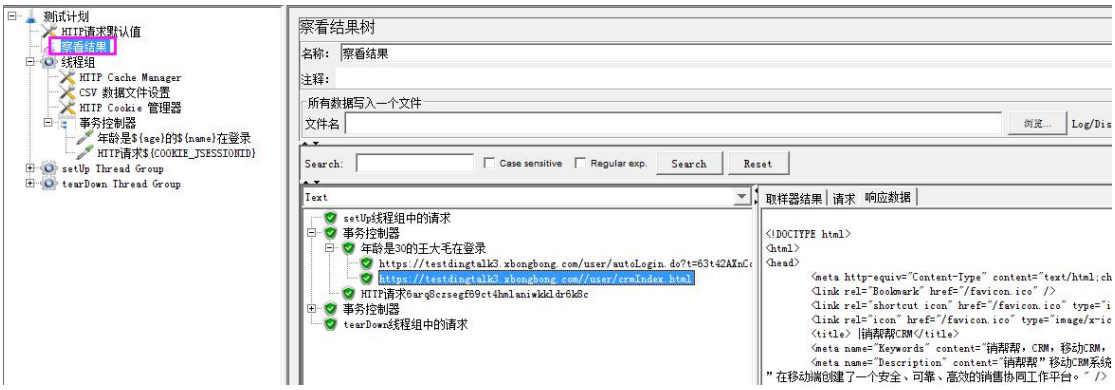
Generate parent sample：生成父取样器。

Include duration of timer and pre-post processor in generated sample：响应时间包含前置处理器(不建议勾选)



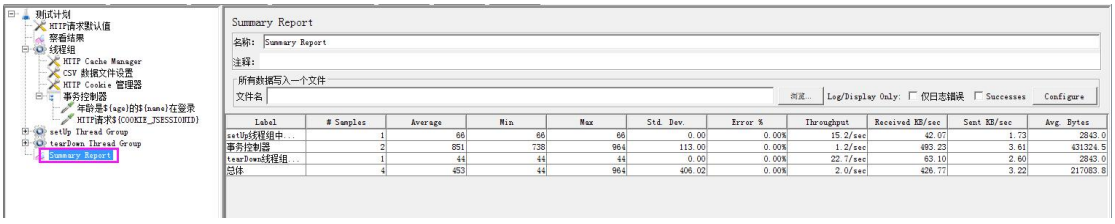
3.9 监听器

3.9.1 查看结果树



对每个取样器都作记录其详细的请求内容和服务器返回的响应报文。

3.9.2 Summary Report



- Label：取样器/监听器名称
- Samples ： 事务数量
- Average：平均一个完成一个事务消耗的时间（平均响应时间）
- Median：所有响应时间的中间值，也就是 50% 用户的响应时间，大概是这个意思
- Min：最小响应时间
- Max：最大响应时间

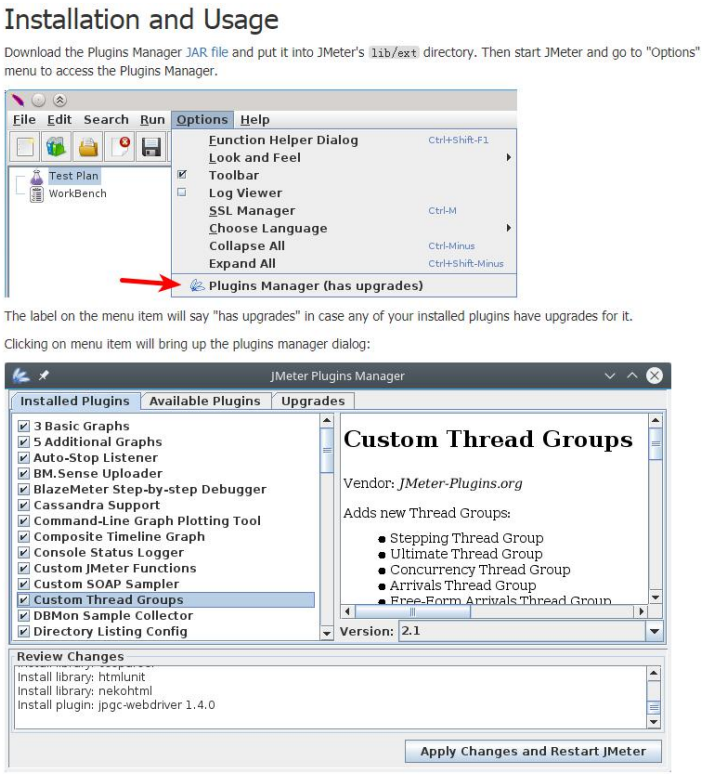
- 以上单位都是 ms
- Std.Dev：偏离量，越小表示越稳定
- Error %：错误事务率
- Throughtput：每秒事务数，即 tps
- KB/sec：网络吞吐量

3.9.3 其他监听器插件

请到 https://jmeter-plugins.org/查看下载您需要的其他插件。

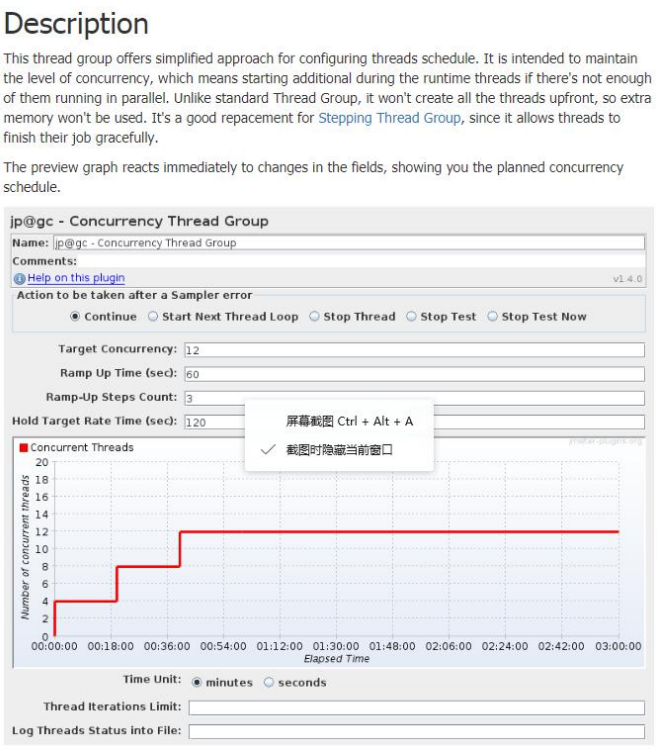
推荐几个不错的。

Plugins Manager：插件管理器



3 Basic Graphs：响应时间 vs 时间图

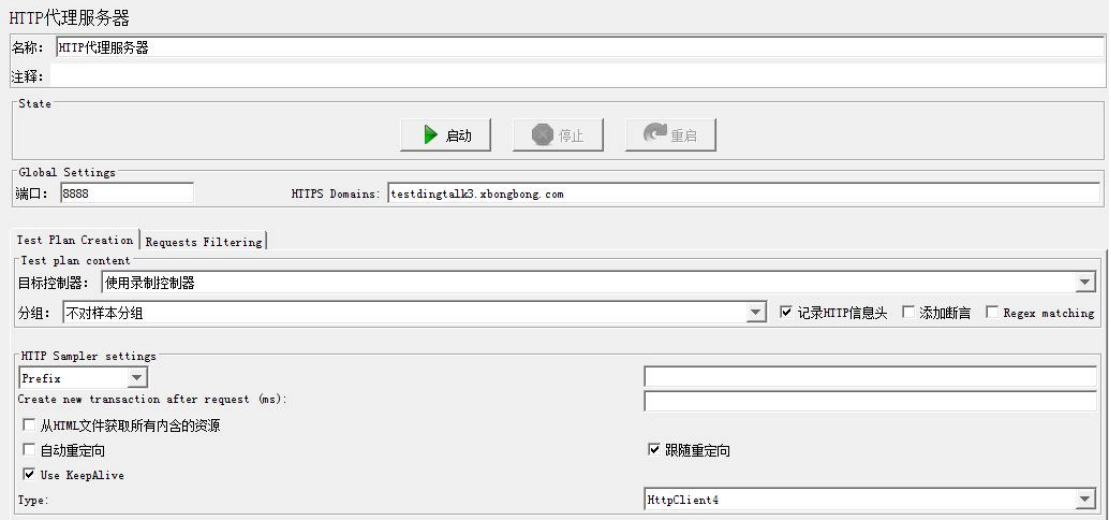
Custom Thread Groups：自定义线程组(下文的浪涌模拟会用到)



四：脚本开发

4.1 使用代理录制

测试计划中添加<非测试元件>-<HTTP 代理服务器>



包含模式：只录制所指定的规则请求。

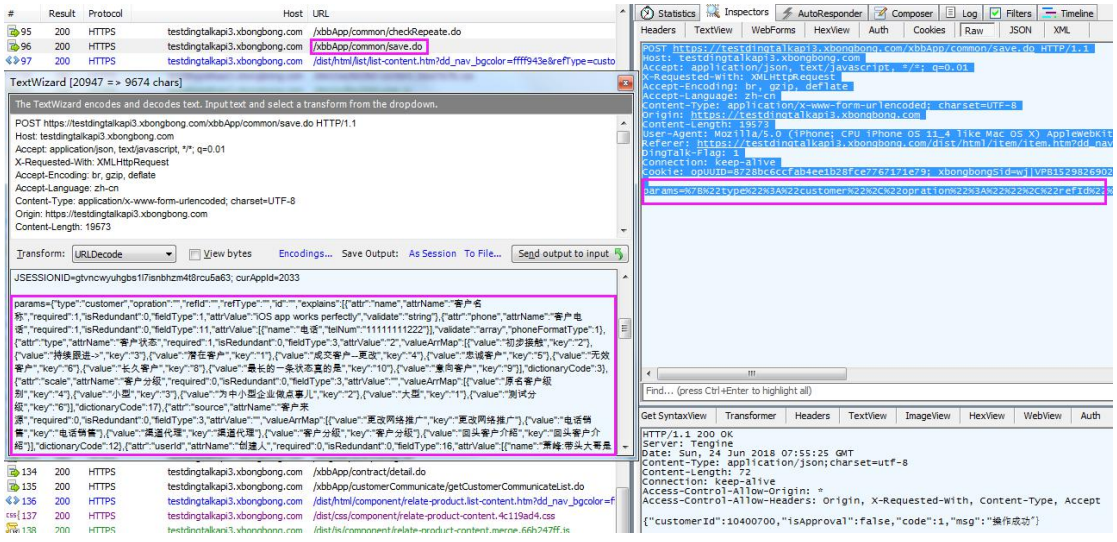
排除模式：不录制所指定的规则请。

重点必考题

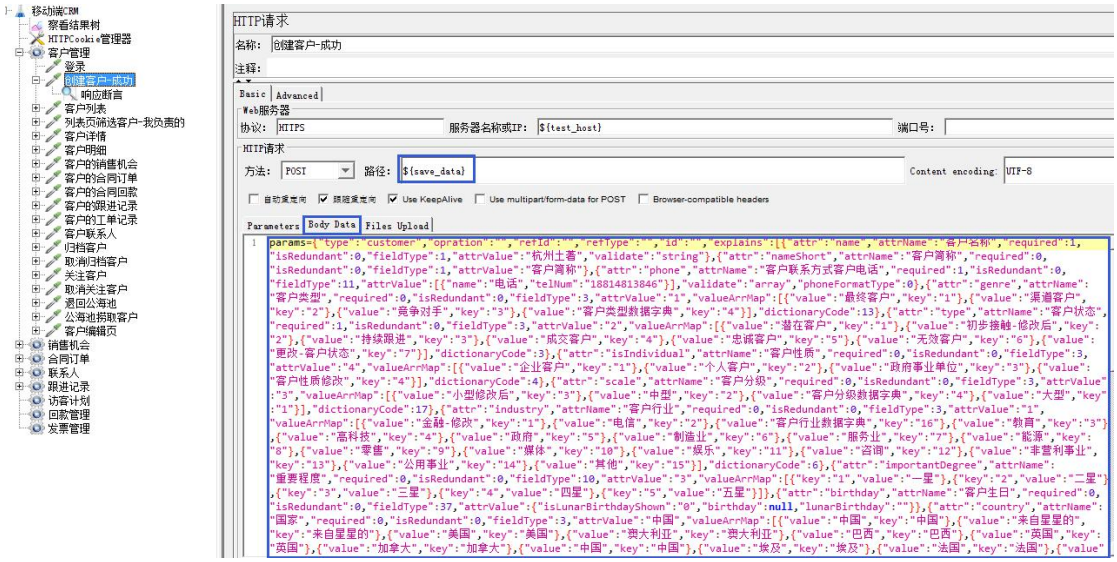
Jmeter 中的脚本模式是 LR 中的 HTML 模式(LR 中有 URL&HMTL 两种模式。) 所以，Jmeter 中的静态资源的请求可以手动屏蔽(如果不需要每次都请求静态资源)。切记录制完成后一定要停止代理，还原设置。

4.2 手工开发

4.2.1 抓取报文，手工设置请求



Fiddler 抓取报文，然后将请求的 body 部分写到 Jmeter-HTTP 请求的 Body Data 里。



五：场景设计

5.1 参数化

为了实现不同用户的不同请求；即：业务逻辑相同，数据不同。

参数化的实现方式有函数和文件两种方式。

1：使用 Jmeter 所提供的一些函数来生成参数值。推荐使用函数助手对话框来实现 2：_javascript()函数允许使用 js 代码来生成一些参数值，但是要求最后一句话是一个变量或变量表达式；这个函数会自动返回最后的变量或者变量表达式的值。

当然也可以使用 BeanShell 来实现，举例如下：

文件方式实现过程中，参数文件类型可以是.csv 或者.txt 类型。通过函数或者配置元件-CSV Data Set Config 组件实现读取。

两点注意事项

Jmeter 中：对变量，参数，函数的使用都是遵循相同的调用格式：${变量名}，${参数名}，${函数名(形参...)}。

在 Jmeter 组件中，所有鼠标点击可以输入的地方都可以做参数调用，实现参数化。

5.2 关联设置

关联最终要做的操作本质是：先存后用(先保存服务器响应的一些特殊作用的值，在后续的请求中使用保存的服务器动态响应的值。)

LR 中的关联通过函数 web_reg_sava_param 一系列注册函数(带 reg)实现数据保存；Jmeter 中通过后置处理器-正则表达式提取器实现。

5.3 检查点

通过断言来实现

5.4 事务

Jmeter 中默认每个请求是一个事务；类比 LR 中每个步骤是一个事务。

如果想把多个请求作为一个事务，使用逻辑控制器-事务控制器元件。

5.5 思考时间

一般使用时间定时器；如：固定/高斯随机定时器。

5.6 集合点

使用定时器-Synchronizing Timer 来实现。

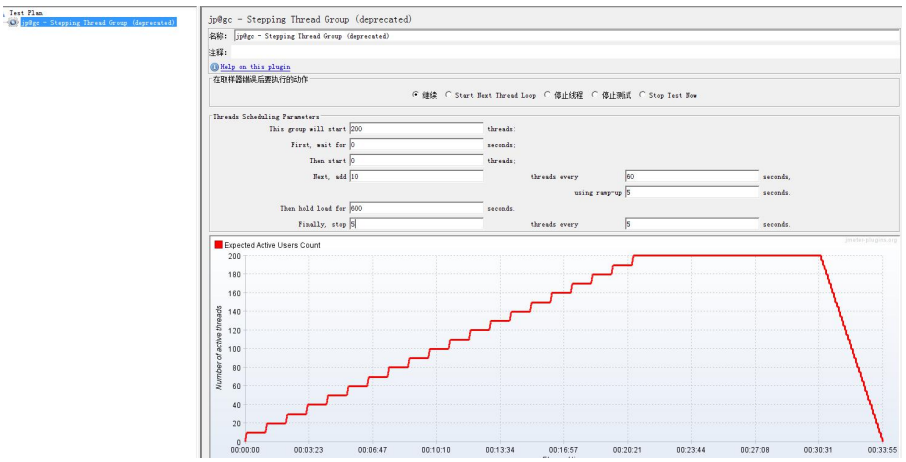
注意：这里的超时设置的是到达集合点的第一个虚拟用户的等待时间。和 LR 不一样，LR 中设置的超时是 Vuser 之间的等待时间。

5.7 浪涌模拟

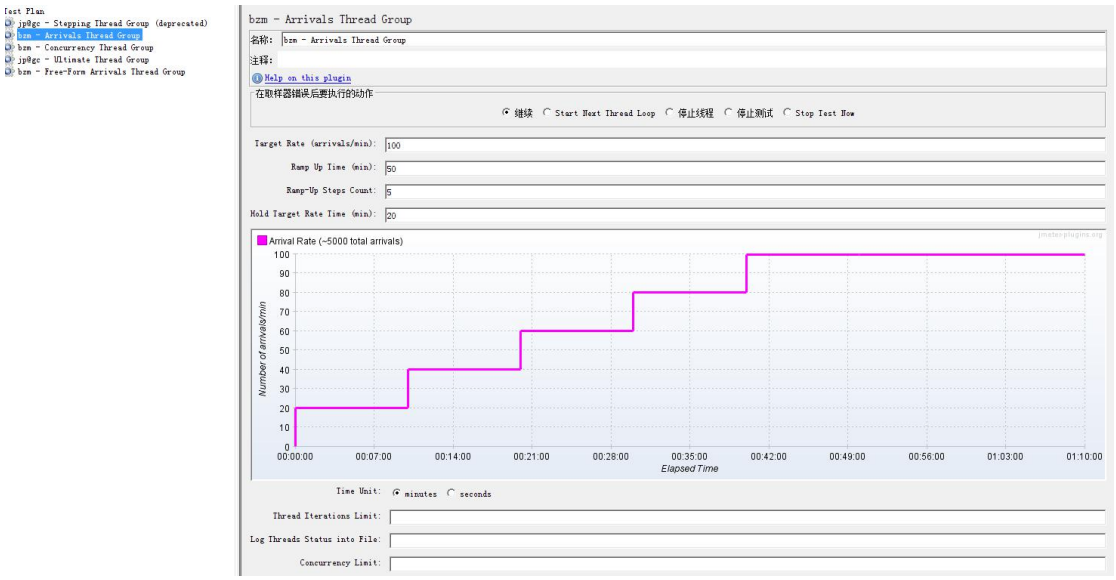
使用 Jmeter 的第三方插件 Custom Thread Groups

有以下几种线程组(模式)可以选择。

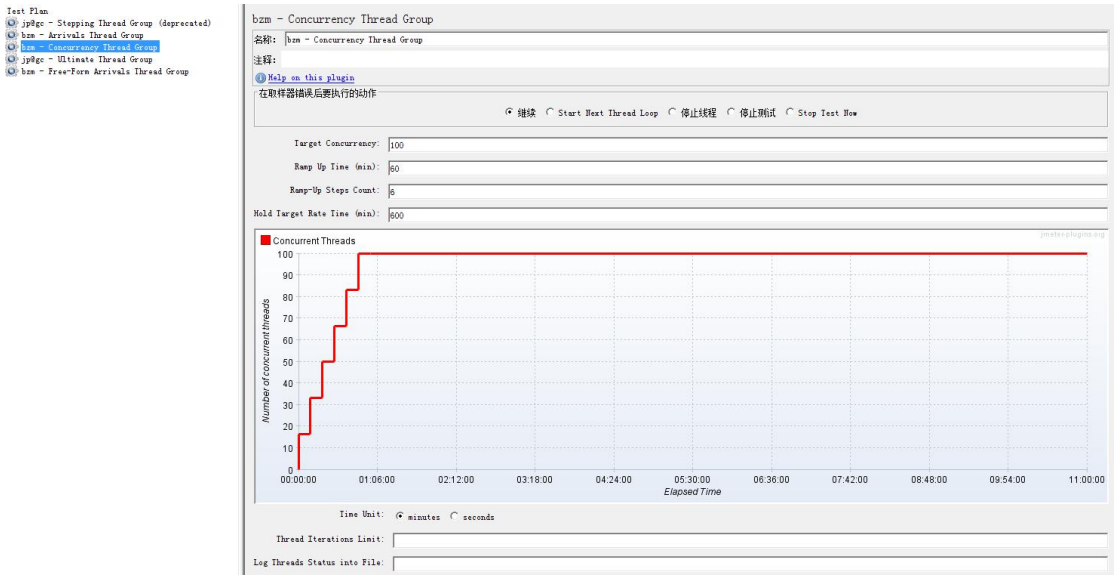
1：Stepping Thread Group (deprecated)



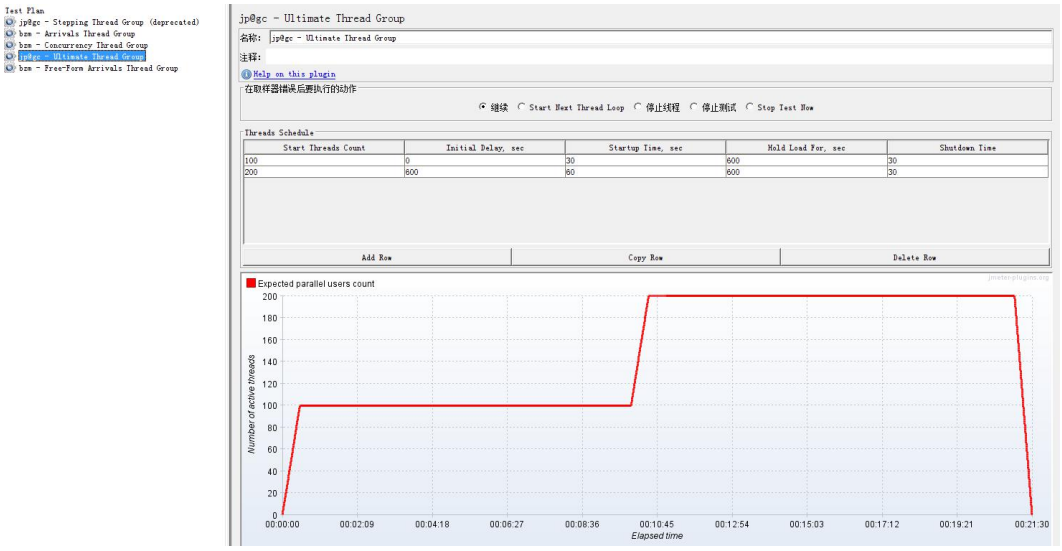
2：Arrivals Thread Group



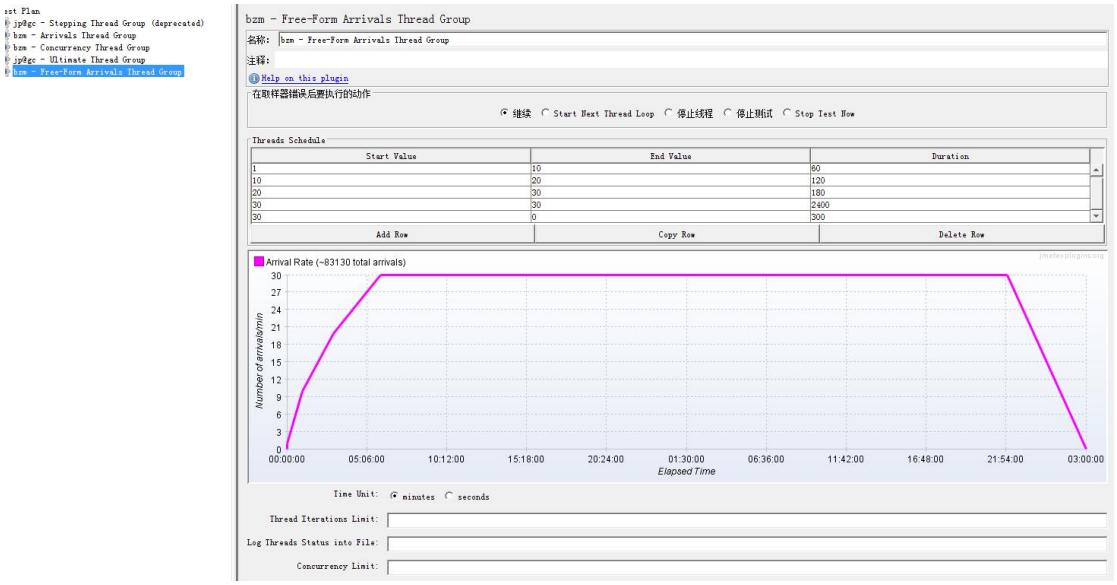
3：Concurrency Thread Group



4：Ultimate Thread Group



5：Free-Form Arrivals Thread Group



六：实施压测

6.1 命令行执行

-h 帮助 -> 打印出有用的信息并退出

-n 非 GUI 模式 -> 在非 GUI 模式下运行 JMeter

-t 测试文件 -> 要运行的 JMeter 测试脚本文件

-l 日志文件 -> 记录结果的文件

-r 远程执行 -> 在 Jmter.properties 文件中指定的所有远程服务器

-H 代理主机 -> 设置 JMeter 使用的代理主机

-P 代理端口 -> 设置 JMeter 使用的代理主机的端口号

示例如下：

例 1：测试计划与结果，都在%JMeter_Home%\bin 目录

> jmeter -n -t test1.jmx -l result.jtl

例 2：指定日志路径的：

> jmeter -n -t test1.jmx -l report\01-result.csv -j report\01-log.log

例 3：默认分布式执行：

> jmeter -n -t test1.jmx -r -l report\01-result.csv -j report\01-log.log

例 4：指定 IP 分布式执行：

> jmeter -n -t test1.jmx -R 192.168.10.25:1036 -l report\01-result.csv -j report\01-log.log

例 5：生成测试报表

> jmeter -n -t 【Jmx 脚本位置】-l 【中间文件 result.jtl 位置】-e -o 【报告指定文件夹】

> jmeter -n -t test1.jmx -l report\01-result.jtl -e -o tableresult

注意：

1）-e -o 之前，需要修改 jmeter.properties，否则会报错；

2）-l 与-o 目录不一样，最后生成两个文件夹下。

3）命令中不写位置的话中间文件默认生成在 bin 下，下次执行不能覆盖，需要先删除 result.jtl；报告指定文件夹同理，需要保证文件夹为空

6.2 联机压测

Step1：在负载机上启动 jmeter-server.bat

注意事项：jmeter-server 通过默认的 1099 端口进行监听和通信，如果 1099 端口被占用，需要修改控制机上 Jmeter 的配置文件 jmeter.properties 中的 server_port 属性。

Step2：在控制机上添加负载机。

修改控制机上的 Jmeter 的配置文件 jmeter.properties 中的 remote_hosts 属性(

形式如下：192.168.10.6:1099,192.168.10.168:1099)，多台负载机用逗号隔开。

Step3：重启控制机 Jmeter，点击远程启动/远程全部启动。

注意事项：

1：联机负载时，脚本的允许环境是负载机的环境，控制机和负载机上 Jmeter 版本，允许环境，环境变量，jar 包，参数文件必须一致

2：如果控制机和负载机的 OS 相同，脚本中对文件的使用可以通过绝对路径实现，如果 OS 不同，只能使用相对路径。

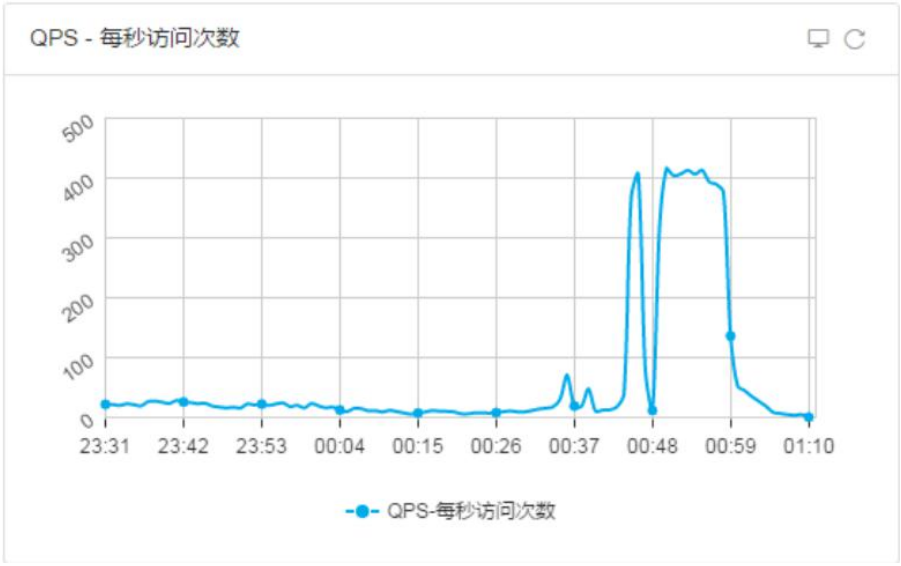
3：Jmeter 联机负载时，线程组的计划分别，同时在不同的负载机上执行，所以对服务器而言：总压力=线程组设定的压力 x 负载机数量。Jmeter 的联机负载和 LR 有很大不同，Jmeter 的联机负载会使负载翻倍，而 LR 的联机负载不会改变控制机上设定的负载。

4：和 LR 一样，报告文件在控制机上查看。

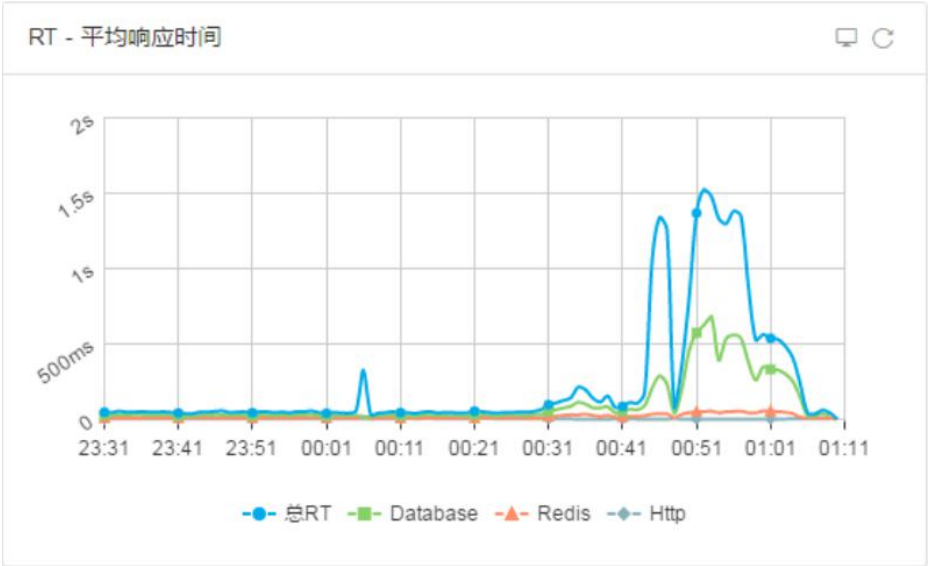
七：压测报告

压测过程中我们对服务器进行了监控，重点关注的性能指标如下：

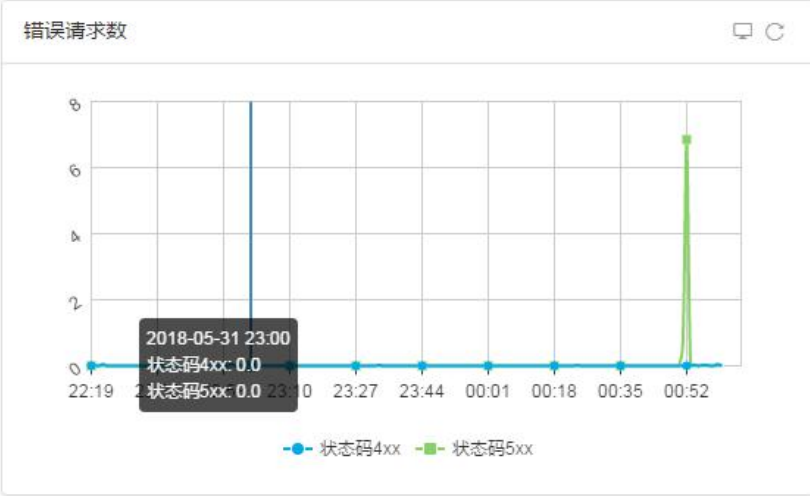
1：QPS-每秒访问次数



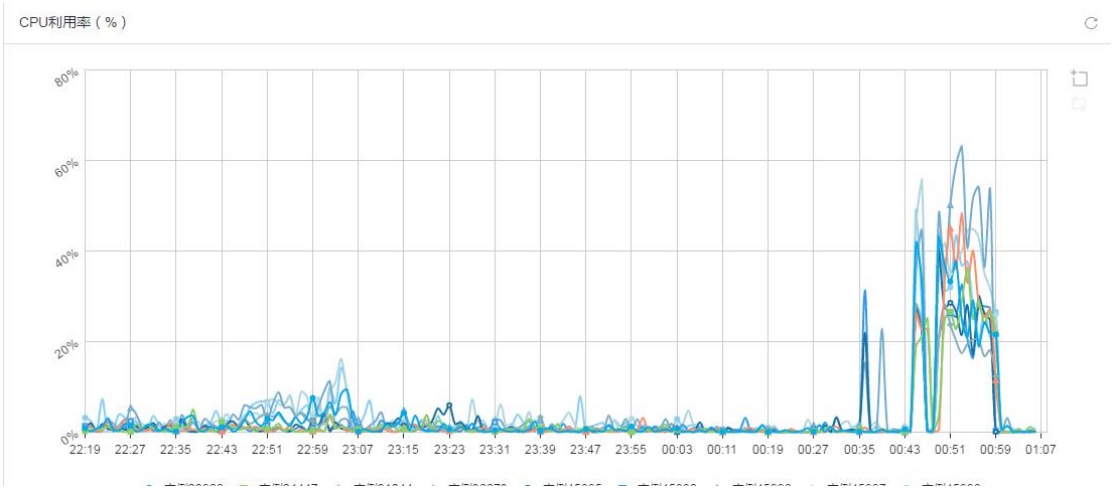
2：RT-平均响应时间



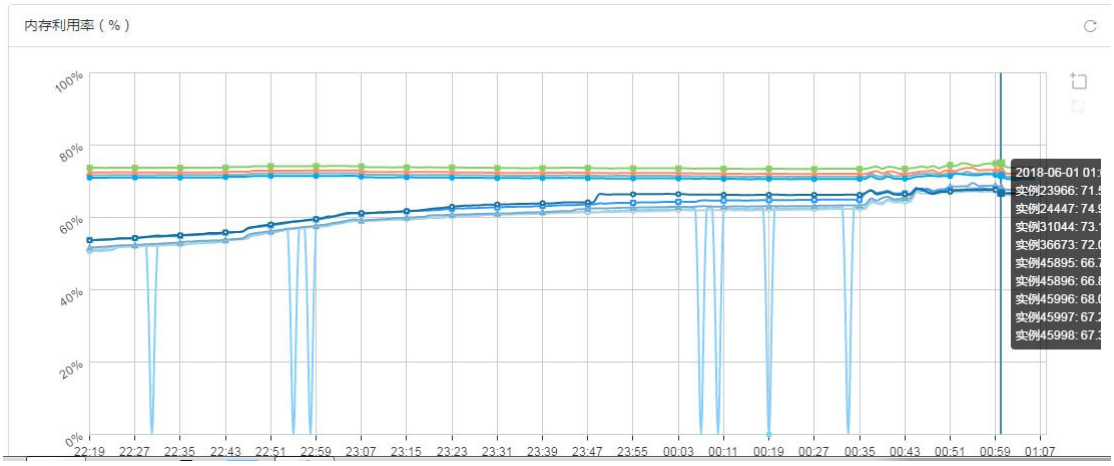
3：错误请求数



4：CPU 使用率



5：内存使用率



首页请求；客户列表；客户详情页；合同订单列表；合同订单详情页。

通过标准	是否符合标准	压测值
预估线上峰值 QPS 值*3<实际压测值	是	410
峰值 RT<=2000ms	是	1.5s
峰值错误率(统计非 200)<=0.1%	是	0.01%

峰值总 cpu 利用率<=70%	是	65%
峰值 load1<=cpu 总核数-0.5	是	3
峰值内存利用率<=80%	是	75%

测试结论：

App 项目日常 QPS 为 60，压测（2018-5-31 0:45 到 0:59）QPS 在 400 左右，RT 曲线和错误率曲线在 QPS 峰值 410 时出现失败事务(事务失败率 0.01%)，其他时间段并没有出现明显异常。系统可满足业务需求。

Jmeter 的强大之处远远不止文章中提到的这些，很多时候我们完全可以继承 AbstractJavaSamplerClient 自行开发脚本。

此外，开发能力强的同学，可以对 Jmeter 进行二次开发或者封装已扩展 Jmeter 的插件或优化使用体验，当然，刚接触到 Jmeter 皮毛的我就不抛砖引玉了，希望真正熟悉 Jmeter 的同学可以作更优秀的分享~ 期待！

❖ 拓展学习

■10 堂课让你“拿下”Jmeter，轻松应对性能测试：http://www.atstudy.com/course/177

测试工程师的捉虫历险记

◆作者：咖啡猫

一、 项目背景：

小 A 新入职一家 IT 公司，该公司主营业务是做监控云平台的。由于公司还处在成长期，所以目前就只有小 A 一名测试工程师。也就是通常所说的全能型人才，需求分析、功能测试、性能测试、持续集成、运维、技术支持、售后培训等等一系列的工作可能都需要小 A 来支持。并且部门内部没有其他人对测试非常了解，小 A 唯有自助，才能得到天助。一开始小 A 很是气馁，感觉自己从一个坑跳到另一个坑~但是他发现其实这样的架构也能有一些好处，比如之前不敢尝试的测试工具和方法可以在这里尝试。

某天，项目经理找到小 A，对他说：我们云平台的基本功能开发已经完成了，马上要进入试运行阶段，试运行阶段的数据量大概在 3000-4000 个终端的样子，希望你来评估一下平台能否承受这么大的压力。

二、需求分析

小 A 听了项目经理的话，就开始分析客户的需求。项目经理的话就是最原始的客户需求。用测试的专业术语来说就是：需要度量平台在现有配置下（CPU、内存、网络、硬盘），最大能承受多少模块的连接。显然这需要进行一次压力测试。

压力测试之前应该先保证所加压的模块功能流程是通的。所以先找开发的同事要来接口文档，弄清楚模块和平台之间的报文交互流程，如下：

登录开始

1.客户端给服务端第一包

696901003a00000001000100000001000100534757354c554241304800000000000053475 7354c554241304800000000000000000000000000000000000000000000036df

2.如果数据正确，服务端给客户端返回如下数据，如果数据不正确，服务端不会返回数据。

6969010012000000000000000000000004000100121E0D1101117AF6

3.客户端发给服务端第二包

696901002a00000001000100000001000300000000000000000000000000000000000000000 0000000000000000000000000000ef22

4.如果数据正确，服务端给客户端返回如下数据，如果数据不正确，服务端不会返回数据。

6969010012000000000000000000000004000100121E0D1101117AF6

至此登陆认证通过

心跳包及业务数据

5. 客户端发送业务数据

6.

69690100fe0000000100010000000400290001000000000140e1101111a000118015842353001011901323137360101 1a0130303530010011b013630000001011c012020202001011d012020202001011e012020202001011f012020000001012201f811000001012301040000000101040100000000010105010000000001010601000000000101070100000000010108010000000001010901000000000101 0a0100000000001010b0100000000001010c0100000000001010d01000000000102fa0000010000140e11011102fb0000010000140e11011102fc0000010000140e11011102fd0000010000140e11011102fe0000010000140e11011102ff0000010000140e11011114b7

7.服务端返回 0x10

8.客户端发送心跳包数据： 0x10

9.服务端返回 0x10

连接断开逻辑

客户端断开的逻辑 就是 4 分 10s 没有收到 心跳包 0x10。。然后客户端断开了 服务端自动断开

注：所有数据都是十六进制的字节流，心跳是 56s 一条，业务数据是 2min 一条

三、测试方案

3.1 测试环境

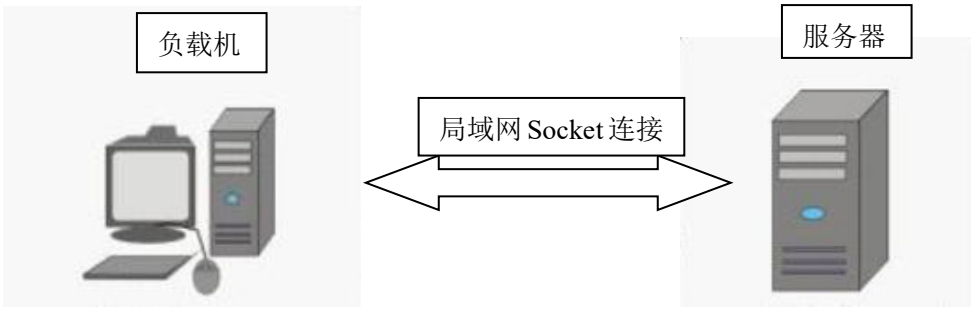
负载机	Intel(R)Xeon(R) CPU E3-1225 v3 @ 3.20GHZ 3.20GHZ 32G 内存（17.8G 可用） 1861.9G 硬盘(1496.6G 可用) WIN7 专业版
服务器 (武汉 Solax03)	Intel(R) Xeon(R) CPU E3-1225 v3 @ 3.20GHz 16G 内存（9.7G 可用） 800G 硬盘（719G 可用） CentOS

3.2 测试安排

时间	2018-3-1 和 2017-3-5 共五天时间

人员	小 A（测试）
输出	压力测试报告一份

3.3 网络拓扑



3.4 测试目标

1.测试系统在 Socket 并发 5000 短连接不断连接/断开的情况下，系统的性能表现。

2.测试系统在 Socket 并发 5000 长连接不断发送业务数据的情况下，系统的性能表现。

3.5 测试用例

编号	场景	连接类型	总事件数	并发数	迭代数	测试点
1	连接->登录->收发数据->断开	--	1	1	0	接口是否畅通
2	连接->登录->收发数据->断开…循环迭代	短连接	100000	5000	10	事件成功率、系统稳定性
3	连接->登录->收发数据→收发数据….循环迭代…断开连接	长连接	55000	5000	10	事件成功率、系统稳定性

注：总事件数是指登录事件+数据传输事件。连接事件包括：收发认证包 1 和收发认证包 2；数据传输事件包括：发送心跳包 1，发送心跳包 2，发送业务数据。

3.6 测试工具

Socket 链接工具、HP LoadRunner 11.0 性能测试工具、Office 办公软件、截图工具

四、测试执行过程

第一阶段，用 socket 连接工具，模拟客户端给平台接口发送数据。

发现 BUG： 发送一包登录数据，返回两包确认数据。

分析：代码 BUG，由于这个后台接口是由两个开发写的，可能是他们交接工作中，某些环节没有处理好。一般来说，报文交互，都是发送一包，返回一包。

解决方案：开发做代码重构，严格按报文交互的需求文档编写代码。

第二阶段，用 LR 的 VU 编写客户端脚本，在场景里加压。VU 脚本中参考函数如下：

```
lrs_create_socket("socket0", "TCP", "LocalHost=0", "RemoteHost=server:60000", LrsLastArg);
lrs_send("socket0", "buf0", LrsLastArg);
lrs_receive("socket0", "buf1", LrsLastArg);
lrs_close_socket("socket0");
数据文件 data.ws:
;WSRData 2 1
send   buf0 3
"123"
recv   buf1 3
"456"
-1
```

发现 BUG：加压不到十分钟，后台 java 线程就挂了。于是将后台日志打开，重复上述加压测试过程，抓住错误 LOG，发现原因在于高并发下 TCP 出现粘包拆包的行为。原因有两个：1 发送端需要等缓冲区满才发送出去，造成粘包。2 接收方不及时接收缓冲区的包，造成多个包接收。

解决方案：接收方创建一预处理线程，对接收到的数据包进行预处理，将粘连的包分开。

第三阶段：分析测试结果，得出测试结论。

用例 2（短连接并发 5000）的测试结果分析

- 事件通过率 100%和平均响应时间 27.6s

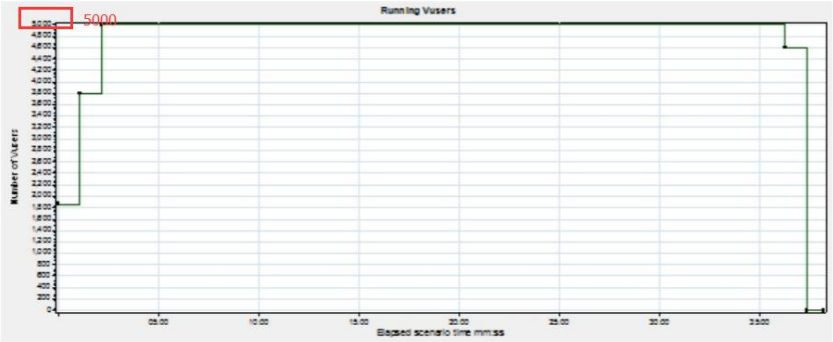
Performance Overview

Measurement	Value
Run Name	res.lrr
Weighted Average of Transaction Response Time	27.6
Total Passed Transactions	100000
Total Failed Transactions	0
Transactions Success Rate, %	100
Total Errors per Second	0
Total Errors	0

Transaction Summary

Run Name	Transaction Name	Minimum	Average	Maximum	Std. Deviation	90%	Pass Count	Fail Count	Stop Count
res.lrr	Conn_1	1.3	55.1	95.5	22.3	86.8	50000	0	0
res.lrr	Trans_1	0	0	0.2	0	0	50000	0	0

- 虚拟用户数曲线

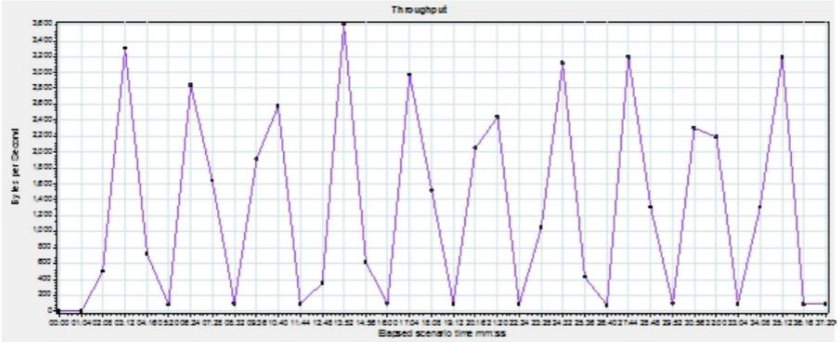


Color	Scale	Measurement	Graph Minimum	Graph Average	Graph Maximum	Graph Median	Graph Std. Deviation
	1	Run	0	2543.167	5000	3796	2050.336

Description: Displays the number of Vusers that executed Vuser scripts, and their status, during each second of a load test. This graph is useful for determining the Vuser load on your server at any given moment.

虚拟用户从 0 开始增长，增长到 5000 后，开始进行迭代操作。

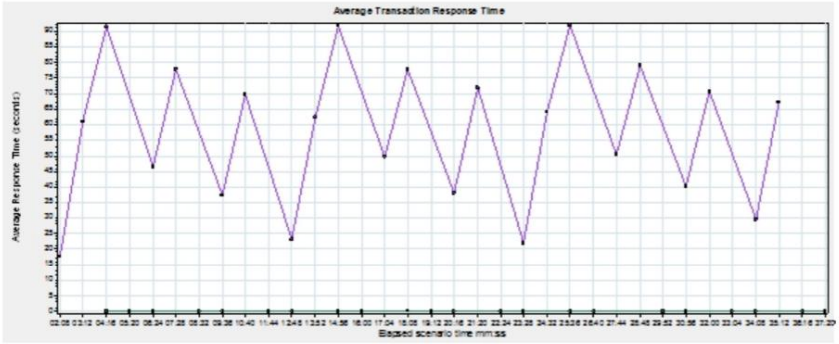
- 系统吞吐率曲线



Color	Scale	Measurement	Graph Minimum	Average	Graph Maximum	Graph Median	Graph Std. Deviation
	1	Throughput	0	1287.647	3611.516	1045.625	1220.282

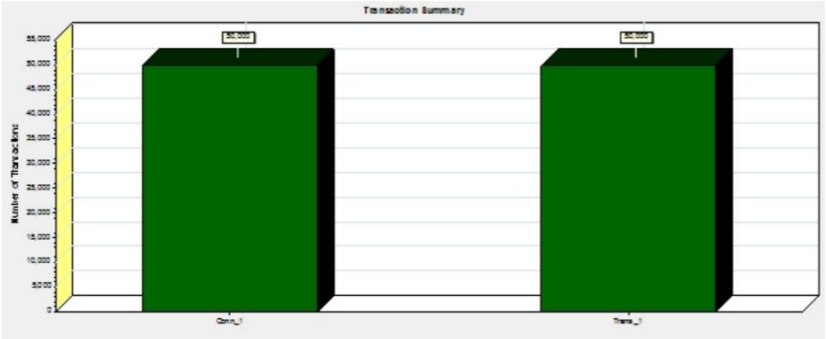
两次迭代之间，吞吐量都会降低，符合业务逻辑。

- 事件响应时间



Color	Scale	Measurement	Graph's Minimum	Graph's Average	Graph's Maximum	Graph's Median	Graph's Std. Deviation
	1	Conn_1	17.812	57.888	91.965	62.342	22.488
	1	Trans_1	0.012	0.023	0.194	0.013	0.038

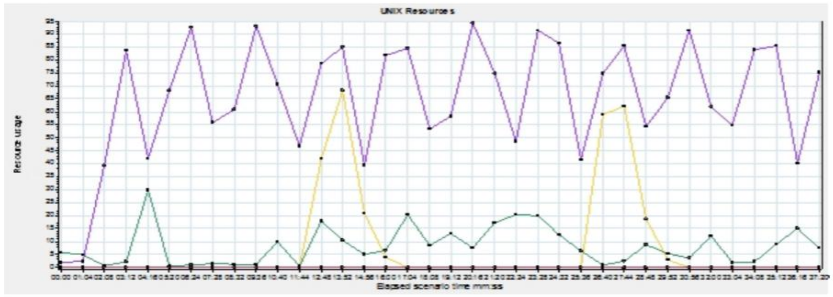
- 事件概览



Color	Scale	Measurement
	1	Pass

Description: Displays the number of transactions that passed, failed, stopped, or ended with errors.

- 系统资源占用情况



Color	Scale	Measurement	Minimum	Average	Maximum	Std. Deviation
	1000	Average load (Unix Kernel Statistics):10.5.0.56	0	0.008	0.156	0.023
	10	CPU Utilization (Unix Kernel Statistics):10.5.0.56	0.042	6.552	11.466	3.01
	10	Disk Traffic (Unix Kernel Statistics):10.5.0.56	0	0.768	13.121	1.899
	1	Paging rate (Unix Kernel Statistics):10.5.0.56	0	0	0	0

图 1　整个测试过程中服务器资源占用情况

```
[root@solax03 ~]# top
top - 07:34:33 up  6:43,  5 users,  load average: 0.10, 0.07, 0.01
Tasks: 176 total,   1 running, 175 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.2%us,  1.0%sy,  0.0%ni, 98.6%id,  0.0%wa,  0.1%hi,  0.1%si,  0.0%st
Mem:  14250876k total,  4026992k used, 10223884k free,   195232k buffers
Swap:  7192568k total,        0k used,  7192568k free,  2328860k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 9575 root      20   0 6177m 494m  13m S  9.3  3.5  27:24.49 java
 3129 root      20   0 1787m  51m 3004 S  0.7  0.4   3:53.50 cmf-agent
   20 root      20   0     0    0    0 S  0.3  0.0   0:51.79 events/1
 1743 root      20   0 10820  636  424 S  0.3  0.0   0:06.12 irqbalance
 2045 root      20   0  8304  588  460 S  0.3  0.0   0:41.98 rpc.rstatd
 2825 root      20   0 3312m 525m  23m S  0.3  3.8   2:40.69 java
 3390 root      20   0  598m  21m 7924 S  0.3  0.2   1:25.54 python2.6
19612 root      20   0 15036 1300  944 R  0.3  0.0   0:02.47 top
    1 root      20   0 19364 1536 1228 S  0.0  0.0   0:02.63 init
    2 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kthreadd
    3 root      RT   0     0    0    0 S  0.0  0.0   0:00.04 migration/0
    4 root      20   0     0    0    0 S  0.0  0.0   0:00.40 ksoftirqd/0
    5 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 migration/0
    6 root      RT   0     0    0    0 S  0.0  0.0   0:00.06 watchdog/0
    7 root      RT   0     0    0    0 S  0.0  0.0   0:00.16 migration/1
    8 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 migration/1
    9 root      20   0     0    0    0 S  0.0  0.0   0:00.54 ksoftirqd/1
   10 root      RT   0     0    0    0 S  0.0  0.0   0:00.05 watchdog/1
   11 root      RT   0     0    0    0 S  0.0  0.0   0:00.21 migration/2
   12 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 migration/2
   13 root      20   0     0    0    0 S  0.0  0.0   0:01.81 ksoftirqd/2
   14 root      RT   0     0    0    0 S  0.0  0.0   0:00.06 watchdog/2
   15 root      RT   0     0    0    0 S  0.0  0.0   0:00.19 migration/3
   16 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 migration/3
   17 root      20   0     0    0    0 S  0.0  0.0   0:00.68 ksoftirqd/3
   18 root      RT   0     0    0    0 S  0.0  0.0   0:00.07 watchdog/3
   19 root      20   0     0    0    0 S  0.0  0.0   0:47.72 events/0
   21 root      20   0     0    0    0 S  0.0  0.0   0:56.47 events/2
   22 root      20   0     0    0    0 S  0.0  0.0   0:20.01 events/3
   23 root      20   0     0    0    0 S  0.0  0.0   0:00.00 cgroup
```

图 2　测试完成后，服务器资源占用情况

随着迭代地进行，socket 连接/断开，连接期间资源占有率高，断开期间资源占有率低。场景结束后，资源得到释放。

用例 2（长连接并发 5000）的测试结果分析

- 事件通过率 100%和平均响应时间 1.5s

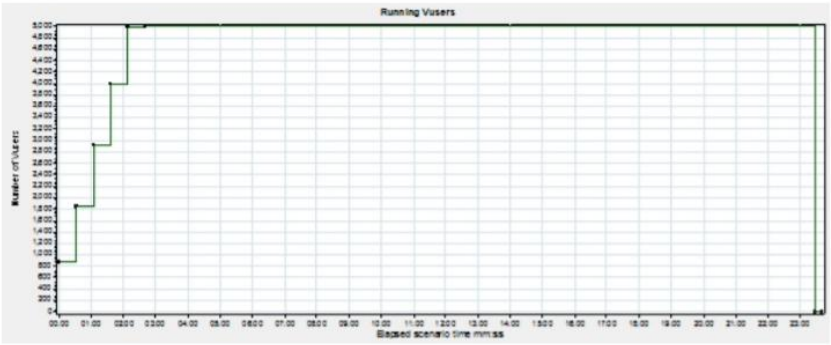
Performance Overview

Measurement	Value
Run Name	res.lrr
Weighted Average of Transaction Response Time	1.5
Total Passed Transactions	55000
Total Failed Transactions	0
Transactions Success Rate, %	100
Total Errors per Second	0
Total Errors	0

Transaction Summary

Run Name	Transaction Name	Minimum	Average	Maximum	Std. Deviation	90%	Pass Count	Fail Count	Stop Count
res.lrr	Conn_1	0.2	0.5	5	0.8	1.6	5000	0	0
res.lrr	Trans_1	0.1	2.5	5.6	1.8	5.3	50000	0	0

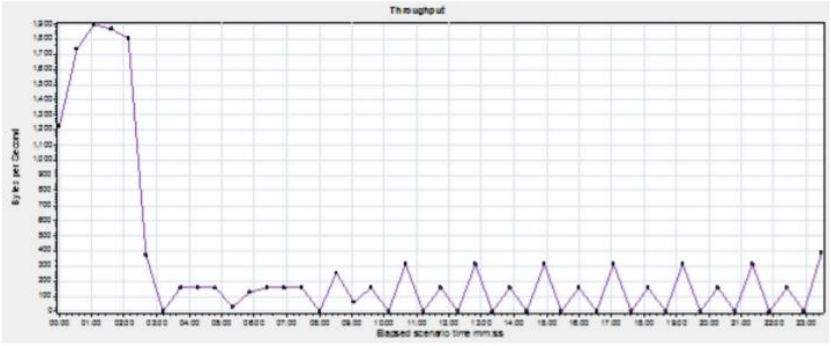
- 虚拟用户数曲线



Color	Scale	Measurement	Graph Minimum	Graph Average	Graph Maximum	Graph Median	Graph Std. Deviation
	1	Run	0	2455.25	5000	2936	1950.177

虚拟用户从 0 开始增长，增长到 5000 后，开始进行迭代操作。

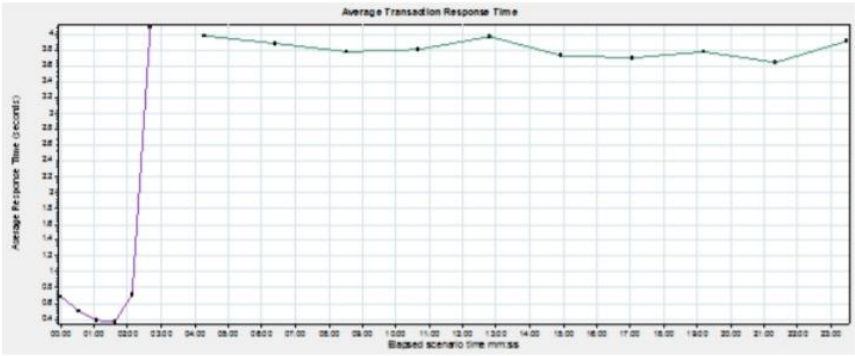
- 系统吞吐率曲线



Color	Scale	Measurement	Graph Minimum	Average	Graph Maximum	Graph Median	Graph Std. Deviation
	1	Throughput	0	302.604	1898.75	156.25	516.247

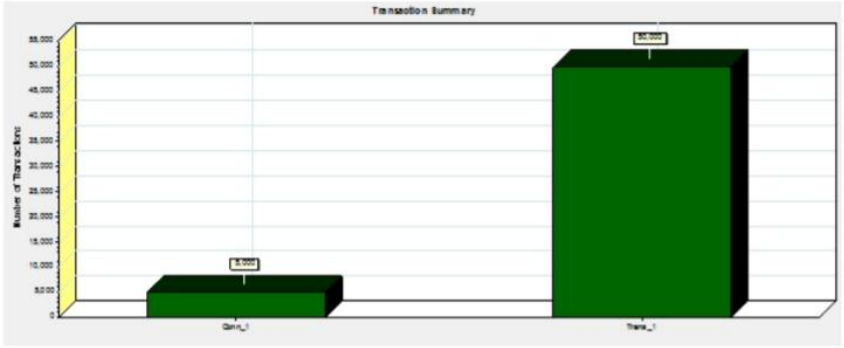
由于长连接是先建立连接，然后不停迭代收发数据，所以开始时吞吐率高，后续较为平稳。

- 事件响应时间



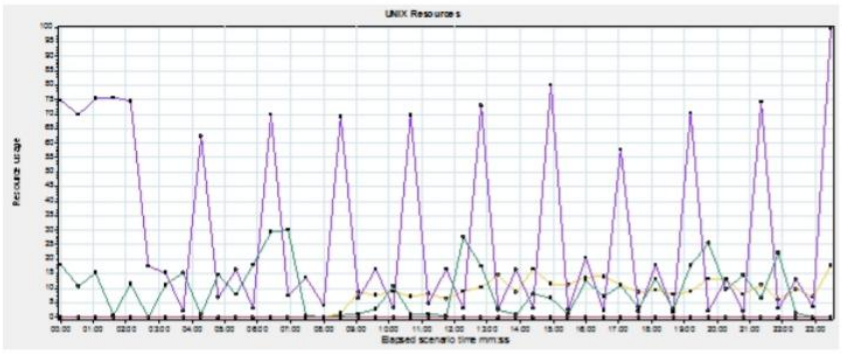
Color	Scale	Measurement	Graph's Minimum	Graph's Average	Graph's Maximum	Graph's Median	Graph's Std. Deviation
	1	Conn_1	0.369	1.125	4.097	0.684	1.336
	1	Trans_1	3.642	3.82	3.982	3.811	0.11

- 事件概览



Color	Scale	Measurement
	1	Pass

- 系统资源占用情况



Color	Scale	Measurement	Minimum	Average	Maximum	Std. Deviation
	100	Average load (Unix Kernel Statistics):10.5.0.56	0	0.064	0.238	0.058
	10	CPU Utilization (Unix Kernel Statistics):10.5.0.56	0.042	2.898	27.441	5.334
	10	Disk Traffic (Unix Kernel Statistics):10.5.0.56	0	0.932	12.106	2.279
	1	Paging rate (Unix Kernel Statistics):10.5.0.56	0	0	0	0

图 1　整个测试过程中服务器资源占用情况

```
top - 08:02:53 up  7:12,  5 users,  load average: 0.01, 0.01, 0.00
Tasks: 178 total,   1 running, 175 sleeping,   2 stopped,   0 zombie
Cpu(s):  0.5%us,  1.3%sy,  0.0%ni, 98.0%id,  0.1%wa,  0.0%hi,  0.1%si,  0.0%st
Mem:  14250876k total,  4025448k used, 10225428k free,   195240k buffers
Swap:  7192568k total,        0k used,  7192568k free,  2329100k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 9575 root      20   0 6177m 491m  13m S 11.7  3.5  30:00.10 java
 3129 root      20   0 1787m  51m 3004 S  2.0  0.4   4:11.07 cmf-agent
 3390 root      20   0  598m  21m 7924 S  2.0  0.2   1:32.79 python2.6
21143 root      20   0 15032 1176  832 R  2.0  0.0   0:00.01 top
    1 root      20   0 19364 1536 1228 S  0.0  0.0   0:02.63 init
    2 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kthreadd
    3 root      RT   0     0    0    0 S  0.0  0.0   0:00.04 migration/0
    4 root      20   0     0    0    0 S  0.0  0.0   0:00.41 ksoftirqd/0
    5 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 migration/0
    6 root      RT   0     0    0    0 S  0.0  0.0   0:00.06 watchdog/0
    7 root      RT   0     0    0    0 S  0.0  0.0   0:00.16 migration/1
    8 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 migration/1
    9 root      20   0     0    0    0 S  0.0  0.0   0:00.56 ksoftirqd/1
   10 root      RT   0     0    0    0 S  0.0  0.0   0:00.06 watchdog/1
   11 root      RT   0     0    0    0 S  0.0  0.0   0:00.21 migration/2
   12 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 migration/2
   13 root      20   0     0    0    0 S  0.0  0.0   0:01.90 ksoftirqd/2
   14 root      RT   0     0    0    0 S  0.0  0.0   0:00.07 watchdog/2
   15 root      RT   0     0    0    0 S  0.0  0.0   0:00.20 migration/3
   16 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 migration/3
   17 root      20   0     0    0    0 S  0.0  0.0   0:00.72 ksoftirqd/3
   18 root      RT   0     0    0    0 S  0.0  0.0   0:00.07 watchdog/3
   19 root      20   0     0    0    0 S  0.0  0.0   0:49.50 events/0
   20 root      20   0     0    0    0 S  0.0  0.0   0:52.08 events/1
   21 root      20   0     0    0    0 S  0.0  0.0   0:56.65 events/2
   22 root      20   0     0    0    0 S  0.0  0.0   0:20.23 events/3
   23 root      20   0     0    0    0 S  0.0  0.0   0:00.00 cgroup
   24 root      20   0     0    0    0 S  0.0  0.0   0:00.00 khelper
   25 root      20   0     0    0    0 S  0.0  0.0   0:00.00 netns
   26 root      20   0     0    0    0 S  0.0  0.0   0:00.00 async/mgr
   27 root      20   0     0    0    0 S  0.0  0.0   0:00.00 pm
   28 root      20   0     0    0    0 S  0.0  0.0   0:00.22 sync_supers
```

图 2　测试完成后，服务器资源占用情况

测试期间资源占有率随迭代进行平稳变化，在服务器可承受范围内，场景结束后资源也都释放掉了。

五、 测试结论

无论是短连接并发 5000 用户迭代 10 次访问，还是长连接并发 5000 用户迭代 10 次访问，事件成功率都是 100%，事件平均响应时间分别是 27.6s 和 1.5s 均在可接受范围内，服务器资源占用情况比较稳定，没有出现系统崩溃内存溢出等异常情况，并且测试结束后，所有资源都释放掉了。

由此可见，平台可以承受试运行期间的数据压力。

后记

至此，小 A 的第一个性能测试项目完成了。课件软件测试的基本流程不管是对于功能测试还是性能测试都是适用的。测试的思路都是一样的，只是侧重点和工具、技术会有所不同罢了。

项目经理眼中的项目计划

◆ 作者：安晨

摘要：

参与过公司产品或者项目开发的人员都应该知道，需求变更几乎是无法避免的，也是常见之事，而能否更合理评估项目工作量对项目能否正常上线和项目质量至关重要。那么，究竟有没有完美的项目计划，能让项目正常按质按时上线呢？

本文将为你讲述项目经理眼中的项目计划究竟是怎样的。

试问，项目成员是否会觉得以下的场景似曾相识，曾经的争论也历历在目：

场景 1：

图 1

场景 2：

图 2

对于图 1 和图 2 的情形，项目经理一定很纠结，经历多了是不是都有点开始怀疑人生了。但往往在绝望时也会想：在项目计划时，项目经理能否做得更好更多呢？

笔者带过的项目基本是这种流程节奏的：

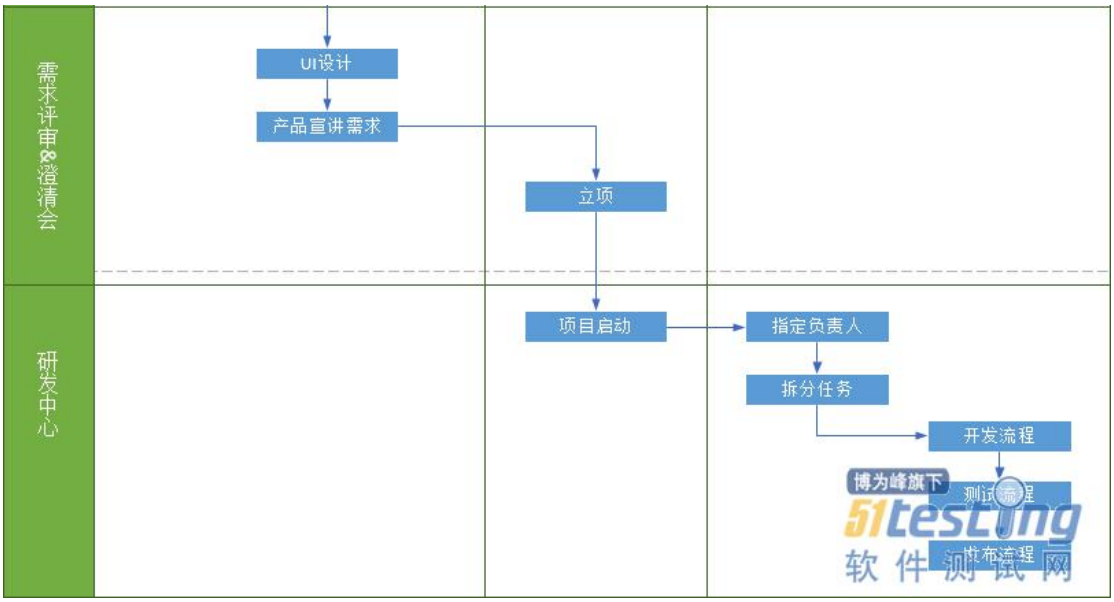


图 3

这种固定节奏下，似乎项目经理存在感略低？可现实却并非如此。本文将从以下几个方面和大家分享，项目经理眼中的项目计划，以及项目经理在整个计划过程的重要作用。

确认版本排期

鉴于我们公司的模式，目前大部分需求都有固定的上线时间，但需求工作量大小需要开发和测试评估确定，这时项目经理根据版本固定节奏初步确定可纳入的版本，是常规迭代版本（固定上线时间）还是独立项目。在排期前，项目经理需将版本节奏同步给各团队。对于确定版本周期一般遇到以下几种情况：

1.单个项目：如果刚刚上线完一个大项目，若有小的优化功能未上线，或者上线后有一些比较影响体验的问题需要修复，那么临近的常规迭代版本就可以考虑上线这类优化需求。

2.运营活动推广：在确定版本周期时需要考虑运营活动类需求，并且确保 APP 端通过审核的时间也能够满足运营推广需求。

3.多团队合作：从目前我们公司需求排期的情况看，大部分需求都是跨多开发团队协作完成的，那在确定排期前就需要考虑多个团队的资源安排。

完善和优化初定项目计划

对一个项目来说，仅靠开发和测试给的时间节点制定项目计划是远远不够的。还有以下很多种情况常常被忽视：

1.分批提测：目前我们公司仍然习惯单一项目统一提测，这对于项目的风险控制是非常不利的。因此项目经理需和开发、测试梳理，是否可将功能分批提测，这样测试就能提前介入，提前进行后端接口测试等，进而缩短项目周期，必要时引入自动化测试进行冒烟，进一步提升效率。

2.需求冻结：带项目这么久，至今还未遇过一个不吐槽需求变更的团队。对于需求变更，我们不能完全拒绝，要学会拥抱变化。但也不是任何时间点都能接受需求变更的，在某个时间点后提出的需求变更就会影响项目按期上线，那我们就有可能拒绝。这个时间点就是需求冻结点。考虑这个时间的合理性，我们约定的需求冻结点是最后一个提测点的后一天。

3.部门相关需求：有些跨部门需求是相关的，上线时间和资源冲突，或者业务相关，这些在需求澄清时就应知会多团队，提前做好相关工作。

4.数据埋点：一直以来埋点都被认为不太重要的工作，开发自测即可，但有时埋点也有可能导致软件崩溃；又或者认为埋点是运营部门的需求，当前需求情况暂不需埋点，所以不够重视。因此项目经理在制定计划时，也需考虑埋点工作。

5.代码冻结：所谓代码冻结意味着代码不允许改动了，在最后时间匆忙改动代码有可能引入未知 bug 且未经测试回归，可能导致无法按时上线。目前情况仍无法引入代码冻结环节，以后待整体项目较稳定后再考虑增加。

合理的人员安排

1.合理分配工作量：当需求澄清时，开发和测试人员都应该评估需求工作量，排期纳入项目后，及时拆分任务和更新进度。开发和测试人员经常会忽视这部分，所以项目经理要帮助项目成员更好地在整个项目中合理分配工作量。

2.多项目并行：目前项目情况来看，项目并行情况并未减轻。当项目经理发现，开发和测试当前任务已饱和时，首先要和产品经理沟通优化项目计划，若上线时间不可改变，则需考虑调配其他人员支持。因为项目经理比具体的成员更了解全局优先级，可考虑将低优先级项目的人员临时支持高优先级的项目，当然先要和团队负责人达成统一意见。

同步项目计划

项目排期后，要确保项目人员都能够获知整个项目计划。一般情况，大部分人只会关心自己的任务，很少关心其他成员的工作，所以项目经理需帮大家补足这部分。那么除了研发团队还有哪些角色会关心呢？

1.需求提出方（包含外部业务方）：这类人会非常关心项目计划，包含需求范围，以及具体的上线时间。

2.风险部人员：风险同事需要根据项目计划确认产品风险需求和规则配置，以及协调部门内部其他需求排期。

3.运营人员：运营同事需要根据项目计划确认版本可上线活动内容，以及结合需求紧急度安排运营推广活动。

4.运维人员：与前面几个角色相比，运维同事更容易被忽略，他们需要提前获知项目上线时间，以及上线系统清单，安排人员支持等。

与外部团队合作的项目，还要把项目计划同步通知合作方，做好沟通协调，以免双方时间不一致。

其他影响

项目难免会遇到假期，因此在排期及制定项目计划时，须提前考虑并获知项目成员休假情况。另外，项目经理还须考虑长假前后工作效率问题，这也带来项目风险，所以计划阶段就应全面考虑。

综上所述，项目经理在项目计划制定过程并不可少，但也并非考虑到上述提及的情况，就能制定出完美的项目计划，即使有完美的项目计划也未必确保项目顺利上线。因此，需要根据项目背景的不同，考虑不同的执行方式；并在项目过程中不断总结经验，反思项目过程问题能否在计划阶段就避免发生，形成项目的 checklist，并在下次做项目计划时多注意。最后需要提及一点，笔者的经验仍非常有限，文中提及的内容并非一定适合其他公司的项目，欢迎大家批评指正和交流心得体会。

❖ 拓展学习

■PM 必备知识,基于 Jenkins 的持续集成测试管理和实：http://www.atstudy.com/course/444

CentOS 上 Nginx 服务器安装 phpmyadmin

◆ 作者：zhengchenhappy

下载并安装 phpmyadmin 工具，一个是把代码拷上去,一个是增加一个站点(就是 nginx conf 文件)。

phpMyAdmin 是一个以 PHP 为基础，以 Web-Base 方式架构在网站主机上的 MySQL 的数据库管理工具，让管理者可用 Web 接口管理 MySQL 数据库。

phpmyadmin 最大的好处之一就是提供虚拟主机的服务商不需要向外界公开 mysql 的地址，

让数据库操作平台直接连接其 LAN 的 IP 地址——既提供了操作数据库的服务，又保证了安全。

另一个好处就是跨平台，Mysql 有很多优秀的 GUI 管理工具，但是，桌面程序都有一个跨平台成本大的通病，phpmyadmin 是 web UI，可以很好的屏蔽这一点，而且免费的。

下载 phpMyAdmin-4.6.4-all-languages 安装包，解压后上传至 nginx 默认 web 路径即 /usr/share/nginx/html。

Nginx 虚拟主机配置（/etc/nginx/conf.d）如下：拷贝除 default 以外的任一配置文件，然后进行修改。

```
server {
listen          82;             //访问的端口
server_name   192.168.xxx.xxx;
index index.html index.htm index.php;
root /usr/share/nginx/html/phpmyadmin;

# root/usr/share/nginx/html 后添加安装的文件名，这样打开 url 就可以简化成
# http://192.168.xxx.xxx:82 即可，否则的话需要输入 url 为
# http://192.168.xxx.xxx:82/phpmyadmin
location ~ \.php$ {
#       root            html;
fastcgi_pass     127.0.0.1:9000;
fastcgi_index   index.php;
fastcgi_param   SCRIPT_FILENAME   /$document_root$fastcgi_script_name;
include          fastcgi_params;
```

}

phpmyadmin 登录成功后，如下图：

解决方法：

升级 mysql 版本为 5.5.0 以上。

192.168. xxx.xxx 上解压 phpmyadmin 到 var/www/html 路径下后，打开浏览器

提示：

PHP 5.5+ is required.
Currently installed version is: 5.4.45

解决办法：

删除已装的 phpmyadmin，降低版本为 4.5.0，即可打开了。

最近开始做网页性能测试相关的，想分享一些怎样提升性能以及怎样去做测试，希望对大家有用。

有了 XPath，再也不用担心元素定位了

作者：王练

1. 摘要

使用 Selenium 进行自动化测试，一大难题就是元素定位。虽然 Selenium 支持使用 ID、name、Class、链接文字等多种方式进行元素识别，但对于复杂的网页，采用富客户端框架的前台界面，部分界面元素特征的稳定性无法保证，导致识别失败。如果生搬硬套使用 ID、name 等方式，会导致自动化脚本可用率降低，增加了投入成本。

本文推荐使用 XPath 定位方式，介绍 XPath 定位的几种方式：绝对路径、相对路径、索引号、属性值、文本内容和 Axis，其中属性值和文本内容说明模糊匹配的使用，Axis 方式说明 Axis 嵌套的方法。在多种 XPath 定位方式中，推荐使用相对路径、属性值、文本内容和 Axis 组合的方式，并给出界面识别的推荐规则。

通过 XPath 的组合，让你的自动化测试不再担心元素定位问题。

2.关键字

自动化测试 Selenium XPath

3.XPath 简介

XPath 不是 Selenium 专用，只是作为一种定位手段，为 Selenium 所用。XPath 是一门在 XML 文档中查找信息的语言。XPath 可用来在 XML 文档中对元素和属性进行遍历。由于 HTML 的层次结构与 XML 的层次结构天然一致，所以使用 XPath 也能够进行 HTML 元素的定位。

本文使用的 HTML 示例源码如下。

```
<!DOCTYPE html>
<html>
    <body>
        <div id="div1">
          <div id="div1-1">
            <input id="0" name="sogou_input" ></>
            <a href="http://www.sogou.com">搜狗搜索1</a>
          </div>
          <div id="div1-2">
            <input id="1" name="sogou_input"></input>
            <a href="http://www.sogou.com">搜狗搜索2</a>
            <img alt="div1-img1" src="http://www.sogou.com/images/logo/new/sogou.png" href="http://www.sogou.com">搜狗图片</img>
            <input id="2" type="button" value="查询"></>
          </div>
        </div>
        <br>
        <div name="div2">
            <input id="0" name="baidu_input"></input>
            <a href="http://www.baidu.com">百度搜索</a>
            <img alt="div2-img2" src="http://www.baidu.com/img/bdlogo.png" href="http://www.baidu.com">百度图片</img>
        </div>
    </body>
</html>
```

Web 界面如下。

4.绝对路径定位

顾名思义，将 XPath 表达式从 HTML 的最外层节点，逐层填写，最后定位到要操作的元素。

以定位第一个 img 元素为例，可以使用如下 XPath 表达式：

/html/body/div/div/img

语句以”/”开始，含义为从”html”节点开始寻找，逐层寻找”body/div/div/img”。通过 FirePath 查看结果如下。

通过绝对路径定位的元素， 一旦路径有变化会导致定位失败。而且完全能够通过路径定位到的元素是比较少见的，都需要增加索引、属性等方式。所以不推荐使用该方式。

5.相对路径定位

绝对路径与相对路径的差别与文件系统中的绝对和相对路径类似，相对路径是只给出元素路径的部分信息，在 HTML 的任意层次中寻找符合条件的元素。

还是以定位第一个 img 元素为例，通过相对路径可以使用如下 XPath 表达式：

//div/div/img

语句以”//”开始，含义为在任意层次下寻找路径层次为”div/div/img”的元素。通过 FirePath 查看结果如下。

通过相对路径定位元素，提取的是元素的部分特征，只要提取恰当，能够保证版本间稳定，是进行自动化测试的首选。示例中通过相对路径直接定位，实际使用中会结合属性等其他特征，共同定位。

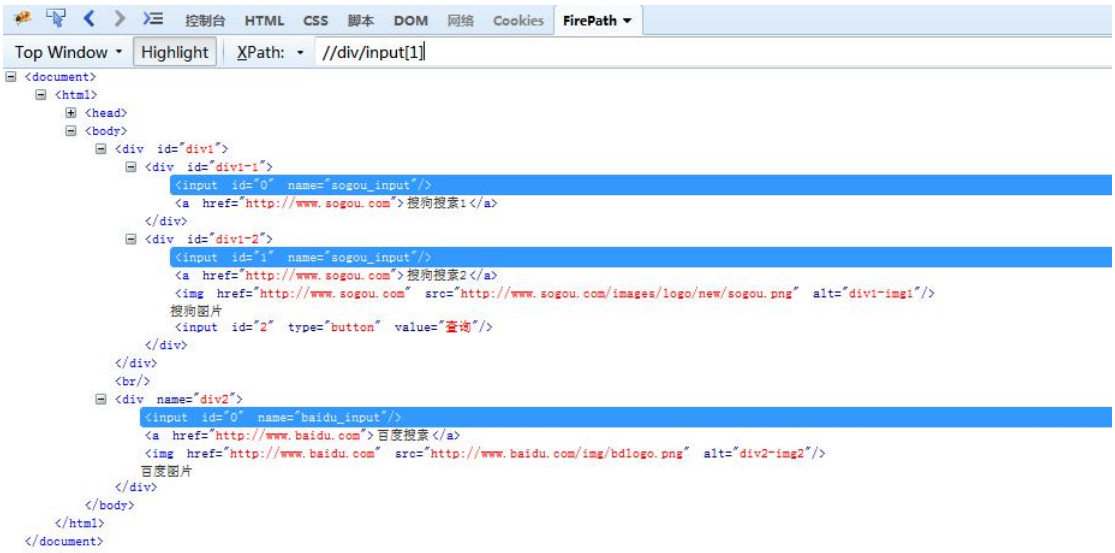
6.索引号定位

索引号与元素属性 id 无关。通过索引号定位是指定元素在父节点下的出现序号，通过序号进行定位的方式。

以期望定位第一个 input 元素为例，表达式如下：

//div/input[1]

该 XPath 是相对路径和索引号组合的方式，含义是任意层次下的”div/input”，其中 input 是”div”下索引是 1 的 input。



不幸的是，符合这个条件的 input 有三个，FirePath 查看结果如上。通过这个例子可以看到，界面元素的 id 属性与索引号无关；相对路径或者绝对路径可以与索引号的方式组合进行定位。

索引号定位的方法，类似于 QTP 定位通过 index 定位 Java Swing 对象，当界面出现变动时，很难保证索引号的稳定。所以不推荐使用。

7.属性值定位

7.1 属性值定位

属性值定位是指通过界面元素属性值的描述进行该元素的定位。

以定位第一个 input 为例，表达式如下：

//input[@id='2']

表达式为相对路径与属性值的组合，含义为在任意层次下，寻找”id”属性值为”2”的 input。FirePath 查看结果如下。

属性值定位类似于 QTP 的描述性编程，QTP 的描述性编程就是利用 Java Swing 属性的值进行 Java 对象的定位。区别在于 QTP 的识别永远是绝对路径，路径的中间层次可以省略，但起始一定是根对象。

通过相对路径和属性值的方式可以定位绝大部分页面对象，但对于富客户端的情况有些棘手。虽然表达式定位的元素是唯一的，但富客户端的界面部分属性值时动态生成的，每次运行会出现不同，导致识别失败。建议使用前通过多次实验进行确定，必要时联系开发确保属性值的稳定性。

7.2 属性值模糊匹配定位

属性值除了全匹配的方法外，还支持模糊匹配。

定位第一个 input 为例，表达式可以写为：

//input[contains(@name,'baidu')]

含义为在任意层次下，寻找”name”属性值包括”baidu”的 input。FirePath 查看结果如下。

模糊匹配的函数包括如下：

XPath 函数	表达式示例	含义
starts-with	//input[starts-with(@name,'sogou')]	属性值以”sogou”开始的 input 元素
contains	//input[contains(@name,'sogou')]	属性值包含”sogou”的 input 元素

这两个函数是 XPath 语法中的字符串函数，XPath 的字符串函数还有很多（如 ends-with、matches 等），但 HTML 还不支持。不过有这两个已经能解决很多问题了。

FirePath 中试用函数时，表达式显示为红色表示语法错误，白色但定位不到元素，即没有符合条件的元素。

8.文本内容定位

8.1 文本内容定位

文本内容定位是利用 HTML 的 text 字段进行定位的方法，可以看做是属性值定位的衍生。

以定位界面的”百度搜索”按钮为例，表达式如下：

//a[text()='百度搜索']

含义是定位 HTML 文本为”百度搜索”的链接元素。FirePath 查看结果如下。

由于”百度搜索”这几个字是浏览器原始界面就可以看到的，我们称为“所见即所得”，这种特征改的可能性非常小，所以非常稳定，优先推荐使用。通过这种“所见即所得”的方式可以借鉴属性值定位中的富客户端问题。

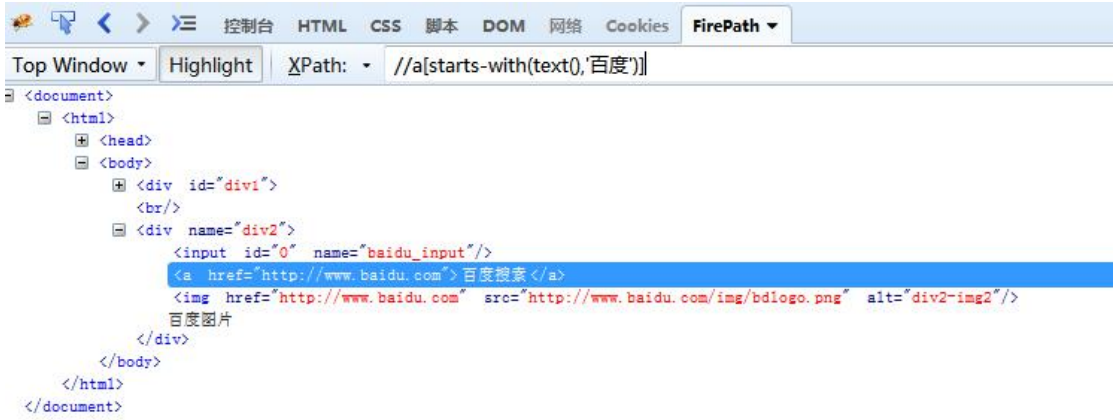
8.2 文本内容模糊匹配定位

与属性值类似，文本内容也支持 starts-with 和 contains 模糊匹配。

上述”百度搜索”按钮的表达式可以修改如下：

//a[starts-with(text(),'百度')]

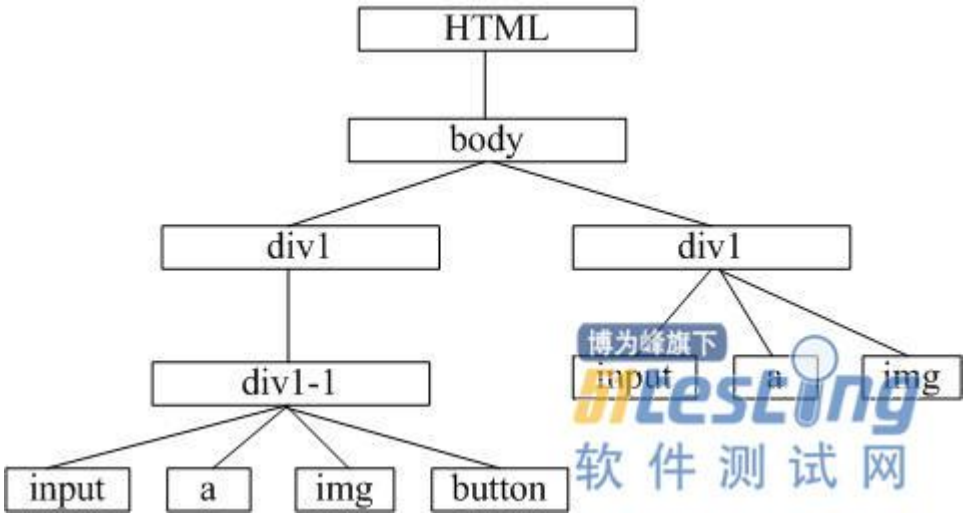
含义是定位界面上 text 以”百度”开头的链接元素。FirePath 查看如下：



9.Axis 定位

Axis 中文是“轴”，在 XPath 语法中用 Axis 来定义相对于当前节点的其他节点集合。先定位到一个好寻找的元素，通过它的 Axis 定义与目标元素的相对位置，从而完成唯一目标元素。

Axis 是通过父子关系进行相对位置的描述，示例的 HTML 层次关系如下图，可以看到任意一个节点都可以通过父子关系与其他节点产生关系。



以定位第一个 input 为例，不希望使用 name 属性值，可以通过如下表达式定位：

//a[text()='搜狗搜索 2']/preceding-sibling::input

首先定位 text 为搜狗搜索的链接，然后寻找他的前面同级的 input 元素。其他支持的 Axis 以及示例如下。

Axis	作用	示例	含义
ancestor	选取当前节点的所有先辈（父、祖父等）	//a[text()='百度搜索']/ancestor::div	text 为”百度搜索”的链接，其先辈 div 元素
ancestor-or-self	选取当前节点的所有先辈（父、祖父等）以及当前节点本身。	//a[text()='百度搜索']/ancestor-or-self::a	text 为”百度搜索”的链接，其先辈或自身链接元素
attribute	选取当前节点的所有属性。	//a[text()='百度搜索']/attribute::href	text 为”百度搜索”的链接，其”href”属性
child	选取当前节点的所有子元素。	//a[text()='搜狗搜索 2']/ancestor::div/child::input[@type='button']	text 为”搜狗搜索 2”的链接，首先查找先辈 div，在任意先辈下定位其子 input 元素，该元素 type 属性为 button
descendant	选取当前节点的所有后代元素（子、孙等）。	//a[text()='搜狗搜索 2']/ancestor::div/descendant::input[@type='button']	text 为”搜狗搜索 2”的链接，首先查找先辈 div，在任意先辈下定位其后代 input 元素，该元素 type 属性为 button
descendant-or-self	选取当前节点的所有后代元素（子、孙等）以及当前节点本身。	//a[text()='搜狗搜索 2']/ancestor::div/descendant-or-self::a[contains(text(),'搜索 1')]	text 为”搜狗搜索 2”的链接，首先查找先辈 div，在任意先辈下定位其后代或自身链接元素，该元素 text 包含”搜索 1”关键字。
following	选取文档中当前节点的结束标签之后的所有节点。	//a[text()='搜狗搜索 1']/ancestor::div/following::input[contains(@type,'button')]	text 为”搜狗搜索 1”的链接，首先查找先辈 div，在任意先辈下定位其后续 input 元素，该元素 type 属性包含”button”关键字。
following-sibling	选取当前节点之后的所有同级节点。	//a[text()='搜狗搜索 2']/following-sibling::input[@type='button']	text 为”搜狗搜索 2”的链接，查找该节点之后的兄弟节点，该元素 type 属性为”button”。
parent	选取当前节点的父节点。	//a[text()='搜狗搜索 2']/parent::div/child::input[@name='sogou_input']	text 为”搜狗搜索 2”的链接，首先查找父节点 div，在该父节点下定位子节点 input，该元素 name 属性的值为 sogou_input。
preceding	选取文档中当前节点的开始标签之前的所有节	//a[text()='搜狗搜索 2']/preceding::div/child::i	text 为”搜狗搜索 2”的链接，首先查找之前节点

	点。	nput[@name='sogou_input']	div，在该节点下定位子节点 input，该元素 name 属性的值为 sogou_input。
preceding-sibling	选取当前节点之前的所有同级节点。	//a[text()='搜狗搜索 2']/preceding-sibling::input[@name='sogou_input']	text 为”搜狗搜索 2”的链接，定位该节点之前的兄弟节点，该元素 name 属性的值为 sogou_input
self	选取当前节点。	//a[text()='搜狗搜索 2']/self::a/preceding-sibling::input[@name='sogou_input']	text 为”搜狗搜索 2”的链接，首先定位当前节点连接元素，之后定位该节点之前的兄弟节点，该元素 name 属性的值为 sogou_input

从上述示例中，可以得出如下结论：

1.ancestor 示例给出了相对路径、文本内容、Axis 的组合示例。可以通过相对路径、文本内容、Axis 组合进行界面元素定位。

2.child 示例给出了相对路径、文本内容、Axis、属性值以及 Axis 嵌套的组合示例。可以通过相对路径、文本内容、Axis、属性值组合进行界面元素定位，Axis 支持嵌套的方式定位。

3.descendant-or-self 示例给出了相对路径、文本内容、Axis 以及文本内容模糊匹给出了配的组合示例。可以通过相对路径、文本内容、Axis、以及文本内容模糊匹配组合进行界面元素定位。

4.following 示例相对路径、文本内容、Axis、属性值模糊匹配的示例。可以通过相对路径、文本内容、Axis、属性值模糊匹配的方式进行定位。

相对路径、文本内容和模糊匹配、属性值和模糊匹配、Axis 和 Axis 嵌套可以根据需要任意组合，组成”丰富多彩”的表达式，完成元素定位的任务。

对比 QTP，Axis 类似于 QTP 的相关对象识别（Visual Relation Identifiers，简称 VRI）。QTP 的 VRI 就是依靠界面的其他对象与目标对象的位置关系进行定位的方法。由于 HTML 的规范性，XPath 的 Axis 更强大、更可靠。通过 Axis 和 Axis 嵌套的方式，可以更稳定的完成 QTP VRI 的 Left、Up、Near 等任务。

10.总结

XPath 的功能非常强大，不仅能够完成界面定位的任务，而且能保证稳定性。实际自动化测试中，能够识别界面元素是重要的，更重要的是要保证版本间的稳定性，减少脚本的维护工作。

如下规则请参考：

1.特征越少越好；

2.特征越是界面可见的越好；

3.避免使用绝对路径；

4.避免使用索引号；

5.使用属性值定位，对于富客户端要慎重；

6.相对路径、属性值、文本内容、Axis 可以任意组合，当然属性值和文本内容的模糊匹配也支持和上述方式任意组合，Axis 可以嵌套使用。

通过 XPath 的各种方式组合，能够解决 Selenium 自动化测试中界面定位的全部问题。

可以说：有了 XPath，再也不用担心元素定位了。

《51 测试天地》（五十）上篇 精彩预览

- 假如给你一个测试团队
- 新任 Leader 的痛点和套路
- 一体化测试管理
- 也谈国际化测试
- Jenkins+SonarQubeda 搭建代码质量管理平台
- 如何进阶成为测试 Leader
- 数据统计测试小结

马上阅读