功能测试 2 年

时间：2016 年 7 月 22 日　　作者：菜鸟 C

写在前面

一些闲话，不喜欢看的朋友可以自行跳过。

在开始正文之前，请允许我先谈一谈我对测试这个职业的看法。从入行到现在也有两年差不多的时间了。最开始，是因为对代码不感兴趣，想混口饭吃，于是选择了测试这个职位。但是随着深入的了解，我慢慢的喜欢上了这份工作。

我们来看看国内的测试现状：

1）**缺少时间**

测试团队介入较晚，很多都是产品和程序已经实现了产品的大部分基础功能后才开始组织测试，编写测试用例的时间极为稀少。

2）**维护困难**

需求变动频繁，变动量大，随之而来的测试用例变动也会频繁和巨大，因此许多团队放弃制作和更新测试用例。

3）**急于发现 Bug**

测试用例需要长期制作和维护才可体现其作用，而目前大多数测试团队都急于找到 Bug，当执行完一遍测试用例后发现没有多少新 Bug，从而开始怠慢测试用例的制作不更新。

4）**缺乏专业知识**

不可否认的，目前软件也好，游戏也罢，测试的从业人员很多，但是技术水平参差不齐，甚至有些同学的对于测试用例的制作方法都了解甚少。

5）**不被重视**

这是我听到的最多的抱怨，在公司怎么样，没有话语权，我们必须承认，在国内这个大环境下，很多公司都是这样。

上面讲的这些，可能大家都有感受到。但是目前并没有什么解决的好方法，一个职业的发展是漫长的，只能说在这个 IT 行业飞速发展的今天，只要大家努力，总有一天会改变这些问题。

关于这个职业，尽管它不能带给我高薪，尽管人们都称呼我们为背锅侠。但是无论如何我都热爱着这个职业。每一个从我手中走出去的项目，就像是自己的孩子一样。我很庆幸，我还没有麻木，我会为我的漏测，我的失误自责。

有些人会问，如果做测试需要一些什么技能。我会告诉你，会使用电脑就可以。不过，你得去做，去学。不会的可以学，但是不学的，就实在没办法了。笔者，还有一些朋友，面试过一些同学。我们都喜欢态度诚恳的初学者，而不是油腔滑调的老油条。

如果你没有一颗强大的内心，如果你不能够承受日复一日的做着同一件事，如果你没有足够的耐心，那么我不推荐你选择去做测试。麻烦出门右转，谢谢。

送给初学者的一些忠告：

1）除了自己，不要相信任何人。在工作中，除了自己谁也不能相信，只要不是自己经

手的东西，一定不要说 OK。

2）不打无准备之仗。无论做什么工作，一定要有计划有目标的去做，做好相应的准备才能轻松获胜。

3）测试的字典里，没有差不多。不要说什么差不多，测试的世界，只有合格与不合格。

4）永远都要坚守自己的原则。无论什么时候，都要坚守自己的原则，你手里握着的整个不仅仅是这个项目，更是整个公司的信誉。

5）不要好高骛远。专心去做一件事，做好。不要看到这个想学，那个也想做。听说过测试山羊吗？山羊每次只迈一步，所以才能无惧陡峭的山壁。

最后，我想再次提醒大家，时刻牢记着我们的职责。

测试概述

基本定义

软件测试（英语：**Software Testing**），描述一种用来促进鉴定软件的正确性、完整性、安全性和质量的过程。换句话说，软件测试是一种实际输出与预期输出间的审核或者比较过程。软件测试的经典定义是：在规定的条件下对程序进行操作，以发现程序错误，衡量软件质量，并对其是否能满足设计要求进行评估的过程。

测试目的

- 测试是为了发现程序中的错误而执行程序的过程。
- 好的测试方案是极可能发现迄今为止尚未发现的错误的测试方案。
- 成功的测试是发现了至今为止尚未发现的错误的测试。
- 测试并不仅仅是为了找出错误。通过分析错误产生的原因和错误的发生趋势，可以帮助项目管理者发现当前软件开发过程中的缺陷，以便及时改进。
- 通过分析帮助测试人员设计出有针对性的测试方法，改善测试的效率和有效性。
- 没有发现错误的测试也是有价值的，完整的测试是评定软件质量的一种方法。
- 根据测试目的的不同，还有回归测试、压力测试、性能测试等，分别为了检验修改或优化过程是否引发新的问题、软件所能达到处理能力和是否达到预期的处理能力等。

测试内容

软件测试主要工作内容是验证（**verification**）和确认（**validation**），下面分别给出其概念：

验证(**verification**)是保证软件正确地实现了一些特定功能的一系列活动， 即保证软件以正确的方式来做了这个事件(**Do it right**)

- 确定软件生存周期中的一个给定阶段的产品是否达到前阶段确立的需求的过程。
- 程序正确性的形式证明，即采用形式理论证明程序符合设计规约规定的过程。
- 评审、审查、测试、检查、审计等各类活动，或对某些项处理、服务或文件等是否

和规定的需求相一致进行判断和提出报告。

确认（**validation**）是一系列的活动和过程，目的是想证实在一个给定的外部环境中软件的逻辑正确性。即保证软件做了你所期望的事情。（**Do the right thing**）

- 静态确认，不在计算机上实际执行程序，通过人工或程序分析来证明软件的正确性。
- 动态确认，通过执行程序做分析，测试程序的动态行为，以证实软件是否存在问题

软件测试的对象不仅仅是程序测试，软件测试应该包括整个软件开发期间各个阶段所产生的文档，如需求规格说明、概要设计文档、详细设计文档，当然软件测试的主要对象还是源程序。

上面这些内容是百度到的。好吧，必须承认，笔者对这些理论不是特别熟。这里之所以会百度这些放在这里的原因呢，是想提醒大家，要牢记我们的职责。从入行的时候起，就有前辈告诉我，身为一个测试，首要的核心工作就是保证产品质量。

从黑盒做起

测试这个职业门槛低，大部分人入门都是从黑盒做起。笔者从最初入行直到现在都是在做黑盒测试。从最开始 **Run Case**，到后来的需求分析、测试设计、测试准备、编写用例、执行测试、提交结果等等，也是经历了很长一段的时间。

大部分人都有些瞧不起黑盒测试，包括曾经的我。觉得黑盒并不存在什么技术含量，只是一味的点点点。看着人家，自动化，性能，安全的大神，拿着高薪羡慕不已。其实我们都走入了一个误区。黑盒的难度并不低，它只是门槛低。从一个测试的角度来讲，黑盒，它是从 UI 层上进行测试，涉及到功能，逻辑，模块的交互，接口的调用，端与端的交互，仅仅是从 UI 层上来进行测试，就要覆盖到这些，是非常考验测试人员的逻辑与用例的场景覆盖的。要保证到场景的覆盖，还要尽可能去模拟用户的真实场景，不仅费神而且费力。

需求分析

为什么要做需求分析

大家都知道我们会做需求分析，但是不知道大家有没有思考过为什么去做？

需求文档，作为黑盒测试中的重要参考文档。它所提供给我们的信息是巨大的。它不仅仅是对功能的描述，更是我们确定测试目标，测试范围的标准。通过需求分析，我们可以了解到，我们的目标，测试的范围，可能用到的技术、工具，以及测试的策略，可能存在的风险，进而做出相应的计划、方案，编写测试用例。

需求挖掘

测试，从这一刻已经开始了。没错，就是现在。

在我们拿到需求文档之后。我们不能仅仅是去阅读需求，更要去挖掘需求。挖掘潜藏的需求，挖掘需求中的缺陷，挖掘需求中的风险。

那么如何去挖掘呢？这似乎是一个问题。一遍又一遍的阅读需求。你没看错就是阅读。很枯燥吧，怕了吗？

我经常会拿到需求，阅读半天甚至一天，然后逐条记录，分析。我会去思考，为什么这样设计，如果我是用户，这样我会怎么想？这样设计带来的影响是什么？这样设计是否存在缺陷？

曾经我遇到过一个设计的缺陷，会造成与之前的功能冲突。（具体的东西不太方便讲出

来，希望见谅）这种情况很少见，但并不代表没有。我们在阅读文档的期间就发现，就节约了很多成本。一旦等到开发完成才发现 ，后果简直不敢想象。

不得不说一下，映客是相当成功的，虽然有一部分原因是人家是勇于吃螃蟹的人，但是从设计的角度来讲，也是相当聪明的。

笔者在前几天无聊的时候玩了一下映客。毕竟人家也是 AppStore 上最火的社交软件。也开了一下直播，在开播的时候会默认分享到朋友圈，然后会跳一下微信。我们来看一下这个页面。

这个功能，最初是让我有那么一点不爽的，但是实际上也并不会影响特别大。我还一直想不明白为什么会有这个设计，觉得有些莫名其妙。但是晚上微信上就有不少朋友问我，你还开直播啊，你还玩映客啊？下次直播我去看啊等等。

从一个测试的角度来看这个问题。我们如果首先看到易用性和体验性的层面，那么这个默认的跳去微信分享实在不是一个好的设计。凭空多了那么一步操作，会让用户觉得不舒服，但是这个不舒服影响的不是特别大。隐藏起来的是，无形的推广与用户的培养。培养的用户基数是不可想象的。用一个影响不是特别大的东西，换取的是无法想象的用户基数以及推广。这也是我佩服的地方。

测试计划&方案

为什么要设计测试计划&方案

可能很多人都接触过测试计划&方案，也可能很多人做过测试计划&方案。

有些同学可能分不清楚测试计划和测试方案的区别，在这里，给大家讲一下。

测试计划，就是一个大纲，它会告诉我们，每个阶段我们需要做什么以及我们的目标是什么，使我们明白项目是如何运作的。就像是海中的灯塔一样，指引着我们，不至于迷路。

测试方案，更像是细化的测试计划。如果说测试计划是海中的灯塔，那么测试方案就更像陆地上的路标。它更倾向于教导我们如何去做。

在正式测试开始之前，我们必须搞清楚的是，**when?what?who?how?**什么时候去做，做什么，谁去做，怎么去做。这就是测试计划&测试方案的作用。

如何设计测试计划&方案

笔者做计划和方案的次数也不多。但是不得不说，就算没有文档的输出，但是也要胸有成竹。

首先是测试计划，我们在做测试计划的时候，眼界一定要放宽，把视角调高，最好是上帝视角。通过需求文档了解我们的目标，然后进行工作量的评估，人员的安排，确定工具和技术。结合开发计划，进行时间的评估，进而对测试工作进行排期，必不可少的是要进行风险的评估，测试过程中可能遇到的问题，以及紧急应对措施都是必须考虑的。

那么我们来看看测试方案。测试方案，重点描述的每一个阶段的测试策略，参与人员，使用的技术，软硬件环境，需要的资源。它不需要有多全面，毕竟是用来指导测试人员如何去做的。所以我们一定要根据实际的情况去做方案，使用什么策略，使用哪些工具，需要哪些资源，如何去做，通过的标准。

请记得分配任务的时候，最好根据不同的人的强项来分配，了解清楚这个，可以提升不少效率哦。

相信大家看到这里，已经迫不及待的想去自己做一下测试计划和方案了。但是，这真的是一个经验积累的过程，并不是我这三言两语说的那么简单。

测试准备

你以为这就可以开始进行对程序的测试了吗？是不是忘了准备工具了？

在测试计划和方案完成之后，就要开始着手准备进行测试工作。

准备哪些东西呢？

- 测试环境（服务器，数据库，测试数据等）
- 测试使用的工具
- 做好技术储备

怎么，嫌烦了吗？也许这个准备看起来不起眼，但是在你开始进行测试之后，你会发现准备期的工作的好处。

搭建测试环境

上面我们说到了测试准备，在准备期间最重要的莫过于测试环境的搭建。本小节我们来看看如何搭建测试环境。

什么是测试环境

在开始搭建测试环境之前我们需要了解清楚，什么是测试环境，测试环境，包括哪些东西。

硬件环境搭建：指测试必须的服务器、客户端、网络连接设备以及答应机/扫描仪等辅助硬件设备所构成的环境；若要求的硬件配置种类较多，可以定义一些基本硬件配置；

软件环境搭建：指测试软件运行时的操作系统、数据库及其他应用软件构成的环境；

（1）共存软件对被测软件的影响：例如公用文件之间的相互影响、公用文件之间的内存冲突以及其他的影响；

（2）共存文件越少越好；在某些有特殊要求的测试中，共存文件必不可少；

利用辅测试环境进行的测试：

兼容性测试：在满足软件运行要求的范围内，可选择一些典型的操作系统和常用应用软件对其进行安装卸载和主要功能的验证

模拟真实环境测试：有些软件，特别是面向大众的商品化软件，在测试时常常需要考察在真实环境中的表现。如测试杀毒软件的扫描速度时，硬盘上布置的不同类型文件的比例要尽量接近真实环境，这样测试出来的数据才有实际意义

横向对比测试：利用辅测试环境“克隆”出完全一致的测试环境，从而保证各个被测软件平等对比

网络环境：指被测软件运行时的网络系统、网络结构以及其他网络设备构成的环境、网络设备、网络结构、网络系统等；

对测试环境的要求

（1）尽可能的真实的环境；

（2）符合软件运行的最低要求；

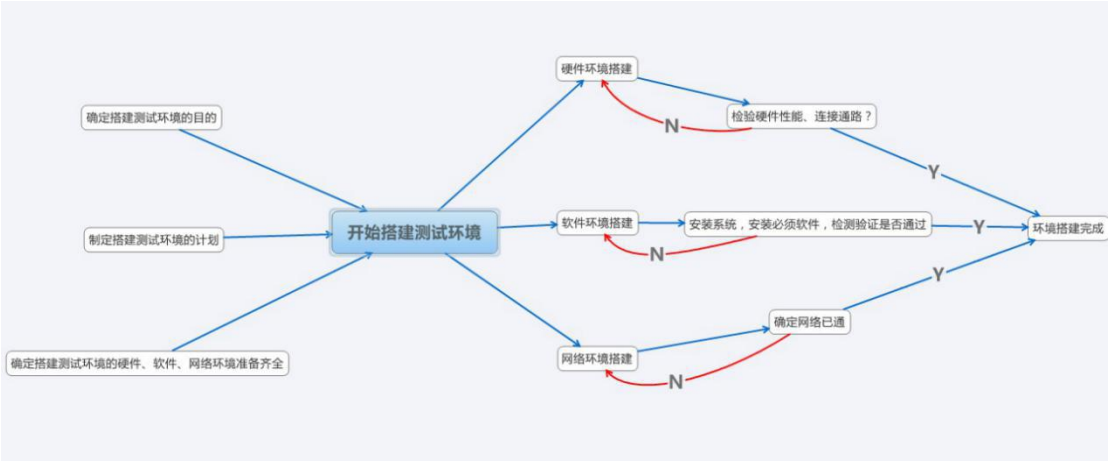
（3）选用比较普及的操作系统和软件平台；

（4）营造纯净、独立的测试环境；

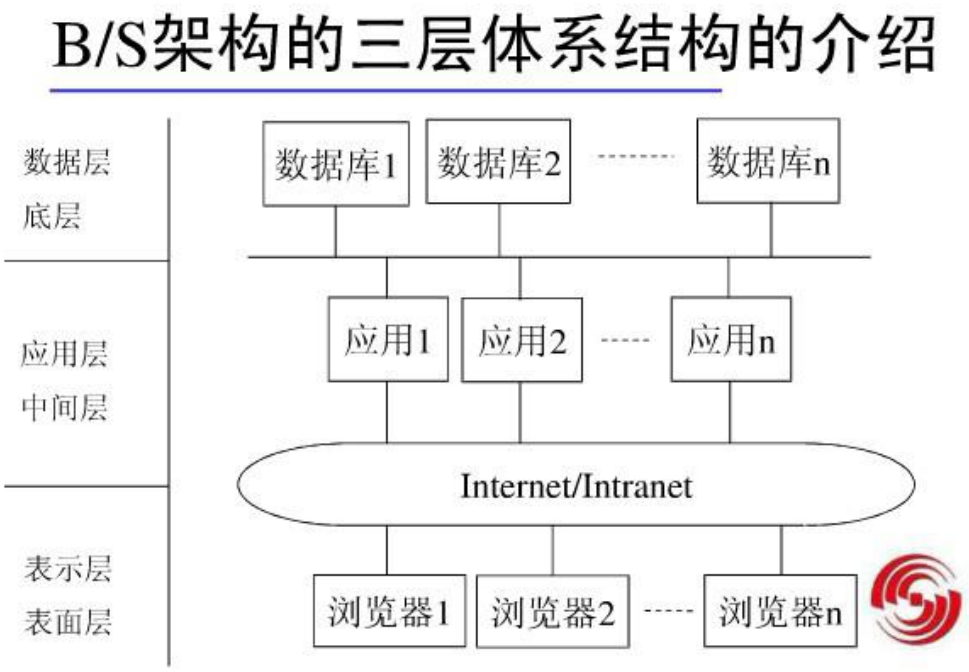
（5）无毒的环境；

测试环境搭建流程

测试环境搭建流程图



B/S 测试环境搭建



BS 测试环境搭建步骤：

1. 数据库服务器端测试环境安装步骤：

（1） 选择服务器
（2） 安装操作系统
（3） 安装数据库
（4） 安装杀毒软件
（5） 杀毒
（6） 制作 Image 文件
（7） 安装软件数据库文件
（8） 进行相关数据库配置
（9） 杀毒
（10） 制作 Image 文件

2. 应用服务器端测试环境安装步骤：

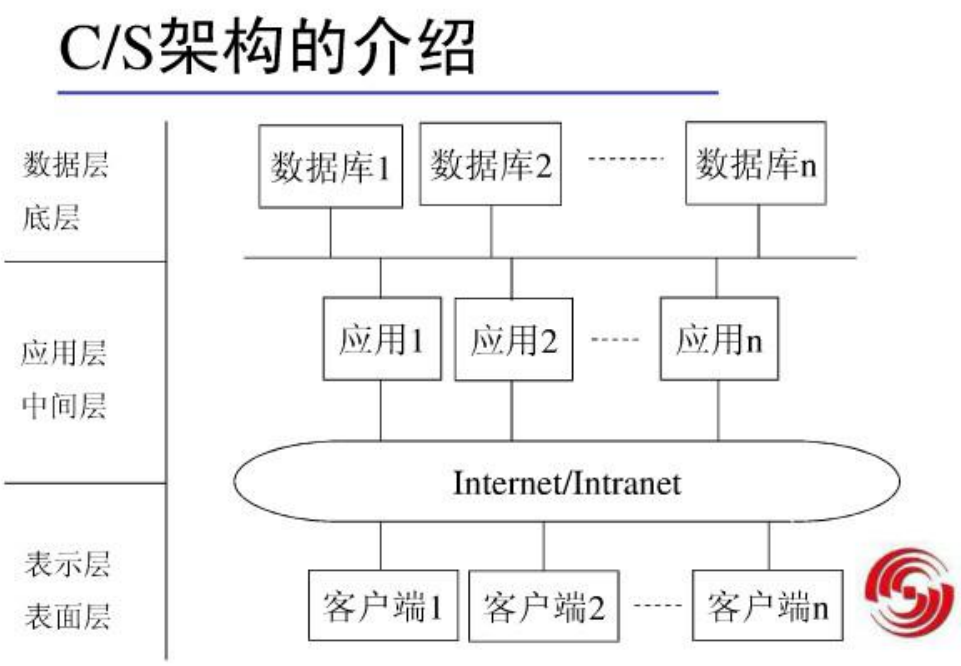
（1） 选择服务器
（2） 安装操作系统
（3） 安装数据库
（4） 安装杀毒软件
（5） 杀毒
（6） 制作 Image 文件
（7） 安装软件数据库文件
（8） 进行相关数据库配置
（9） 杀毒
（10）制作 Image 文件

3. 客户端测试环境安装步骤：

（1） 选择 PC 机
（2） 安装操作系统

（3） 安装杀毒软件
（4） 安装软件要求的浏览器版本
（5） 测试与应用服务器的链接
（6） 杀毒
（7） 制作 Image 文件

C/S 测试环境搭建



1.应用服务器端测试环境安装步骤：
（1） 选择服务器或 PC 机
（2） 安装操作系统
（3） 安装杀毒软件
（4） 安装服务器软件
（5） 杀毒
（6） 制作 Image 文件
（7） 安装服务器端应用软件
（8） 进行相关配置、测试与数据库服务器的链接
（9） 杀毒
（10）制作 Image 文件
2. 客户端测试环境安装步骤：
（1） 选择 PC 机
（2） 安装操作系统
（3） 安装杀毒软件
（4） 杀毒
（5） 制作 Image 文件

（6） 安装客户端应用软件
（7） 进行相关配置、测试与应用服务器的链接
（8） 杀毒
（9） 制作 Image 文件

测试环境的数据生产与维护

测试数据：
（1） 测试数据也是影响测试环境的一个因素
（2） 测试数据直接影响着测试的质量
（3） 测试数据有多种来源：用户、测试人员设计、随机生成
对测试数据的要求：
（1） 功能测试不需要大量的数据
（2） 功能测试需要数据的覆盖率高
（3） 功能测试的测试数据要求尽量真实
（4） 性能测试需要大量的数据
（5） 性能测试的测试数据应尽可能的达到符合实际的数据分配
测试数据的获取：
（1） 用户数据—从最终用户处获取实际数据
（2） 手动获取—对测试数据进行手动添加
（3） 自动获取—利用自动测试工具进行自动的数据添加
（4） 编码—利用数据库接口通过编制程序来获取数据
（5） 随机生成—任意的向软件中输入数据

测试环境备份

目的：
（1） 维持测试环境的一致性；
（2） 恢复测试数据；
（3） 恢复测试环境的当前状态；
备份的内容：
（1） 利用备份工具将常用的操作系统做一个比较完整的 GHOST 文件；
（2） 测试系统环境备份；
（3） 项目定期备份到服务器（将比较重要的测试用例或过程文档保留一份在服务器）

为什么要备份？
测试过程中会遇到多种不可预测的事情，一旦造成系统崩溃，则会造成测试数据丢失、测试过程中断或者测试环境的重新搭建；经常对测试环境进行多次必要的备份是一个必备的预防措施和一个比较好的习惯；对测试环境的备份可以换回不必要的损失、节省测试的时间、保持测试的连续性；

备份的原则：
经常对测试环境进行多次必要的备份是必备的预防措施和好的习惯

备份的好处：

挽回不必要的损失、节省测试的时间、保持测试的连续性

测试环境恢复

一旦测试环境遭到破坏，可以还原最近备份的系统，实现测试环境的恢复

目的：

（1） 维持测试环境的一致性；

（2） 恢复测试数据；

（3） 回复测试华景的当前状态；

测试环境的快速恢复：

（1） 将对测试环境备份获得的 Image 文件复制到其他硬件配置相同的计算机上

（2） 利用备份恢复工具进行恢复

（3） 实现多台机器同样的测试环境的快速搭建，从而节省搭建测试环境的时间

测试环境备份与恢复的工具 **Gohst（Symantes）\Partimage** 等

Gohst 的使用说明：

（1） Ghost 是一个硬盘拷贝软件、它把整个硬盘映像压缩保存为 Image 文件制作；

（2） 通过制作 Image 文件来实现测试环境的备份；

（3） 将原先制作的 Image 文件进行恢复来实现整个硬盘的还原，从而实现测试环境的恢复；

测试用例

测试用例的编写

测试用例是一名测试的基本功。至于测试用例的设计方法，相信大家也都不陌生，在这里笔者就不讲了。如果有不了解的同学，麻烦自己去百度一下。

测试用例是测试工作的核心，它的作用是指导测试完成工作。我看到有一些同学讲，测试的时间少，或者是我们是敏捷开发等等，导致没有编写测试用例。

我也有遇到这种情况，不过我的处理方法是编写测试点，并尽量细化。我实在无法想象在没有 **CheckList** 或者是 **Testcase** 要怎样去开展测试工作。我在刚入行的时候，前几个月，也是一脸懵逼，没有 **CheckList**，没有 **Testcase**，没有任何文档（包括需求文档），就那样浑浑噩噩的进行测试。导致的结果就是，没有目标，毫无目的性，做了很多重复的无用功，也漏测了很多。所以我给大家的建议就是，哪怕时间不够，也要去做个 **list**，不然开展工作真的很难。

下面我给大家分享一下我写的用例和测试点。

CheckList

Testcase

模块		预置条件	编号	操作步骤	预期结果	实际结果	编写人
安装							
安装	安装方式	能够下载安装包	1	使用设备下载安装包，进行安装	正常完成安装		
			2	通过第三方工具，安装到设备上	正常完成安装		
			3	PC端下载，安装包移到设备内存安装	正常完成安装		
	提示语		4	检查安装时的提示语（权限）	无异常提示		
	安装路径	能够选择安装的位置	5	选择SD卡进行安装	能够完成安装		
			6	选择设备自身内存进行安装	能够完成安装		
			7	安装在与安装包不一样的地址	能够正常进行安装		
	安装结束		8	删除安装包	能够删除安装包，且不影响应用正常使用		

上面是本人写的测试点和测试用例。相比测试用例，测试点会简洁很多。主要是因为时间的原因。测试点写起来更省时，也省力一点。我一般在写测试点的时候也会尽量去细化测试点。我现在在时间不够的情况下都是先写 list，然后有时间的时候再补充 case。这是一种习惯，我也希望大家能养成这种习惯。

测试用例的模板有很多，模板的选择完全看个人习惯，每个人习惯不同，只要保证用例的核心内容就可以了。

没有文档怎么办

就像我上面讲到的，有些公司可能没有任何文档，这个时候我们要怎么办？不写吗？我当时遇到这种情况，就是一遍又一遍的去找产品确认需求。是不是很蠢？可是也这样走过来了。至今我没有想到什么好的解决方法。如果哪位同学有好的方法建议可以告诉我。

测试用例，一定要保证覆盖率。我们如何去保证覆盖率呢？

- 异常场景。一定要考虑异常场景，用户是不知道我们的运行环境要求的，所以一切都有可能发生。尽量去模拟用户可能发生的场景。套用一位前辈的话，测试的时候一定要把用户都当成 SB。
- 用例评审。完成编写之后，进行用例的评审。之所以进行这个评审，是因为，我们个人的思维始终是有局限性的，而且由于用例本身就是我们自己编写的，所以我们自己的思维已经被固化了，这个时候就需要通过评审，通过其他人，来找到我们没有考虑到的场景。

测试用例的评审

上面我们讲了测试用例的评审，接着我们来看看，如何评审，哪些人参与评审。

每个公司都有每个公司内部的用例评审标准，不过一般来讲，对用例的评审要求不外乎下面几种：

- 测试用例是否具有指导性
- 测试用例的完整性、准确性、易懂程度、可操作性、可维护性

- 测试用例的执行效率，尽量避免冗余的用例
- 是否覆盖率所有的需求
- 是否完全遵守了需求的规定

一般来讲，用例的评审都是通过会议的方式，与会者在设计人员讲解之后给出意见和建议，同时进行详细的评审记录。

笔者之前的公司做评审的时候参与评审会议的包括：项目组中所有的测试人员，项目负责人，项目组中所有的开发人员。

测试用例的维护

需求的变动导致版本一直在变，而每一次版本变动都会对测试用例集产生影响，所以测试用例集也需要不断的变更和维护，使之与产品变动保持一致。下面介绍几种可能导致测试用例变更的原因：

1）软件需求变更：软件需求变更可能导致软件功能的增加、删除、修改等变化，应遵循需求变更控制管理方法，同样变更的测试用例也需要执行变更管理流程。

2）测试需求的遗漏和误解：由于测试需求分析不到位，可能导致测试需求遗漏或者误解，相应的测试用力也要进行变更。特别是对于软件隐性需求，在测试需求分析阶段容易遗漏，而在测试执行过程中被发现，这时需要补充测试用例。

3）测试用例遗漏：在测试过程中，发现测试用例未覆盖全部需求，需要补充相应的测试用例。

4）软件发布后，用户反馈的缺陷：表明测试不全面，存在尚未发现的缺陷，需要补充或者修改测试用例。

对于提供软件服务的产品，其多个版本常常共存，而对应的测试用例也是共存的，而且测试用例需要专人定期维护，并遵循以下原则：

1）及时删除过时的测试用例

需求变更可能导致原有部分测试用例不再适合新的需求要求。例如，删除了某个功能，那么针对该功能的测试用例也不再需要。所以随着需求的每一次变更，都要删除那些不再使用的测试用例。

2）及时删除冗余的测试用例

在设计测试用例时，可能存在两个或者多个用例测试相同内容，降低回归测试效率，所以要定期整理测试用例集，及时删除冗余的测试用例。

3）增加新的测试用例

由于需求变更、用例遗漏或者版本发布后发现缺陷等原因，原有的测试用例集没有完全覆盖软件需求，需要增加新的测试用例。

4）改进测试用例

随着开发工作进行，测试用例不断增加，某些用例随着系统输入和当前状态的变化而变得不再适用，这些用例难以重用，影响回归测试的效率，需要进行改进，使之可重用可控制。

总之，测试用例的维护是一个长期的过程，也是一个不断改进和完善的过程。

实施测试

终于开始测试了。你觉得怎样进行测试吗？严格的执行测试用例？OH，NO！也太 Low 了点吧。

测试进行时

时刻记着，用例是我们工作的核心，它指导我们开展测试的工作。但是用例是死的，人是活的，它不能用来局限我们的思维。我们要做的不仅仅是执行用例。仅仅只是执行用例，并不能达到我们的预期。

还要进行发散性测试，冒烟测试等等。所谓冒烟测试，也就是随机测试，根据自己的经验，去做一些随机的操作。这样更有利于我们发现一些潜在的问题。

对了，提醒大家，一定要相信自己的直觉。很多时候，我们的直觉会带来意想不到的收获。有几次笔者都是通过直觉发现了一些隐藏的缺陷。这些毫无科学根据，但是却是客观的存在着。

在做完这些测试之后，为了保证覆盖率，减少漏测，我们可以与其他的同学进行交换测试。

进行测试的同时，也要记录测试结果、跟踪缺陷吧啦吧啦的，一些繁琐的事，相信大家都能搞定，笔者也就不啰嗦了。

再多嘴一句，无论巨细，只要是缺陷，一定要记录下来。一定！

关于偶现的缺陷

在测试过程中我们可能遇到这样或者那样的问题。导致我们很头痛。最典型的就是偶现的缺陷。一般遇到偶现的问题，尤其是没有 log 的情况下，尽量去尝试复现吧。有的时候，我甚至为了复现一个 BUG 忙上几天。当然，这是时间充足的情况下。在时间紧迫的时候，一般都会汇报给领导，然后进行风险评估，再进行相应的处理。

关于漏测

说到了漏测，最初的时候，笔者也是兢兢业业的，超级害怕漏测，担心线上出问题，担心自己做的不够好受到责备等等。但是做到现在，已经能够坦然面对了。

漏测是难以避免的。如果真的出现什么特别严重的问题，那责任也是整个团队的，就算怪，也怪不到你的头上，你一个小测试也承担不起这么大的责任。所以不必害怕。大家尽力就好，也没必要去纠结什么。出了问题解决就可以了。与其把时间浪费在纠结是否存在漏测的问题上，倒不如花更多时间，去做一些有意义的事。比如，检查，验证自己的工作。然后去 get 新技能。

2/8 定律

80%的缺陷都集中在 20%的模块中。我们通过一段时间的接触，慢慢就会发现程序员编

码的习惯，也会慢慢就发现缺陷集中的地方，哪些地方容易或者可能出缺陷。下面笔者和大家分享一下这些年遇到的最多的缺陷集中的地方。

- 边界值，几乎是最常遇到的问题之一。这个点要特别注意。可能因为程序的疏忽，一个判断的错误，导致边界值出错。
- 前后端数据的一致性，也是容易频发的缺陷，可能由于，前端的请求未发送，后端的结果未返回，或者是弱网条件下丢包等原因造成的。
- 功能实现错误。包括出现 BUG，或者功能实现的与需求不符。BUG 是由于代码的错误造成的，包括逻辑，算法等等错误。与需求不符，一般就是由于沟通上的问题导致需求理解错误。
- 文件配置错误。有的时候我们可能会需要用到一些配置文件，去做活动或者什么的。这里一般出错的原因是文件的误删，文件配置修改却没有上传。也可能是配置文件没有即时更新。
- 服务端的验证。服务端对客户端请求的验证，现在很多工具，都可以抓包，改参数，做了服务端的验证，可以更加有效的防范这些东西。

做好黑盒不容易

看到这里，如果大家能够完成上面的讲述的东西，那么相信大家已经可以很出色的完成自己的工作了。不过，做好黑盒测试，真的很不容易。

我们不仅仅需要完成的是功能测试，很多时候，我们还要对软件的易用性，稳定性，客户端的性能进行测试评估，要协助开发定位缺陷，要学会去使用工具，要学会去获取 **Log。**

路还很长，我们慢慢走。本章节中将会为大家讲述，软件的易用性测试，稳定性测试，客户端的性能测试，以及如何获取 log，还会介绍笔者常用的一些小工具。

易用性测试

所谓易用性测试，其实就是体验性测试。这里为了显得逼格高一点，所以叫易用性测试了。一般来讲易用性测试，都是 UI 层的测试，主要是用来检查 UI 是否符合需求，是否美观，是否用利于用户的操作，审美等等。

如果是定制的软件，在用户需求明确的情况下，实施易用性测试可能目标就会清晰很多。然而笔者所做过的项目，大部分都是面向广大的用户的，并没有客户定制。所以在做易用性测试的时候，笔者都会去熟悉，体验市场上类似的产品，进行比较和分析。

一般情况下在进行的易用性测试都会从固定的几个点进行测试：

- UI 界面静态测试
  对软件中 UI 静态界面与元素集合设计静态测试用例
- UI 界面操作测试
  对软件中的所有交互的 UI 进行操作
- UI 逻辑流程测试
  对软件中的操作流程的逻辑，交互的逻辑进行测试
- 帮助文档测试
  对软件中的帮助说明（包括用户使用手册），进行测试

由于笔者主要从事的是 APP 的测试，所以这里我们大部分情况都是从 APP 的角度去给大家介绍如何进行上面几种情况的测试。

UI 界面静态测试

根据需求文档与 UI 效果图，确认所有 UI 元素图形,文字显示正常,位置正确,内容正确。UI 元素包括下拉式菜单、工具条、滚动条、对话框、按钮、图标和其他控制。

- 锁屏、横竖屏切换或者是后台运行程序和恢复后对界面有没有造成变形，显示失常等影响。
- 使用不同分辨率的设备对 UI 界面有没有造成变形，显示失常等影响。

UI 界面操作测试

- 菜单，按钮等常规控件的操作
- 虚拟键盘，触摸屏等其他操作方式
- 手写

UI 逻辑流程测试

- 状态跳转测试
- 输入输出测试
- 错误提示测试
- 安装跳转测试

帮助文档测试

- 帮助文档内容的正确性
- 帮助文档内容的一致性
- 帮助文档的搜索功能

健壮性测试

健壮性测试也被称为容错性测试，主要用来检测，当系统出现异常时的处理机制。

一般情况下做这个测试，采用异常场景居多，一般采取中断测试和弱网测试为主。

中断测试的异常场景

弱网测试的测试点

正如我们上面所讲到的，健壮性测试主要是用来检测系统在面对异常时的处理机制，所以我们要尽可能的去考虑异常场景，并且要伴随着冒烟测试去做。

稳定性测试

在这里我们所说的稳定性，仅仅只是针对客户端，笔者目前还没有做到服务端的测试哈。

OK，我来给大家介绍一下我所了解的稳定性测试使用的一些工具和方法。

monkey

做 APP 的同学对这个应该不会陌生吧。曾经我也是觉得这个东西好厉害，很高大上，错误的以为这个东西是做性能测试的。很多人都知道这个是做压力测试的。没错，它就是做压力测试的，不过它也仅仅只是针对 UI 层的。

我这样解释一下，**monkey** 就是模拟用户的随机操作，但是也正是因为它是随机事件流，我很少使用，也并不喜欢它。这不是对它的否认，只是个人原因罢了。

monkey 的安装

monkey 程序由 **Android** 系统自带，使用 **Java** 语言写成，所以在使用 **monkey** 之前我们一定要配置好 **Java** 和 **Android** 的环境变量。

环境配置完成之后，就可以使用 monkey 了，那么我们先来检查一下 monkey 能否使用。

在 **cmd.exe** 中运行一下 **adb shell monkey -help**

```
F:\sdk\sdk\platform-tools>adb shell monkey -help
usage: monkey [-p ALLOWED_PACKAGE [-p ALLOWED_PACKAGE] ...]
              [-c MAIN_CATEGORY [-c MAIN_CATEGORY] ...]
              [--ignore-crashes] [--ignore-timeouts]
              [--ignore-security-exceptions]
              [--monitor-native-crashes] [--ignore-native-crashes]
              [--kill-process-after-error] [--hprof]
              [--pct-touch PERCENT] [--pct-motion PERCENT]
              [--pct-trackball PERCENT] [--pct-syskeys PERCENT]
              [--pct-nav PERCENT] [--pct-majornav PERCENT]
              [--pct-appswitch PERCENT] [--pct-flip PERCENT]
              [--pct-anyevent PERCENT] [--pct-pinchzoom PERCENT]
              [--pkg-blacklist-file PACKAGE_BLACKLIST_FILE]
              [--pkg-whitelist-file PACKAGE_WHITELIST_FILE]
              [--wait-dbg] [--dbg-no-events]
              [--setup scriptfile] [-f scriptfile [-f scriptfile] ...]
              [--port port]
              [-s SEED] [-v [-v] ...]
              [--throttle MILLISEC] [--randomize-throttle]
              [--profile-wait MILLISEC]
              [--device-sleep-time MILLISEC]
              [--randomize-script]
              [--script-log]
```

如果出现了这个就说明，我们已经可以开始使用 monkey 了。

一般的 **monkey** 无法使用，都是因为 **Java** 或者是 **Android** 的环境没有配置好，自己检查一下把环境配好就没问题了。所以在配环境的时候一定要留心。

monkey 的基本操作指令

monkey 一般都是通过 **cmd.exe** 运行：**adb shell monkey**（+命令参数）来进行 **monkey** 测试了。

1）参数： -p

参数**-p** 用于约束限制，用此参数指定一个或多个包（**Package**，即 **App**）。指定

包之后，**monkey** 将只允许系统启动指定的 **APP**。如果不指定包，**monkey** 将允许系统启动设备中的所有 **APP**。

* 指定一个包： **adb shell monkey -p com.example.sellclientapp 100**

说明：**com.example.sellClientAPP** 为包名，100 是事件计数（即让 **monkey** 程序模拟 100 次随机用户事件）。

* 指定多个包：**adb shell monkey -p com.htc.Weather –p com.htc.pdfreader -p com.htc.photo.widgets 100** 如图所示：

```
F:\sdk\sdk\platform-tools>adb shell monkey -p com.example.sellclientapp 10
Events injected: 10
## Network stats: elapsed time=38ms (0ms mobile, 38ms wifi, 0ms not connected)
```

* 不指定包：**adb shell monkey 100**

说明：**monkey** 随机启动 **APP** 并发送 100 个随机事件。

```
F:\sdk\sdk\platform-tools>adb shell monkey -p com.example.sellclientapp 1000
// CRASH: com.example.sellclientapp (pid 3323)
// Short Msg: java.lang.RuntimeException
// Long Msg: java.lang.RuntimeException
// Build Label: samsung/kyleizn/kyleichn:4.0.4/IMM76I/S7562iZNAMB1:user/release
keys
// Build Changelist: S7562iZNAMB1
// Build Time: 1361782029000
// java.lang.RuntimeException
//       at com.example.sellclientgui.view.dialog.MessageModelDialog$1.handleMes
age(MessageModelDialog.java:135)
//       at android.os.Handler.dispatchMessage(Handler.java:99)
//       at android.os.Looper.loop(Looper.java:137)
//       at android.app.ActivityThread.main(ActivityThread.java:4517)
//       at java.lang.reflect.Method.invokeNative(Native Method)
//       at java.lang.reflect.Method.invoke(Method.java:511)
//       at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteIni
.java:993)
//       at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:760)
//       at dalvik.system.NativeStart.main(Native Method)
```

* 要查看设备中所有的包，在 CMD 窗口中执行以下命令：

>**adb shell**

#**cd data/data**

#**ls**

我的手机没有 root 所以不能用这个属性了。

2）参数： -v

用于指定反馈信息级别（信息级别就是日志的详细程度），总共分 3 个级别，分别对应的参数如下表所示：

日志级别 **Level 0**

示例 **adb shell monkey -p com.htc.Weather –v 100**

说明缺省值，仅提供启动提示、测试完成和最终结果等少量信息相应源代码如图所示了，这十分有利于调试了。

```
管理员: C:\Windows\system32\cmd.exe
F:\sdk\sdk\platform-tools>
F:\sdk\sdk\platform-tools>adb shell monkey -p com.example.sellclientapp -v 1000
:Monkey: seed=0 count=1000
:AllowPackage: com.example.sellclientapp
:IncludeCategory: android.intent.category.LAUNCHER
:IncludeCategory: android.intent.category.MONKEY
// Event percentages:
//   0: 15.0%
//   1: 10.0%
//   2: 2.0%
//   3: 15.0%
//   4: -0.0%
//   5: 25.0%
//   6: 15.0%
//   7: 2.0%
//   8: 2.0%
//   9: 1.0%
//   10: 13.0%
:Switch: #Intent;action=android.intent.action.MAIN;category=android.intent.categ
ory.LAUNCHER;launchFlags=0x10200000;component=com.example.sellclientapp/.main.Ma
inActivity;end
    // Allowing start of Intent { act=android.intent.action.MAIN cat=[android.in
tent.category.LAUNCHER] cmp=com.example.sellclientapp/.main.MainActivity } in pa
ckage com.example.sellclientapp
:Sending Flip keyboardOpen=false
:Sending Touch (ACTION_DOWN): 0:(73.0,687.0)
:Sending Touch (ACTION_UP): 0:(73.19875,687.4951)
:Sending Touch (ACTION_DOWN): 0:(187.0,84.0)
:Sending Touch (ACTION_UP): 0:(192.7109,82.90212)
:Sending Trackball (ACTION_MOVE): 0:(-1.0,-1.0)
:Sending Trackball (ACTION_MOVE): 0:(1.0,-3.0)
:Sending Trackball (ACTION_MOVE): 0:(-1.0,-2.0)
:Sending Trackball (ACTION_UP): 0:(0.0,0.0)
:Sending Touch (ACTION_DOWN): 0:(497.0,1263.0)
```

日志级别 **Level 1**

示例 **adb shell monkey -p com.htc.Weather –v -v 100**

说明　提供较为详细的日志，包括每个发送到 **Activity** 的事件信息

日志级别 **Level 2**

示例 **adb shell monkey -p com.htc.Weather –v -v –v 100**

说明　最详细的日志，包括了测试中选中/未选中的 **Activity** 信息

3)参数：-s

于指定伪随机数生成器的 seed 值，如果 seed 相同，则两次 monkey 测试所产生的事件序列也相同的。

* 示例：

monkey 测试 1：adb shell monkey -p com.htc.Weather –s 10 100

monkey 测试 2：adb shell monkey -p com.htc.Weather –s 10 100

两次测试的效果是相同的，因为模拟的用户操作序列（每次操作按照一定的先后顺序所组成的一系列操作，即一个序列）是一样的。操作序列虽然是随机生成的，但是只要我们指定了相同的 Seed 值，就可以保证两次测试产生的随机操作序列是完全相同的，所以这个操作序列伪随机的；

```
F:\sdk\sdk\platform-tools>adb shell monkey -p com.example.sellclientapp -s 10 10
0
Events injected: 100
## Network stats: elapsed time=205ms (0ms mobile, 205ms wifi, 0ms not connected)
```

由于本人并不使用 monkey，所以只能从网上找了些命令的参数给大家来解释。笔者也大致的看了下，并没有多难，一般的常用命令就那么几个，不过就是用起来的话要 root 手机。命令的话，大家不用记住，百度上有很多，用的时候直接百度就好

UIAutoMonkey

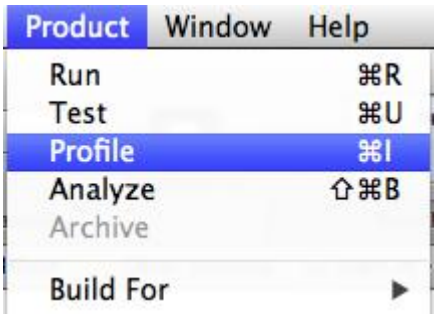
上面为大家讲述的是安卓的稳定性的测试工具 **monkey**。在工作过程中，也会有不少的同学问我，**iOS** 的稳定性怎么测试？最初的我也是一脸懵逼，因为我很少做稳定性测试这一块儿。后来也是在前辈的教导下，知道了 **UI Automation** 这个东西。这个工具在 **iOS** 上就相当于是安卓的 **monkey**。

UI Automation 的安装

首先我们要有一台 **Mac** 电脑这个工具离开 **Mac** 电脑是无法使用的。然后，你需要安装 **Xcode**。

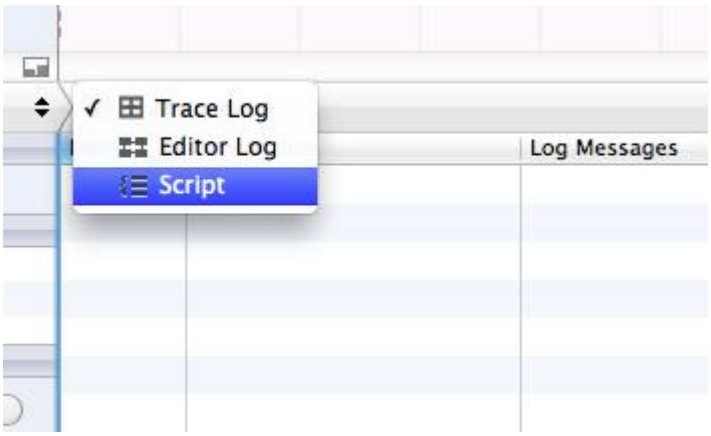
Xcode 中自带了 **UI Automation** 和 **Instrument**，然后我们按照下面的步骤进行操作，就可以使用 **UIAutomonkey.js** 这个 **js** 脚本，进行 **monkey** 测试了。

1）首先，使用 **xcode** 打开你的 **ios** 项目，从“Product”菜单中选择“**Profile**”（或者直接快捷键 **Command+i**),这样就可以构建 **ios** 项目，并启动工具模板选择器。

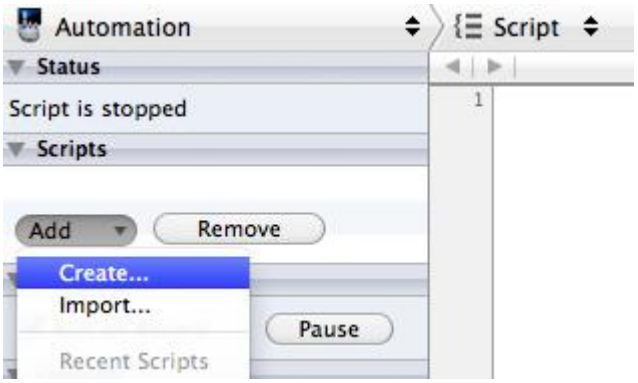


2）下一步，在选择器中选择“**UI Automation**”。当我们创建了自动化模板后，就可以测试 **APP** 的稳定性了。

3）在 **Scripts** 面板中，点击“**Editor Log**”，在下拉列表中选择“Script”，点击“**Add**”按钮，在下拉列表中选择”**Create**”,就可以新建一个 JS 脚本。



4）将 **UIAutoMonkey.js** 这个文件的内容粘贴到新建的 **js** 脚本中（或者可以直接将 **UIAutoMonkey.jsimport** 进去。



5）此时，我们可以直接点击播放按钮，来执行这段脚本，**monkey** 测试就开始了。

额外配置

UIAutoMonkey.js 脚本，开头是下面的代码：

```
    config: {
        numberOfEvents: 1000,
        delayBetweenEvents: 0.05,    // In seconds

        //各事件的几率.
        // 数字越大，几率越大.
        eventWeights: {
            tap: 30,
            drag: 1,
            flick: 1,
            orientation: 1,
            clickVolumeUp: 1,
            clickVolumeDown: 1,
            lock: 1,
            pinchClose: 10,
            pinchOpen: 10,
            shake: 1
        },

        // Probability that touch events will have these different properties
        touchProbability: {
            multipleTaps: 0.05,
            multipleTouches: 0.05,
            longPress: 0.05
        }
        },
```

numberOfEvent 的意思很明确，代表需要产生随机事件的个数。
delayBetweenEvents 代表两个事件之间的延迟时间。这个值一般是需要调整的。如果该值为 0，那么脚本会尽可能快的向设备发送事件。
eventWeights 这个值代表每个事件的触发几率。如果 tab 事件的值为 100、orientation 事件的值为 1，那么 tab 事件触发的几率就是 orientation 的 100 倍。
touchProbability 控制着不同种类的 tab 事件。默认情况下，tab 就是单击事件。调整这些参数可以设置双击、长按事件发生的频率。这些值要界于 0、1 之间。

稳定性测试的到这里也算是告一段落了，那么接下来，我们来看看客户端的性能测试。

客户端性能测试

使用工具

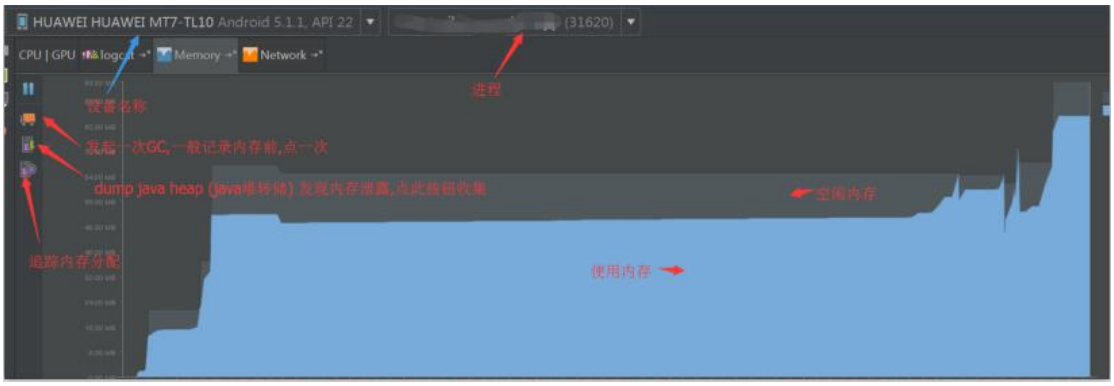
所谓的客户端性能，其实就是监测产品在真机上运行时的各项数据。那么我们要检测哪些数据呢？又要如何去监测呢？
我们要检测的数据如下：
- 内存占用
- CPU 占用
- 电量消耗
- 流量消耗
- 帧数

说到工具，现在工具有很多，大部分还是会使用，**emmagee** 和 **GT**，还有 **Itest** 等一些工具去监测产品在真机上运行时的各项数据。当然，还有一些是程序嵌入的可视化插件去监测。虽然这些监测的工具，所得到的数据并不是那么精确，但是我们还是能够通过多次对比，使误差最小化，从而得出结论。

不过目前笔者还是习惯使用开发工具去监测这些数据，感觉会比第三方的工具稍微精确一点。安卓的话就用 **Android Studio**，iOS 就用 **Xcode**。通常监测时长为 2 小时左右。

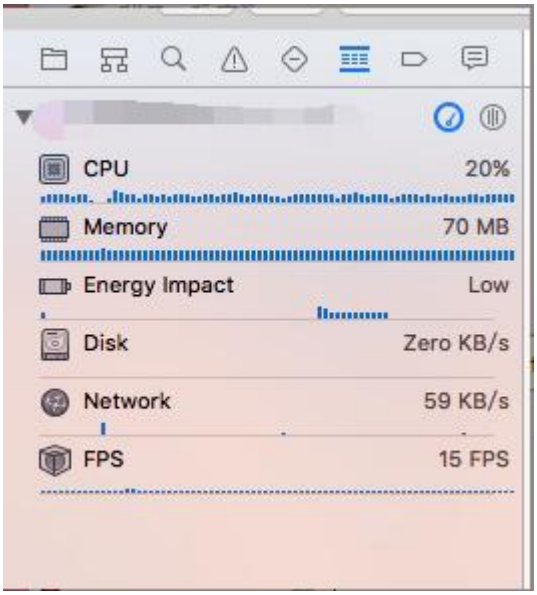
我们来看一下 **Android Studio** 和 **Xcode** 的监测的数据截图

Android Studio



这里并不是 CPU、GPU、内存、网络一起显示的，而是分开显示在不同的栏，需要自己去点击进行选择查看的。它所有的东西都是呈现为这种走势图的看起来很清晰。

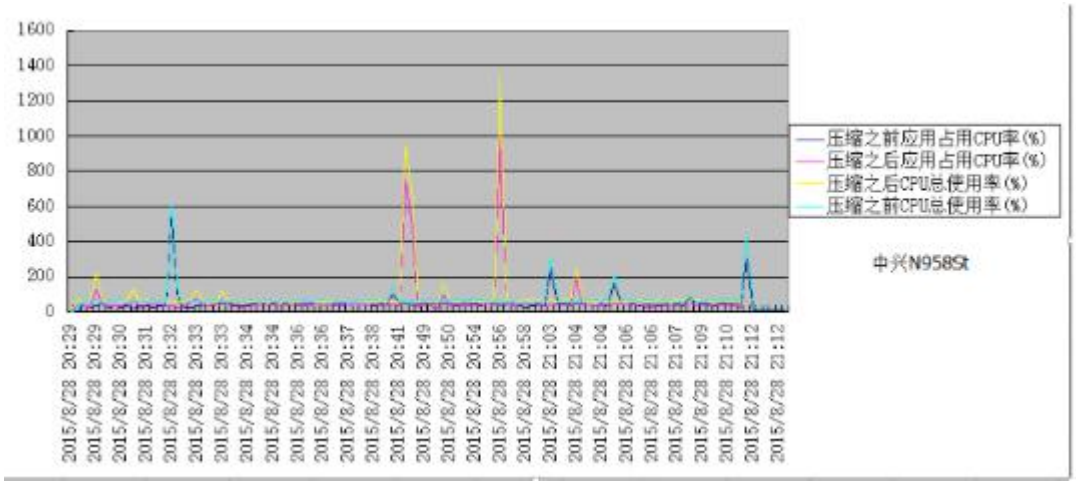
Xcode



相比较而言，这里就是集中在一起显示所有的数据了，不过如果看那种走势图的话，也是分开的哦。

结果分析

测试的过程中，可能会遇到一些数据的异常，那么我们就需要去分析出现这些异常的原因了。我曾经遇到过，CPU 的占用，一下子飙升甚至超过了 100%。如下图：



那么为什么会出现这种情况呢？我当时请教了大企鹅的测试朋友。他给出的结论是，出现这种情况只有两种原因。其中一个我记不太清楚了，但是我当时的分析结果是因为 FPS 的卡顿造成的。由于 FPS 的卡顿，导致进程短时间内出现卡死的现象，CPU 占用会瞬间飙升。

看完了问题，我们再来看如何去分析这些数据。

通过数据，与当时相应的操作，对应的模块分析客户端上的瓶颈。并且要进行纵横的对比，然后来提出优化的建议。所谓的纵横对比就是横向对比市场上其他产品，纵向对比本产品之前的不同版本。

好的，现在我们来说说优化的问题。我所接触到，了解的优化的方法目前只有两种，在这里分享给大家：

资源压缩：图片，配置文件等，进行压缩，尽量删除一些不必要的文件。减少安装包包体大小。

分包：由于整包的包体太大，采用分包的方法，使用动态加载的方式，让玩家在初次下载的时候，不会因为看到包体内存望而却步。当然也有缺点，个人感觉缺点就是，动态加载的时候，会有点卡卡的。

学会使用工具

截止到目前为止，黑盒测试中，我所做过的东西都已经讲的差不多了。

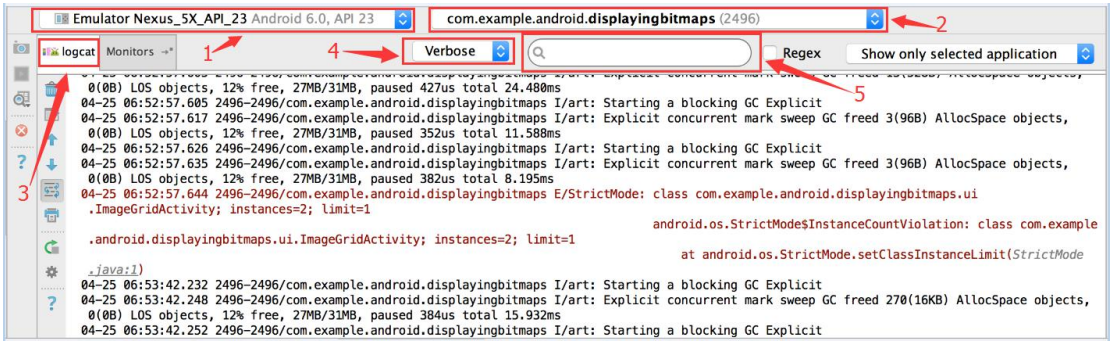
但是在测试的过程中我们总会会遇到这样那样的问题，一定要学会使用工具去帮助我们工作。跟大家讲讲笔者经常用到的一些工具吧。

在测试的过程中，我们要做的不仅仅是测试，还需要协助开发去定位缺陷，这个时候有 Log 就会方便很多。下面给大家分享一下笔者最常用的两种查看 Log 的工具。

Android Studio

这个工具是我现在用来查看安卓 APP 的运行的时候的日志用的。配置好 Java 和 Android 的环境之后，直接下载安装就可以使用了。之前我一直用的是 **Eclipse**，在用过这个之后，深深的被这个工具吸引了。它不仅比 **Eclipse** 更好用，而且还能监测客户端的性能。

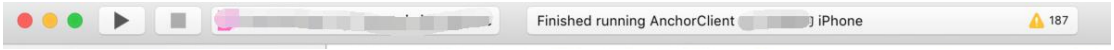
给大家看一下 **Android Studio** 的截图：



1) 使用设备连接电脑之后，打开 **Android Studio** 会在上图的 1 的位置显示你的设备信息，多台设备同时连接的时候，可以通过 1 的下拉列表选择设备进行 Log 的查看。
2) 连接成功后运行需要测试的 APP，这是 APP 的信息会显示在图中 2 的位置，也可以通过点击 2 右方的按钮，在下拉列表中选择需要查看 Log 的应用程序。
3) 一定要点击 Logcat，这才会显示出程序运行时的 Log。Logcat 的位置在图中 3 的位置。
4) 图中 4 的位置是对日志级别的筛选，日志共分为 6 个级别：
   ① Verbose ： 显示所有日志消息。
   ② Debug ： 显示在开发过程中有用的日志消息。
   ③ Info ： 显示一些普通的信息。
   ④ Warn ： 显示警告信息。
   ⑤ Error ： 显示错误日志消息。
   ⑥ Assert ： 显示开发者期望不会发生的事。
5) 5 是个输入栏，主要作用是搜索，通过它可以搜索到你想要的东西。

Xcode

iOS 的 APP 我一般都是使用 Xcode 来查看日志。这有一定的局限性，首先我们必须要有一台 Mac 电脑，然后要安装 Xcode，其次还需要有测试对象的源码。然后使用设备连接电脑之后，打开 Xcode，选择项目，选择设备然后运行就可以在控制台看到我们想要看的日志了。不过这里的日志也包含了设备自身系统的日志。



这里，最右边那一栏是选择设备用的
左边的那个播放按钮点击之后就会运行项目
那个类似暂停播放的按钮就是结束运行
一旦项目开始运行之后，就会在控制台输出设备上的 Log 信息

其他辅助工具

除了查看 LOG 使用的工具，还会用到很多其他工具，在这里依次跟大家说道说道：

- **SVN** 主要作用就是版本控制和管理
- 禅道 集项目管理与缺陷管理，测试用例集管理为一体。
- **Network-Emulator** 弱网测试的工具，能够设置丢包率（iOS 设备可以在开发者选项中进入 **network** 设置丢包率）
- **Fiddler** 抓包工具。这个抓包工具抓的是 http 和 https 的包，不分 C/S 和 B/S 架构，设置好代理就可以了。具体的使用方法麻烦自行百度，这里就不再做介绍了。
- **Xmind** 思维导图 主要用来编写 CheckList
- **Visio** 用来画流程图是非常漂亮的
- **Xshell** 用来连接服务器，进行服务器的相关操作
- **Navicat** 用来连接数据库，进行数据库的相关操作
- **VirtualBox** 用来安装虚拟机

基本常用的工具也就是上面讲述的这些，作用也都有说明，至于有感兴趣的同学，麻烦还是自己百度一下，教程很多，这里就不再做详细的介绍了。

测试进阶-管理 or 技术？

管理 or 技术？

到现在做了差不多两年多的黑盒测试了。此时，已经感觉到在黑盒测试的路上，无法再快速的取得进步。不敢说精通黑盒测试，但是已经能够相当熟练的完成各项任务。

所以，我也在考虑以后得路。我经常在想，怎么在保证产品质量的情况下，提升工作效率，如何才能更高效的去工作？我想到了两种方法，一种就是管理路线，从流程上进行改变，提升工作效率。另一种就是技术路线，通过提升自己的技术，寻找高效工作的方法。

最初的我是想走管理路线的，因为我确实不喜欢搞技术，觉得烦闷无趣。但是一直没有机会去尝试接触到管理的岗位，最好的时候也就带带四五个人的小团队。而且也有前辈曾经告诫我，就算做管理也还是要会技术。所以在经过一番考虑之后，我还是选择了技术路线。

之所以我会选择技术路线，是因为纯管理的，我真的见到的不多，截止到目前为止，也就遇见过两个。而且由于不懂技术的原因，在项目中沟通起来还是存在一些困难的，因为有些东西他是无法理解的。当然这也是个例，我相信纯管理的大腿都是有真材实料的。

技术的发展方向

选择了技术，那么又要选择技术的发展方向，现在软件测试行业里，发展方向众多，自动化，性能，安全，测试开发，白盒测试，灰盒测试。这就要根据个人的兴趣来选择了。

我本人选择的是接口测试，打算从接口测试做起，然后是性能测试和接口自动化。为什么我会这样选择呢？

先来谈谈自动化，自动化现在是待遇相当好的，但是我并没有选择。首先自动化本身的局限性太大了，必须要流程相对的固定的，可重复性高，自动化的测试两个重点要素就是可

重复性和可维护性，而且大部分情况是用来做回归测试和验收测试。而且成本是相当高的，一般的公司根本做不了。我见过太多的自动化测试进入公司之后却从事着黑盒手工测试的工作，所以我没有选择自动化。我觉得如果我会自动化，进入公司之后从事着黑盒手工测试的工作，这是对我这个职业的亵渎。可能话讲的严重了一些，但是这就是我的想法，我希望能够在项目中尽力。

然后是安全测试，接口测试的另一个发展方向就是安全测试。安全测试的待遇也是相当不错的，但是难度太大了，笔者是零基础。结合实际情况进行考虑，笔者还是放弃了安全测试这个方向。其实也有一部分原因是对安全测试不感兴趣吧。脑子里对这个完全没有概念。

至于测试开发、白盒测试、灰盒测试。那是什么鬼，我才不要做好吗？我最讨厌的就是代码，居然让我去接触代码。尽管我现在在坚持学习 Python，但是真的是想吐的好嘛，如果不是为了当初答应人家说 Python 我才不要坚持。

说了这么多，无论选择哪个方向，一定要感兴趣，要坚持下去。IT 行业技术更新很快，不进步，就灭亡。

好了，下面将会为大家讲解笔者通过这段时间的学习眼中的接口测试。

神秘的接口测试

我们先来看一下接口测试的定义：

接口测试是测试系统组件间接口的一种测试。接口测试主要用于检测外部系统与系统之间以及内部各个子系统之间的交互点。测试的重点是要检查数据的交换，传递和控制管理过程，以及系统间的相互逻辑依赖关系等。

接口，一个很熟悉的名词，经常听人提起，却又是那么神秘。也有很多的同学问我如何去做。大家都觉得接口测试很神秘，其实它并不神秘，我们大家在 UI 层的测试就做过，只是不知道而已。现在无非是提升到另一个层级罢了。

抛开 UI 层，客户端我们来看。所谓的接口充满着整个项目，模块与模块之间的交互、调用，端与端的交互、数据的传递，还有外部接口的调用，最常见的就是第三方集成的 SDK 的调用（比如支付、登录、注册）。这些都要通过一个一个的接口去实现。通过接口，将模块，端与端衔接起来，这样才能使我们的项目正常的进行工作。

最简单的接口测试应该就是，拿一个接口，粘贴到浏览器的地址栏然后拼接参数，检查返回数据的正确性，这里这个正确性不仅仅包括数据是否正确还要检查数据格式。当然，我们也能通过工具去完成这项测试。

那么再进一步应该就是通过设置线程、循环、启动时常，去完成压力、并发的测试。这里也可以算作是性能接口测试。这些的通过工具可以实现。用代码实现起来的话，监测的数据没有工具那么全面，但是代码比工具更加灵活。

高级接口测试就涉及到了签名加密的接口，这一类的接口一般都是通过写脚本去进行测试，目前笔者还不了解有什么工具可以完成这一类的接口的测试。

在这里有必要提醒一句，想做接口测试的话，最好是了解网络协议。如果有同学不了解，还请自己去先行了解一下网络协议，这样能够在你接下来的阅读以及再开展接口测试的时候不至于有太多的困惑。

测试金字塔

在学习接口测试期间，曾经看到一个测试金字塔，我觉得还是很不错的，在这里展示给大家看一下：

首先我们来看看这个分层
手动测试：就是我现在这样手动的黑盒测试
界面测试：一般都是 UI 自动化测试，我所了解到的使用的工具就是 **selenium**，通过对 UI 的自动化测试，也能进行功能上的验收。
服务测试：就是对产品所提供的服务进行测试
组件测试：接口测试，可以通过代码实现，也可以借助工具，当前使用最多的工具应该就是 **poster**、**postman** 和 **Jmeter**
单元测试：针对类库和程序集进行测试

从上面这个金字塔我们基本可以看出以下几个特点：

1) 测试越往下面测试效率越高，测试质量保障程度越高

单元测试时对软件中最小可测试单元进行测试，所以在缺陷的定位上更精确。而不同的语言都有自己的单元测试框架，从而也能够保证单元测试的规范。和其他测试相比，单元测试对测试员的要求更高，但是也能够更好的保证测试的质量。

2) 测试越往下面测试成本越低

大家也都知道，Bug 越早发现，损失就越小，成本也越低。这个金字塔，也是符合这个规则的。虽然目前敏捷开发盛行，但是真正的敏捷团队，我至今没有遇到多少，大部分的团队仍然是瀑布式模型。那么按照瀑布模型来看，单元测试，是最早开始实施的。等到界面测试、手工测试的时候才发现的 Bug，再进行定位、解决，花费的时间会更多。

3) 测试越往下面，职业前景越好

越往金字塔底层，测试的技术含量要求更高，测试人员的核心竞争力更大，薪酬当然要高一些，如果从技术方向来说，可以做高级测试工程师、测试架构师都有可能。

接口的分类

外部系统的接口

其实无论是 C/S 还是 B/S 的产品，都会存在外部接口的调用。最经常见到的就是，接入的第三方登录（QQ、微信、微博等），在产品上，使用你们在接入的平台已经注册过得账号进行登录，然后产品就会调用接入的接口，交给接入的第三方进行验证，验证通过的话，返回给产品一个登录成功的信息，就实现了第三方接入的登录。

服务与服务的接口

服务间的接口，在我的眼里看来就是不同模块的互相调用，实现功能上的交互。

举个例子来说，比如我们注册一个账号，这个时候，就会先去调用数据查询的服务，检查数据是否有重复，如果无重复数据再允许注册，然后还要调用新增数据的接口，这样才可以完成一个账号的创建。

上层服务对下层服务的调用

应用层：系统的功能
服务层：服务器对数据和逻辑的处理
数据库：主要用来存放数据
就像上面的例子，我们去注册一个用户。

首先在应用层，我们通过输入框和注册的按钮，输入我们将要注册的用户的信息，然后通过注册按钮，提交请求至服务层。

这个时候服务层，在接收到应用层的请求之后，要先去判断是否是重复的数据，所以就要去调用数据库的查询接口，如果发现是重复数据，那么返回一个是重复数据的状态码到服务层，服务层，根据状态码做出判断，返回到应用层，应用层根据服务层返回的内容，做出相应的处理。如果不是重复数据，那么也会返回一个状态码给服务层，服务层判断之后作出处理，返回到应用层。然后我们在应用层上看到的就是注册成功了。

上面讲述的这个例子，在各个层级中的数据传递，层级之间的交互都是通过接口的。

我目前所接触的就是服务与服务的调用与应用层和服务层的调用稍微多一点，目前接触到的接口都还是 http 的协议。所以可能这里写的不是很好，不过也绝对不会去误人子弟。

接口测试的意义

到了这里，相信大家对接口和接口测试已经有了一定程度上的了解。在实际工作中，可能听到的更多的是自动化，性能，安全等，接口测试的相关内容感觉并不是多么常见。那么接口测试的意义又是什么？

保证系统的稳定性

做了这么久的测试了，相信大家也都知道，越接近底层对系统的影响就越大。服务端的一个缺陷，可能会引起客户端的几个甚至十几个缺陷，还有一些会直接导致客户端的崩溃。这对整个系统造成的损失是不可估量的，因此服务端接口的质量将直接影响到系统的正确和稳定。

正常来讲，一般的开发或者是白盒测试，都需要进行单元测试，而相对于单元测试来讲，接口测试需要测试的接口或者函数的数量会远远小于单元测试，所以，接口定义的稳定性会远远高于类级别的函数。

将缺陷控制在项目前期

对于我们测试来说，越早介入测试越好。由测试金字塔中的介绍可以看出，他的粒度比单元测试更粗。一般是基于子系统或者是子模块的接口层面的测试，所以并不需要等到完全集成，所有功能开发完成才能进行测试。从这个角度来看，它确实能够帮助我们将缺陷控制在项目前期。

节省测试成本

我们上面也说到了，接口测试是基于子系统或者是子模块的接口层面的测试。因此，接口测试需要测试的接口或者函数的数量会远远小于单元测试，所以，接口测试用例代码的改动量也远远小于单元测试，代码维护成本会比单元测试少很多，因而测试的投入量会小很多。而且将缺陷有效的控制在了项目前期，避免了很多不必要的麻烦，所以我认为，接口测试能够节省测试成本。

接口测试用例设计

笔者在最初接触到接口测试用例设计的时候，根本不知道应该怎么做，不过后来通过一段时间的学习加上自己的实践。也是明白了接口测试用例设计的一些小技巧。其实接口测试用例的设计和我们做功能的用例的设计并没有什么不同。只不过黑盒的时候，我们的用例是操作步骤、预期结果描述的文字信息，而在接口测试里面操作步骤变成了数据。同样的是都要保证覆盖率。

如何简单设计接口测试的设计用例

a) 明确出发点——测试的目的是为了让找出软件的缺口，修复并使之更加完善。在这一基础点上，接口测试也不例外。以找出软件的误漏为出发点，测试用例需紧贴此线，更容易找出问题所在。

b) 明确测试点——选择好的测试对象。系统内部层次繁复复杂，任何一个接口的变动都将导致用例失效。（可将这些最外层的接口根据数据的流向分为进入和流出两类，进入系

统的接口实际上是我们用例的执行调用的接口。可通过参数对这些接口进行调用，模拟外部的使用；而流出的接口则是我们用例真正该验证的点。数据从哪里流出，流出的状态如何，此时系统的状态都是作为测试目的所要着重关注的部分）

c) 确认完整的测试对象的功能——确认外部接口提供给使用这些接口的外部用户什么样的功能，外部用户真正需要的时什么样的功能予以区别。用例的设计要严格按照测试对象功能设计才是正确的用例。

接口测试用例有哪些要求：结构好，可读性高，渗透性强。

接口测试用例包括的内容

a) **接口测试测试的功能点**：如果一个接口功能过于复杂时，可以对接口用例进行结构划分（如根据层次，平台，功能点等等），这样用例具有更好的可读性（接口划分原则为：以接口提供的功能点的不同进行合适粒度的划分，同一功能点的用例又可根据测试环境的不同，数据的不同进行用例的填充）

b) **接口测试用例的环境**：程序内部环境和程序所调用的外部接口的环境。

c) **关于接口测试测试数据**：分为两部分：接口参数数据和用例执行所需系统数据。数据的设计、准备测试用例的数据不可马虎。通过好的测试数据查找问题，能极大的提高工作效率。接口参数数据需要对每个参数根据测试接口的实际功能进行分析，在符合业务逻辑的情况下进行逻辑组合排列，一定要记得边界值和错误点的数据，这样用例更容易发现问题。

d) **执行操作**：即对所测接口的调用。

e) **预期结果**：根据需求进行验证，是衡量软件是否达到预期的标准。应该着重细致，每个用例均需验证，应该避免一个用例重复做相同的验证，提高测试用例的效率。

具体测试用例的参考点

a) **参数测试**：针对参数进行的测试，也可以说是假定接口参数的不正确性进行的测试，确保接口对任意类型的输入都做了相应的处理：输入参数合法（不合法），输入参数为空，为null，输入参数超长等等；

b) **功能测试**：“接口是否满足了所提供的功能，相当于正常情况测试，如果一个接口功能复杂时推荐对接口用例进行结构划分，这样子用例觉有更好的可读性和可维护性；

c) **逻辑测试**：保持内部逻辑的正确性，从设计文档中考虑内部逻辑错误的分之情况和异常；

d) **异常情况测试**：接口实现是否对清楚情况都进行了处理，接口输入参数虽然合法，但是在接口实现中，也会出现异常，因为内部的异常不一定是输入的数据造成的，而有可能是其他逻辑造成的，程序需要对任何异常都进行处理。

下面给大家分享一下实例：

1	获取订单列表接口	getAllOrderList	test-N-所有默认值参数都不填	shopId 参数值：正确且存在； startDate 参数值：正确
2			test-N-带设备token查询	shopId 参数值：正确且存在； token 参数值：正确且存在； startDate 参数值：正确
3			test-N-按订单时间类型查询-下单时间	shopId 参数值：正确且存在； startDate 参数值：正确 dateType 参数值：1
4			test-N-按订单时间类型查询-完成时间	shopId 参数值：正确且存在； startDate 参数值：正确 dateType 参数值：2
5			test-N-按订单时间类型查询-结算时间	shopId 参数值：正确且存在； startDate 参数值：正确 dateType 参数值：3
6			test-N-按查询结束日期查询-单天	shopId 参数值：正确且存在； startDate 参数值：正确 endDate 参数值：与startDate相同
7			test-N-按查询结束日期查询-跨天	shopId 参数值：正确且存在； startDate 参数值：正确 endDate 参数值：与startDate不同
8			test-N-按订单状态查询-单个状态	shopId 参数值：正确且存在； startDate 参数值：正确 orderStatus 参数值：多个状态
9			test-N-按订单状态查询-多个状态	shopId 参数值：正确且存在； startDate 参数值：正确 orderStatus 参数值：多个状态
10			test-N-按交易状态查询	shopId 参数值：正确且存在； startDate 参数值：正确 orderTransactionType 参数值：单个状态
11			test-N-按支付方式查询	shopId 参数值：正确且存在； startDate 参数值：正确 payType 参数值：单种支付方式
12			test-N-按收银员查询-收银员存在	shopId 参数值：正确且存在； startDate 参数值：正确 cashierId 参数值：正确且存在的收银员
13			test-N-按收银员查询-收银员不存在	shopId 参数值：正确且存在； startDate 参数值：正确 cashierId 参数值：正确，但不存在的收银员
14			test-N-按导购员查询-导购员存在	shopId 参数值：正确且存在； startDate 参数值：正确 cashierId 参数值：正确且存在的导购员
15			test-N-按导购员查询-导购员不存在	shopId 参数值：正确且存在； startDate 参数值：正确 cashierId 参数值：正确，但存在的导购员
16			test-N-按页码查询-存在页码	shopId 参数值：正确且存在； startDate 参数值：正确 pNo 参数值：正确且存在的页码
17			test-N-设置页面容量	shopId 参数值：正确且存在； startDate 参数值：正确 pSize 参数值：正确
18			test-N-条件组合	所有参数 参数值：正确，符合规则
19			test-E-按商铺id查询-商铺id不存在	shopId 参数值：正确，但不存在； startDate 参数值：正确
20			test-E-带设备token查询-token不存在	shopId 参数值：正确且存在； token 参数值：正确，但不存在； startDate 参数值：正确
21			test-E-按商铺id查询-商铺id为空	startDate 参数值：正确
22			test-E-按查询起始日期查询-日期为空	shopId 参数值：正确且存在
23			test-E-按时间类型查询-时间类型不在定义范围内	shopId 参数值：正确且存在； startDate 参数值：正确 dateType 参数值：不在定义范围内的合法值
24			test-E-按订单状态查询-订单状态不在定义范围内	shopId 参数值：正确且存在； startDate 参数值：正确 orderStatus 参数值：不在定义范围内的合法值
25			test-E-按订单交易状态查询-交易类型不在定义范围内	shopId 参数值：正确且存在； startDate 参数值：正确 orderTransactionType 参数值：不在定义范围内的合法值
26			test-E-按支付方式查询-payType不在定义范围内	shopId 参数值：正确且存在； startDate 参数值：正确 payType 参数值：不在定义范围内的合法值
27			test-E-查询结束日期小于查询起始时间	shopId 参数值：正确且存在； startDate 参数值：正确 endDate 参数值：小于startDate

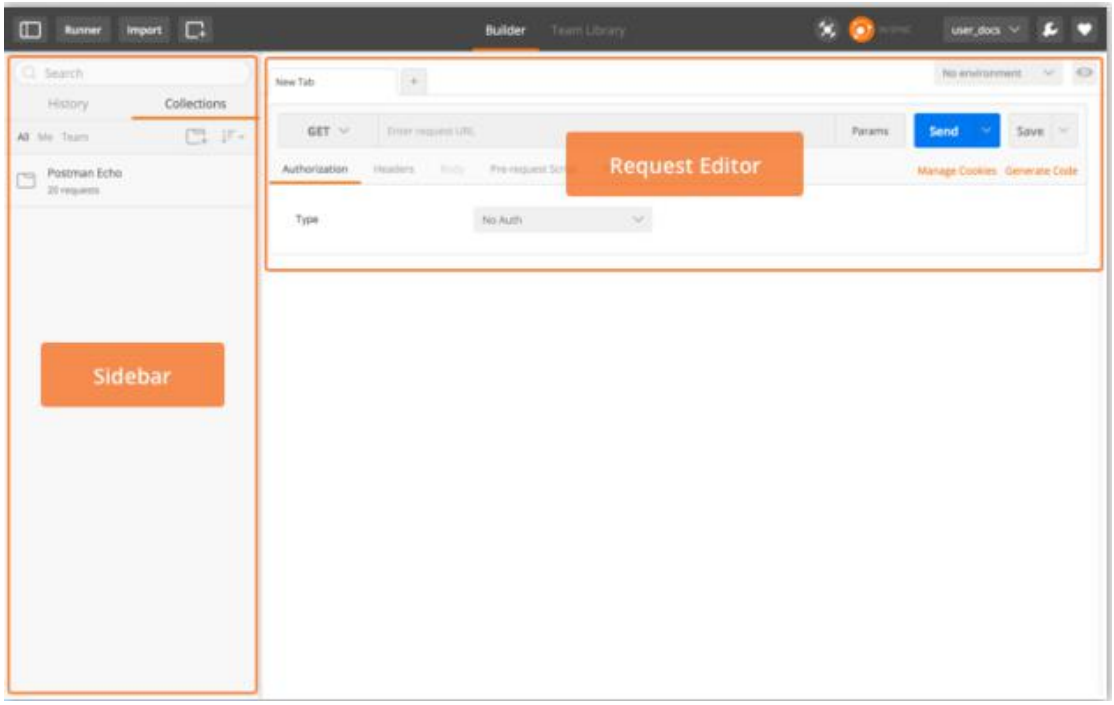
28			test-E-按页码查询-页码不存在	shopId 参数值：正确且存在； startDate 参数值：正确 pSize 参数值：正确 pNo 参数值：正确，但不存在的页码
29			test-E-按商铺id查询-商铺id非int型	shopId 参数值：非int型 startDate 参数值：正确
30			test-E-按设备token查询-token非string类型	shopId 参数值：正确且存在 startDate 参数值：正确 token 参数值：非string类型
31			test-E-按订单时间类型查询-时间类型非int型	shopId 参数值：正确且存在； startDate 参数值：正确 dateType 参数值：非int值
32			test-E-按起始日期查询-时间类型非date型	shopId 参数值：正确且存在； startDate 参数值：非date endDate 参数值：正确
33			test-E-按结束日期查询-时间类型非date型	shopId 参数值：正确且存在； startDate 参数值：正确 endDate 参数值：非date
34			test-E-按订单状态查询-订单状态非string类型	shopId 参数值：正确且存在； startDate 参数值：正确 orderStatus 参数值：非string类型
35			test-E-按交易状态查询-交易状态非int型	shopId 参数值：正确且存在； startDate 参数值：正确 orderTransactionType 参数值：非int值
36			test-E-按支付方式查询-支付方式非int值	shopId 参数值：正确且存在； startDate 参数值：正确 payType 参数值：非int值
37			test-E-按收银员查询-收银员id非int值	shopId 参数值：正确且存在； startDate 参数值：正确 cashierId 参数值：非int值
38			test-E-按导购员查询-导购员id非int值	shopId 参数值：正确且存在； startDate 参数值：正确 cashierId 参数值：非int值
39			test-E-按页码查询-页码非int值	shopId 参数值：正确且存在； startDate 参数值：正确 pNo 参数值：非int值
40			test-E-按商铺id查询-商铺id为空	startDate 参数值：正确
41			test-N-按参数类型最大值查询	所有参数 参数值：参数值存在，且为参数类型最大值，比如32位 int最大值：2147483647
42			test-E-按商铺id查询-商铺id超过类型范围值	shopId 参数值：值超过int类型的最大值 startDate 参数值：正确
43			test-E-按订单状态查询-订单状态值超过类型最大值	shopId 参数值：正确且存在； startDate 参数值：正确 orderStatus 参数值：值超过string类型的最大值
44			test-E-按交易状态查询-交易状态值超过int类型最大值	shopId 参数值：正确且存在； startDate 参数值：正确 orderTransactionType 参数值：非int值
45			略	

接口测试工具简介

了解到这里，基本也就差不多可以开始我们接口测试的修炼了。笔者截止到目前为止，使用过的接口测试的工具，也就是 Jmeter 还仅仅用过两次。其实我是一直在学 python，打算通过脚本去做接口测试来着。不要问我为什么去学！我只是被逼无奈的。不过我也见过听过一些接口测试的工具。今天就在这里介绍给大家。使用方法必须是百度出来的，不喜欢的同学，可以自己去百度，跳过这里哈。

Postman

Postman 是 chrome 浏览器自带的插件。有些同学问我，这个能不能做 APP 的接口测试，我只想说，接口测试只看协议，不分架构，无论你是 C/S 还是 B/S。无论你是 web 还是 APP，只要你是 http 协议，它就能够帮助你完成你的测试。

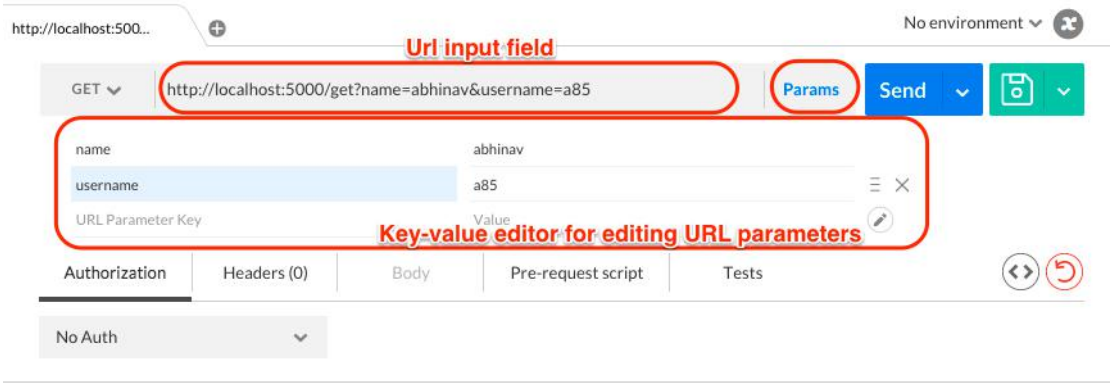


Postman 的界面分为 左边的侧边栏 和 右边的请求构建器 两部分。请求构建器允许你可以快速的创建几乎任何类型的请求。一个 HTTP 请求的四部分：URL、Method、Headers、Body，在 Postman 中都可以设置。

这里简单的和大家介绍一下 Method、URL。

URL

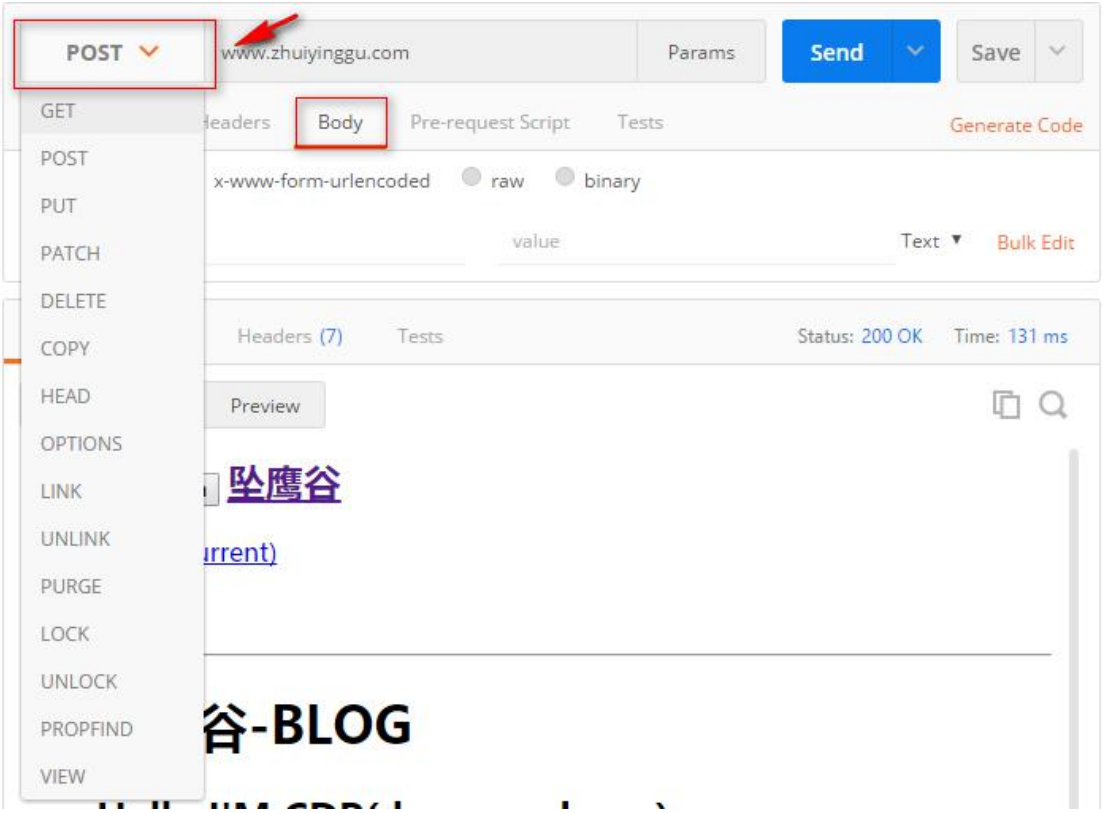
我们在进行 request 请求的时候，首先要设置的就是 URL。在 URL 输入框中输入你请求的链接，你可以单击 Params 按钮，在编辑器中输入 key-value 格式的 URL 参数，直接将参数拼接到 URL 的后面。



注意：在 URL 地址栏中的输入和编辑器中输入的 key-value 参数，不会自动的编码为 URL-encoded，选中要编码的文本，右键选择 EncodeURIComponent，手动编码参数值。

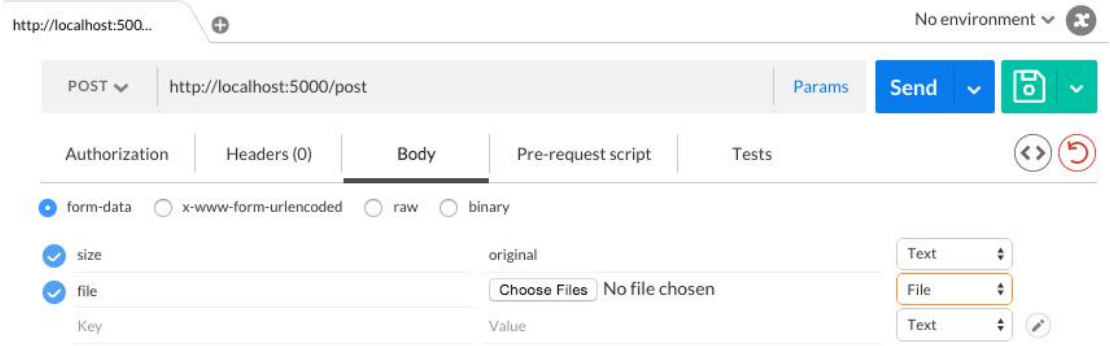
Method

单击 HTTP 的请求方法按钮，在给出的下拉菜单中选择使用的方法即可。



这个请求方法，在接口文档里就有，根据所测试的接口的请求方法来选择就好了。

请求对应的方法需要 body 的，body 部分便被设置为可填写。比如我们选择 post 的时候，就需要去设置 body。get 请求是直接在接口地址后面拼接的参数，而 post 则需要把参数写入到 body 的 form-data 里面。

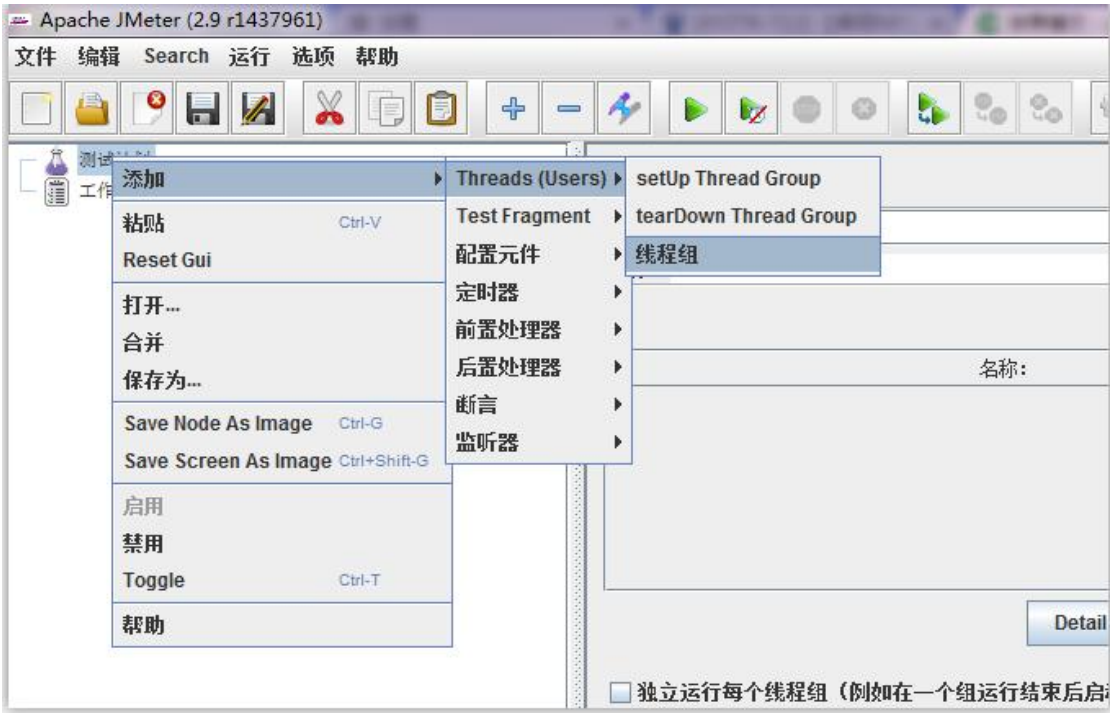


这里也就不在做其他的介绍了。下面我们来看看 Jmeter。

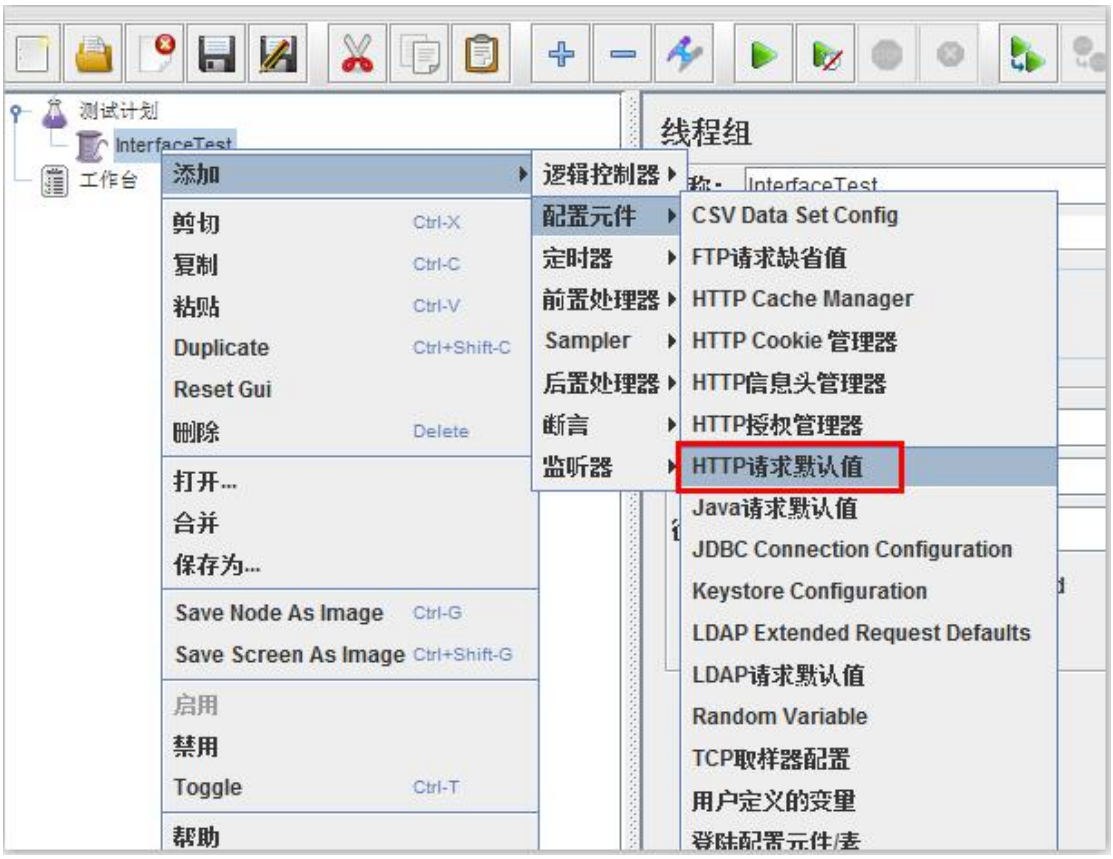
Jmeter

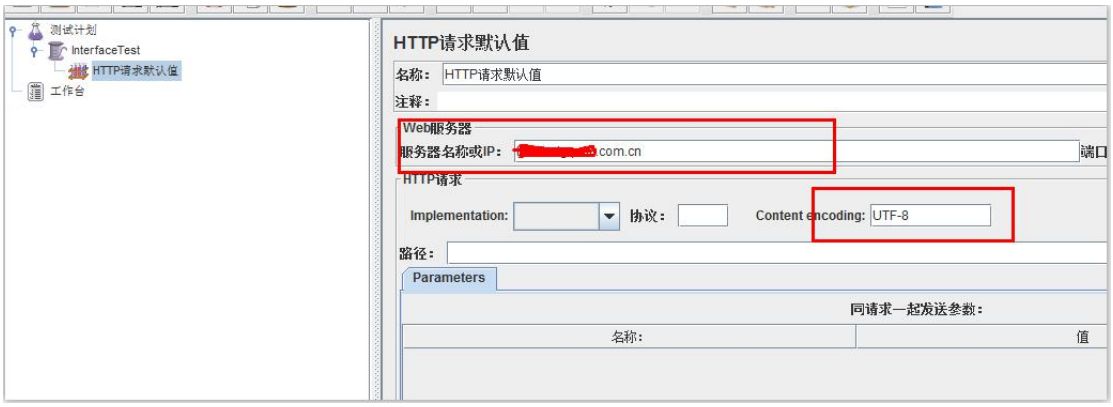
可能大家对 jmeter 更多的认知，是觉得它是一个性能自动化测试工具。但是同时它也是一个接口自动化的测试工具，那么在这里，将会告诉大家如何使用 jmeter 去完成接口测试。

1、首先邮件添加一个线程组，这里我们重命名 InterfaceTest

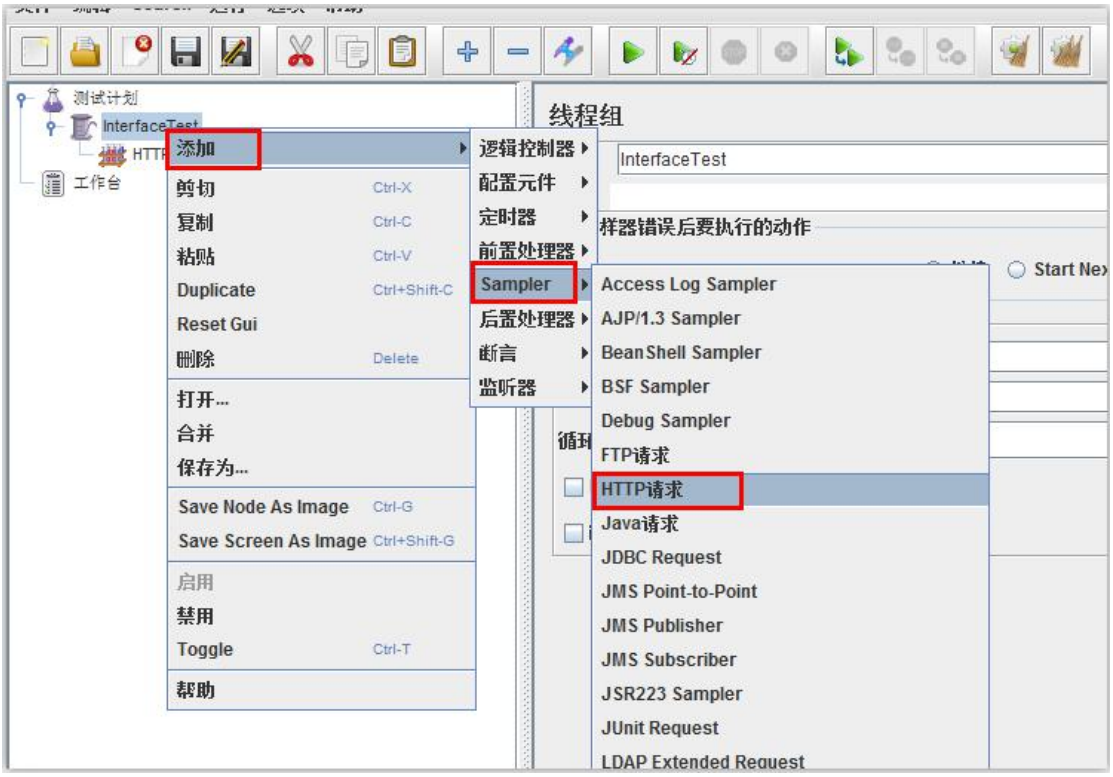


2、在线程组上添加一个 Http 默认请求，并配置服务器的 IP 地址和传输编码

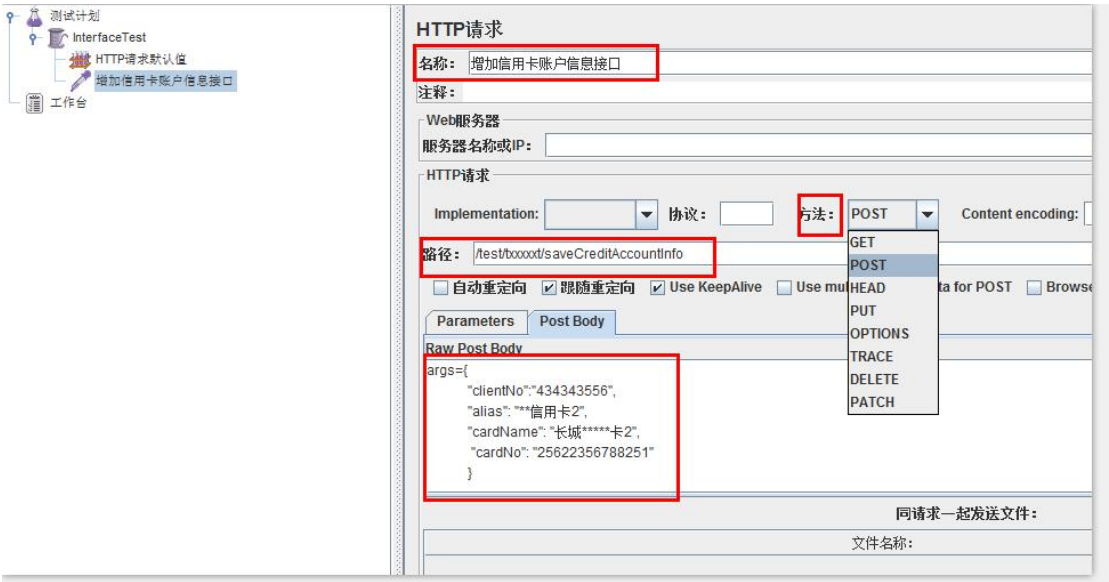




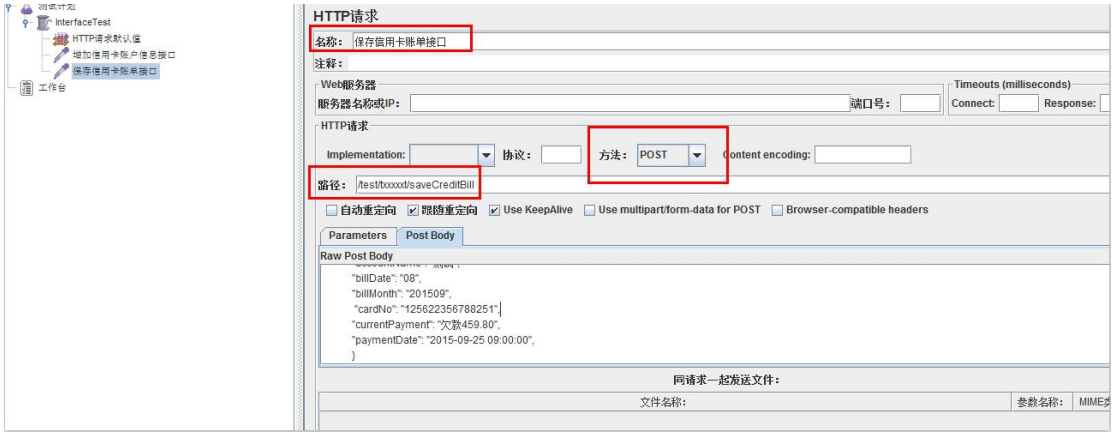
3、在线程组中添加一个 HTTP 请求，这里我们重命名“增加信用卡账户信息接口”



4、配置接口请求信息，这配置示例如下：

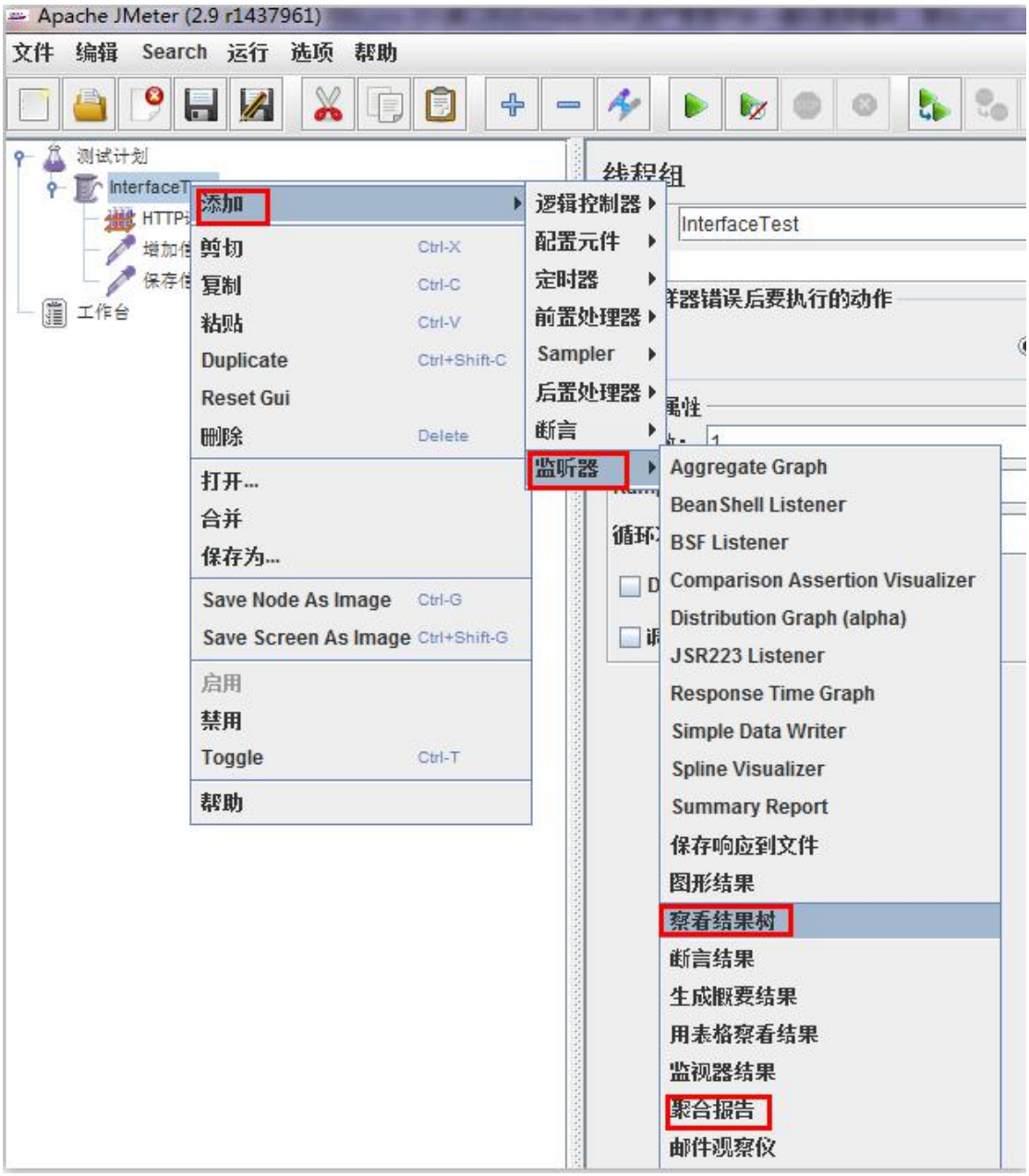


5，在保存信用卡账单接口请求，示例如下：

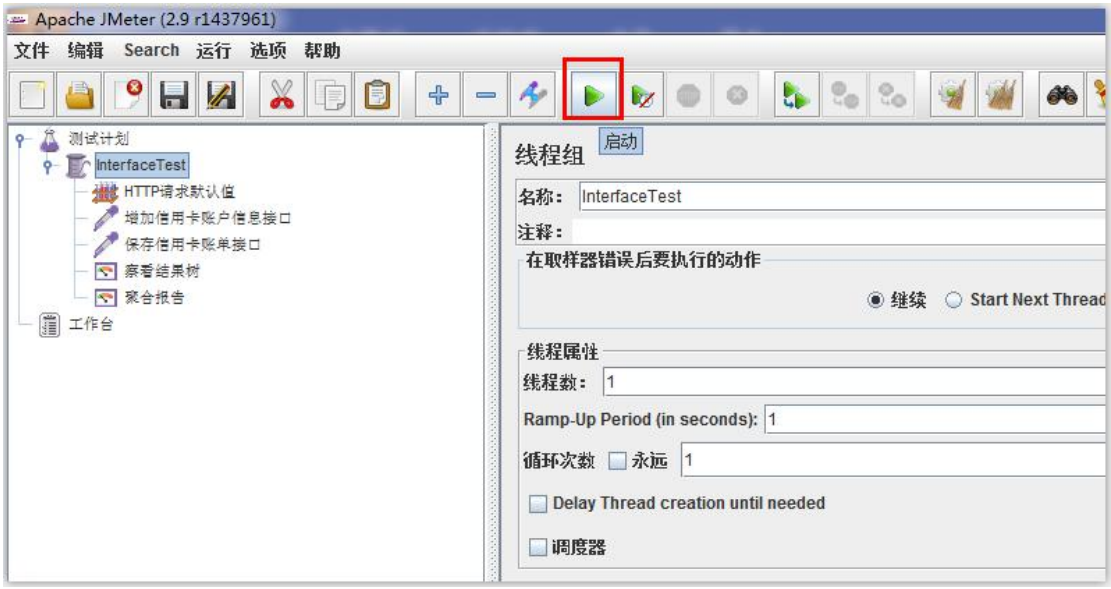


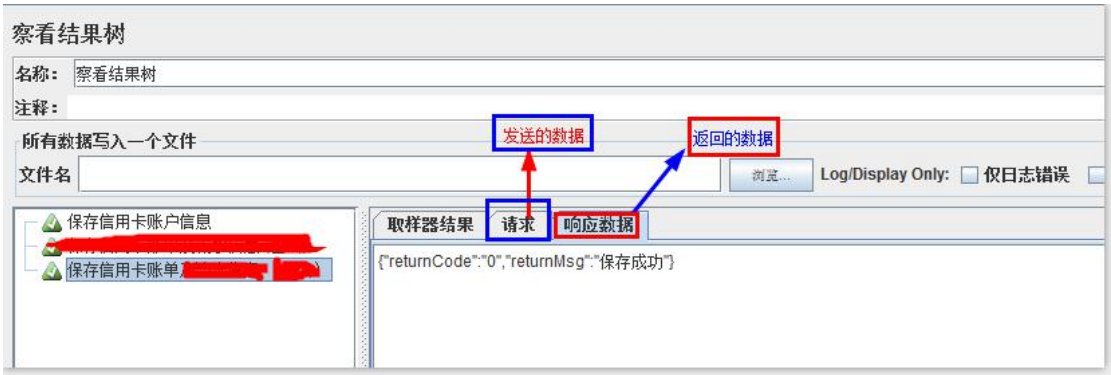
注：由于 Jmeter 请求线程组内的请求时从第一个开始执行，所以我们将需要最先执行的请求放在前面

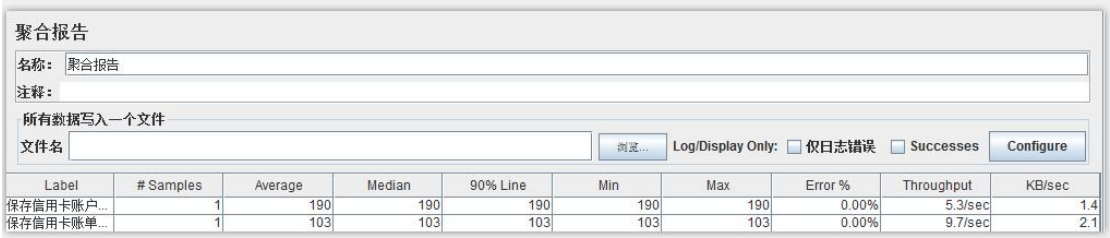
6、在线程组上添加监听器，察看结果树和聚合报告



7、点击启动，运行结束后查看，结果树和聚合报告







Label	# Samples	Average	Median	90% Line	Min	Max	Error %	Throughput	KB/sec
保存信用卡账户...	1	190	190	190	190	190	0.00%	5.3/sec	1.4
保存信用卡账单...	1	103	103	103	103	103	0.00%	9.7/sec	2.1

8、去数据库中核对数据

用来做接口测试的话，应该已经足够了，而且我觉得已经很详细了。虽然工具的用法是我百度找到的。

两年了，回头看看已经走了挺远了。希望大家一起加油吧。诸君，共勉。