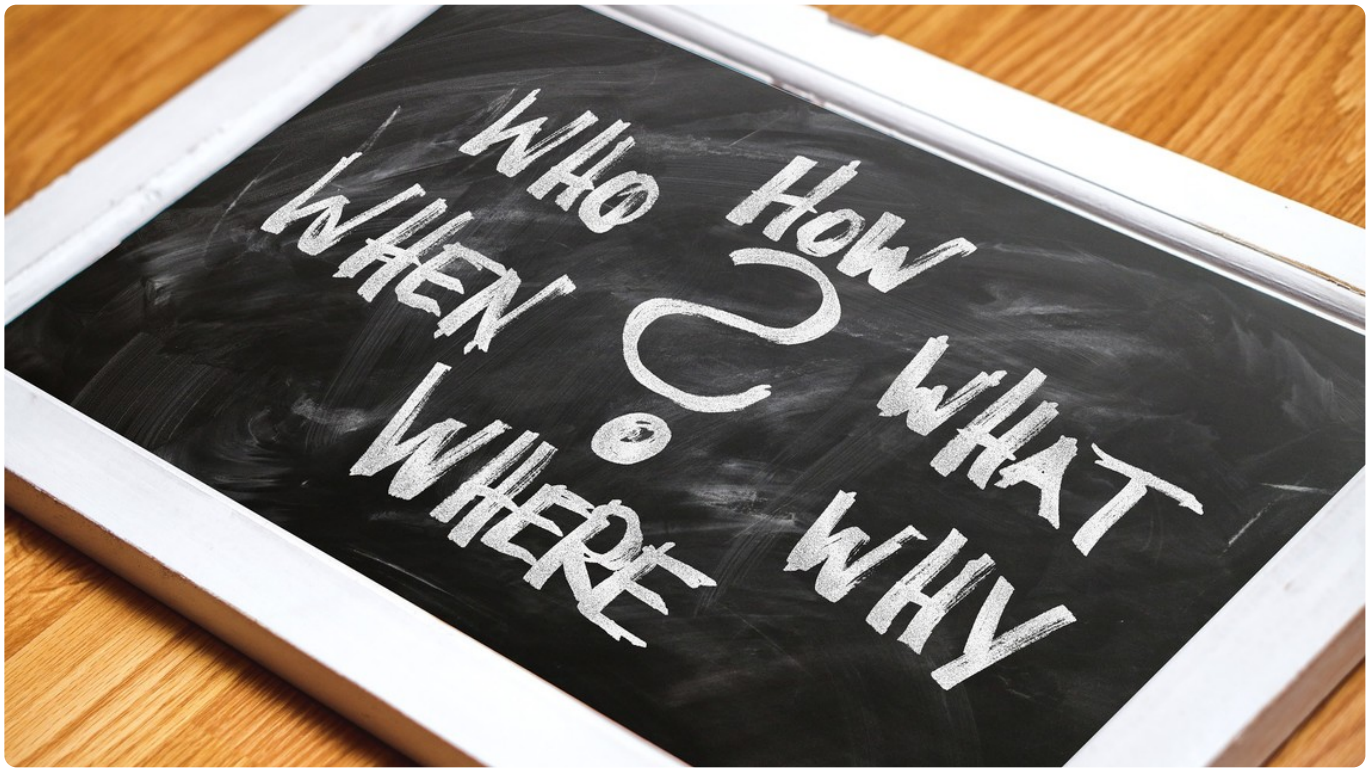
# 测试专栏特别放送 | 答疑解惑第四期

2018-11-05 茹炳晟






你好，我是茹炳晟。

今天的“答疑解惑”文章，我将针对 API 自动化测试和代码级测试这两个系列 6 篇文章中的问题，和你展开分享。

我还是会先简单概括下每篇文章的内容，并给出文章链接，帮助你复习相应的内容。同时，如果你再次阅读时还有哪些疑问的话，也欢迎你在文章下面继续留言。我会一直关注着你的学习情况，希望可以扫清软件测试精进道路上的障碍。

现在，我们就开始今天的主题吧。

## 问题一：实际项目中，往往会存在按时序的 API 调用以及异步 API 调用，这类 API 测试要如何开展？

Postman 对其进行最基本的功能测试，希望可以让你先对 API 测试有个感性认识。另外，在这篇文章中，我还和你分享了目前一些常见的典型复杂场景，以及相应的测试思路和方法。

而在文章最后，我希望你思考的是实际项目中往往会存在按时序的 API 调用以及异步 API 调用，这类 API 测试要如何开展？现在，我来说说我的经验吧。

**我们先一起看看按时序调用的 API 序列的测试场景。**

对于此类测试，我一般建议通过 GUI 操作来录制 API 的调用。比如，在启用 Fiddler 的情况下，通过 GUI 来完成业务操作，随后去分析 Fiddler 抓取到的后端 API 请求顺序，然后以此来开发 API 测试用例。

开发测试用例的过程中还需要特别关注前后两个 API 调用之间的数据传递，比如需要将前一个 API 调用返回的 response 中的某个值作为参数传递给下一个 API 调用。

**其次是异步 API 的测试。**对于异步 API 测试的场景，我们往往先会采取“只验证其是否发起了正确的调用，而不直接验证操作结果”的方式。比如，你的被测 API 是一个异步操作的 API，那么我只会去验证这个 API 是否按照预期发起了正确的异步调用请求，而不会直接去验证异步操作的结果。如果这类测试全部通过后，我们才会考虑真正验证异步操作结果的测试用例。

举个实际的例子，假设你的被测 API A 完成的是下订单的操作。这个 API A 完成下订单操作要通过调用另外一个 API B 将订单信息写入到消息队列中去。而真正下订单成功指的是消息队列中的消息被后续服务正确处理并且成功了。此时，这里的后续消息处理就是异步的操作了。

那么，当我们要测试这个 API A 的时候，我们只需要验证它是否正确地发起了对 API B 的调用即可，而不用关心 API B 的具体行为结果。

也就是说，我们只关注 API A 是否以正确的参数调用了 API B 即可，而无需关注 API B 是否正确地执行了将订单信息写入消息队列的操作，更不用关注，消息队列中的消息被异步处理的结果。

## 问题二：对基于配置文件的 API 测试框架，你有哪些看法呢?

在专栏的第 23 篇文章《知其然知其所以然：聊聊 API 自动化测试框架的前世今生》中，我和你分享了 API 自动化测试框架的发展历程，帮助你理解 API 测试是如何一步一步地发展成今天的样子，希望可以以这种“知其所以然”的方式加深你对 API 自动化测试的理解。

而在这篇文章最后，我提到了基于配置文件的 API 测试框架，比如典型的 HttpRunner。在此类 API 测试框架的支持下，测试用例本身往往就是纯粹的配置文件了。如果你用过这个框架的话，我希望你可以谈谈你的看法。

对于基于配置文件的 API 测试框架的确是个不错的方向，尤其是国内的开源框架 HttpRunner 更是推动了这种测试框架的普及。

**基于配置文件的 API 测试框架的优势，可以归纳为以下三方面**：

1. 降低了测试用例开发的门槛，使得完全没有代码基础的同学也可以很容易地完成 API 测试；
2. 可以很方便地将 API 功能测试用例直接转换成 API 性能测试的用例（HttpRunner 可以使用 lucust 直接实现这样的转换）；
3. 同时，HttpRunner 这类工具还支持直接从网络转发工具得到的 HAR 中提取 API 调用的测试用例，进一步降低了 API 测试用例的开发成本。

所以，基于配置文件的 API 测试框架很受初学者的欢迎。

但是，为了完成一些复杂场景的测试用例设计以及复杂的结果判断，你还是需要具备基本的代码能力，以完成这些复杂场景的测试实现。比如，HttpRunner 就会涉及到使用 debugtalk.py 来实现 hook 函数的功能扩展。

也就是说，完全不写代码的 API 测试只能覆盖大部分的简单测试场景，如果你要搞定复杂场景的 API 测试的话，你是要必须掌握一些基本的开发技能，这里没有任何捷径可走。

另外，很多读者的留言也很精彩，我这里特地选取了两条供大家参考。从 Cynthia 的留言中，我看得出她已经完全习得了这篇文章中描述的方法的精髓，这也正是我想要传达给你的

Cynthia 🌸

写于 2018/08/21

感觉本篇对我最有用的，就是 Response 结果发生变化时的自动识别这块啊！自己在项目中遇到过类似的问题，虽然采取了一定的措施，但是没有想到作者这个解决方案。准备顺着作者的思路捋一捋，看看把他具体实现一下。

> 引自：软件测试52讲
> 23 | 知其然知其所以然：聊聊API自动化测试框架的前世今生

识别二维码打开原文
「极客时间」 App

Martin 在留言的后半部分中提到，通过 HttpRunner 来实现轻量级的 API 测试的确是个好

而且支持各种语言的各种框架，基本可以实现一键操作，所以，其实很多没有采用 HttpRunner 的企业都还在普遍使用这个方法。

Martin 龚平

写于 2018/09/07

用 postman 转 python 或者 java 测试脚本还是太慢了，而且需要一定编程技能，感觉已经是上一代了，我现在根据 httprunner 的 yml 的脚本规则，加上一些开源的组件，做了一个 web 页面可以进行代理抓包，测试人员无论从 web 页面还是 app 操作只要设置代理过来，就可以看到自己的所有请求，然后选择想自动化的请求，后台自动转成测试脚本，再在管理界面上通过拖拽等性质组装成自动化测试集，并可以执行调试、定时任务等。这样的自动化程度基本对编程技能降到了最低，而且生成测试脚本的成本比上一代低了很多

> 引自：软件测试52讲
> 23 | 知其然知其所以然：聊聊API自动化测试框架的前世今生

## 问题三：如果无法通过 API Gateway 方法得到契约的话，应该采用什么方法来解决呢？

在专栏的第 24 篇文章《紧跟时代步伐：微服务模式下 API 测试要怎么做？》中，我和你分享了微服务模式下的 API 测试，旨在帮助你认清庞大的测试用例数量、微服务之间的相互耦合这两个问题的本质，以更好地开展测试。所以，我今天分享这个主题的目的就是，帮你理解这两个问题的本质，以及如何基于消费者契约的方法来解决这两个难题。

而在今天这篇文章最后，我希望你思考的是：基于消费者契约的 API 测试中，对于那些新开发的 API，或者加了新功能的 API，由于之前都没有实际的消费者，所以你无法通过 API Gateway 方法得到契约。对于这种情况，你会采用什么方法来解决呢？

从我的经验来看，因为缺乏契约，所以还是会采用传统的 API 测试方法，也就是根据 API 设计文档来设计测试用例。

这时，我们采取的 API 测试策略是：

> 对于已经上线的 API 我们会通过契约测试来保证质量；
>
> 而对于新的 API，或者是加了新功能的 API，则还是采用传统的基于 API 设计文档来设计测试用例，同时基于代码覆盖率来指导补充遗漏测试用例的方式来保证质量。当这些新 API 上线运行了一段时间后，我们就会缩小测试的范围，逐渐向契约测试过渡。

## 问题四：你所在公司，在进行代码级测试时，采用过哪些方法呢？

在专栏的第 25 篇文章《不破不立：掌握代码级测试的基本理念与方法》中，我根据实际工程项目中的实践，总结了五种常见的代码错误，以及对应的四大类代码级测试方法。这里我还想在多啰嗦一句，代码级测试的测试方法一定是一套测试方法的集合，而不是一个测试方法。

在 eBay，代码质量保障已经完全纳入了 CI/CD 流水线。

首先，我们基于 Sonar 启用了静态代码。除了在上传 Git 的时候触发静态扫描外，开发人员在本地 IDE 中也会进行实时的静态扫描，并可以实时看到分析结果，这样就可以在代码被递交到代码仓库前就已经完成了预检测。

其实，我们并没有直接采用标准的代码静态扫描的规则库，而是删除了其中很多我们认为过于严格的规则，同时加入了一些我们认为比较重要的检测项，使得这个规则库更符合我们的业务场景。一般情况下，这些规则的修订是由测试架构师牵头，与开发主管和资深的开发人员一起协商决定的。

这里需要注意的是，我们并不要求静态扫描上报的所有错误都被修复后才能发布，只要求解决最关键的问题即可。而对于那些所谓的 Code Smell 问题，基于研发成本的考虑，我们并不会要求完全修复。

接着 CI/CD 流水线会触发代码动态测试，即单元测试。这里，我们不仅要求单元测试能够 100% 通过，并且会要求达到一定的代码覆盖率。在 eBay，我们对不同模块的代码覆盖率要求也不一样，并没有一个硬性指标。其实，这也是出于研发成本的考虑。

通常来讲，对底层模块以及提供公共服务的中间件的代码覆盖率指标的要求，一般都会比较高。而我们对前端模块的覆盖率要求，就会低很多。

## 问题五：除了 Sonar，你还用过哪些静态代码扫描工具，使用过程中遇到过哪些问题？

在专栏的第 26 篇文章《深入浅出之静态测试方法》中，我和你详细分享了人工静态测试方法和自动静态测试方法，来帮你理解研发流程上是如何保证代码质量的。另外，我以 Sonar 为例，和你分享了如何搭建自己的自动静态代码扫描方案，并且应用到项目的日常开发工作中去。

而在这篇文章最后，我希望你分享的是除了 Sonar 你还用过哪些静态代码扫描工具，使用过程中遇到过哪些问题。

**第一是误报率。**过高的误报率会降低开发人员对测试工具的信任度，而且还会引入很多人为标注的工作量。

**第二是规则库的完备性和实用性。**很多时候你会发现，标准代码规则库中的一些规则设计不够合理，有点教条主义。比如，有些规则库会强行规定一个函数的代码行数不能超过 200 行，从代码的模块化和易维护性角度来讲，过长的函数实现体的确不利于代码健康。但是，也不能完全一刀切，毕竟有些函数就是实现起来比较复杂。所以，很多时候我们需要对标准规则库进行深层次地裁剪，以更好地适应企业的实际情况。

**第三是自定义规则的难易程度。**虽然很多静态代码工具都提供了规则编辑器，来方便你实现自己的规则，但是这些规则编辑器的使用方法和语法的学习成本比较高，对初学者不够友好。

## 问题六：在单元测试过程中，你都遇到过哪些问题，又是如何解决的呢？

在专栏的第 27 篇文章《深入浅出之动态测试方法》中，我和你分享了人工动态测试方法和自动动态测试方法。因为自动动态方法并不能理解代码逻辑，所以仅仅被用于发现异常、崩溃和超时这类“有特征”的错误，而对于代码逻辑功能的测试，主要还是要依靠人工动态方法。

在这篇文章最后，我希望你分享的是，除了我在文中提到的几个单元测试的难点问题，你还遇到过哪些问题，又是如何解决的。

这里，我想再和你分享我曾在单元测试中遇到过的问题。

**单元测试的难点中较为典型的就是对内部输入的控制。**对于内部输入的控制有数十种不同的场景，这里我就举一个例子，意在抛砖引玉。

首先为了达到较高的测试代码覆盖率，如果代码中包括了 if-else 分支，那么我们的测试就需要分别执行到这两个分支。假设，现在有一个 if-else 分支是根据 malloc 这个内存分配函数的结果进行不同的处理，如果内存分配成功了，就执行 A 逻辑，如果执行失败了就执行 B 逻辑。

控制。

为了解决这个问题，我们就可以采用桩函数的思想，引入一个 malloc 的桩函数，在这个桩函数的内部再去调用真正的系统 malloc 函数，如果需要模拟真正的 malloc 函数的失败，就在桩函数里面直接返回 malloc 函数失败的返回值，来达到模拟真正 malloc 函数失败场景的目的。这样就能在被测函数中通过可控的方式，来模拟系统底层函数的返回值了。

最后，感谢你能认真阅读第 22~27 这 6 篇文章的内容，并写下了你的想法和问题。期待你能继续关注我的专栏，继续留下你对文章内容的思考，我也在一直关注着你的留言、你的学习情况。

感谢你的支持，我们下一期答疑文章再见！



上一篇 测试专栏特别放送 | 答疑解惑第三期

下一篇 测试专栏特别放送 | 答疑解惑第五期

**口水窝**
2019-06-05

打卡

展开

**Alice**
2019-03-08

顾老师，请问:桩函数和MOCK有什么区别么?

展开

**么西卡**
2018-11-05

老师能讲讲测试简历怎么写比较优雅吗。还有大厂面试有哪些经验之谈呢

展开