https://account.geekbang.org/dashboard/buy

1. 关于安全测试的具体实现
2. 关于密码等关键信息存储加密、传输加密
3. 关于测试方法的三大方法：等价类（有效、无效）、边界值、错误推测方法的充分发挥
4. 关于错误推断的定义、现实应用场景：

**定义**：错误推测方法是指基于对被测试软件系统设计的理解、过往经验以及个人直觉，推测出软件可能存在的缺陷，从而有针对性地设计测试用例的方法。这个方法强调的是对被测试软件的需求理解以及设计实现的细节把握，当然还有个人的能力。

**应用场景示例：**比如，Web 界面的 GUI 功能测试，需要考虑浏览器在有缓存和没有缓存下的表现；Web Service 的 API 测试，需要考虑被测 API 所依赖的第三方 API 出错下的处理逻辑；对于代码级的单元测试，需要考虑被测函数的输入参数为空情况下的内部处理逻辑等等。由此可见，这些测试用例的设计都是基于曾经遇到的问题而进行的错误推测，很大程度上取决于个人能力。

1. Wiki缺陷管理-----自检测试用例设计&完善wiki页面的借鉴

在软件企业的具体实践中，为了降低对个人能力的依赖，通常会建立常见缺陷知识库，在测试设计的过程中，会使用缺陷知识库作为检查点列表（checklist），去帮助优化补充测试用例的设计。

对于中小企业，可能最初的方法就是建立一个简单的 wiki 页面，让测试工程师完成测试用例的最初设计后对应这个 wiki 页面先做一轮自检，如果在后续测试中发现了新的点，就会继续完善这个 wiki 页面。

1. 以该缺陷知识库作为数据驱动测试的输入来自动生成部分的测试数据

对于测试基础架构比较成熟的中大型软件企业，通常会以该缺陷知识库作为数据驱动测试的输入来自动生成部分的测试数据，这部分内容我会在后面的文章中详细介绍。

1. 在具体的用例设计时，首先需要搞清楚每一个业务需求所对应的多个软件功能需求点，然后分析出每个软件功能需求点对应的多个测试需求点，最后再针对每个测试需求点设计测试用例。

**业务需求**---》**软件功能需求点**---》**多个测试需求点**---》**测试用例**

1. 只有深入理解被测试软件的架构，你才能设计出“有的放矢”的测试用例集，去发现系统边界以及系统集成上的潜在缺陷。  
   作为测试工程师，切忌不能把整个被测系统看作一个大黑盒，你必须对内部的架构有清楚的认识，比如数据库连接方式、数据库的读写分离、消息中间件 Kafka 的配置、缓存系统的层级分布、第三方系统的集成等等。
2. 必须深入理解被测软件的设计与实现细节，深入理解软件内部的处理逻辑。  
   单单根据测试需求点设计的用例，只能覆盖“表面”的一层，往往会覆盖不到内部的处理流程、分支处理，而没有覆盖到的部分就很可能出现缺陷遗漏。在具体实践中，你可以通过代码覆盖率指标找出可能的测试遗漏点。  
   同时，切忌不要以开发代码的实现为依据设计测试用例。因为开发代码实现的错误会导致测试用例也出错，所以你应该根据原始需求设计测试用例。
3. 需要引入需求覆盖率和代码覆盖率来衡量测试执行的完备性，并以此为依据来找出遗漏的测试点。
4. **单元测试**：如果有任何一个分类遗漏，都会产生缺陷；如果有任何一个分类错误，也会产生缺陷；如果分类正确也没有遗漏，但是分类时的处理逻辑错误，也同样会产生缺陷。

通常来讲，单元测试的用例是一个“输入数据”和“预计输出”的集合。

**驱动代码，桩代码和 Mock 代码：**驱动代码，桩代码和 Mock 代码，是单元测试中最常出现的三个名词。驱动代码是用来调用被测函数的，而桩代码和 Mock 代码是用来代替被测函数调用的真实代码的。

十二、**自动化测试：**自动化测试是，把人对软件的测试行为转化为由机器执行测试行为的一种实践

另外，某些用例的自动化必须要求开发人员在产品中预留可测试性接口，否则后续的自动化会很难开展。比如，有些用户登录操作，需要图片验证码，如果开发人员没有提供绕开图片验证码的路径，那么自动化测试就必须借助光学字符识别（OCR）技术来对图片验证码进行模式识别，而它的设计初衷是为了防止机器人操作，可想而知 OCR 的识别率会很低，就会直接影响用例的稳定性。

十三：**自动化测试：**在软件研发生命周期的各个阶段都有自动化测试技术的存在，并且对提升测试效率有着至关重要的作用。今天这篇文章，我将会以不同的软件开发阶段涉及的自动化测试技术为主线，带你了解单元测试、代码级集成测试、Web Service 测试和 GUI 测试阶段的自动化技术，希望可以帮助你更深入地理解“自动化测试”的内涵以及外延。https://time.geekbang.org/column/article/10572https://time.geekbang.org/column/article/10572

**【1】单元测试:** 从广义上讲，单元测试阶段的“自动化”内涵不仅仅指测试用例执行的自动化，还应该包含以下五个方面：1、用例框架代码生成的自动化；2、部分测试输入数据的自动化生成；3、自动桩代码的生成；4、被测代码的自动化静态分析；5、测试覆盖率的自动统计与分析。

【2】**代码级集成测试：**代码级集成测试与单元测试最大的区别只是，代码级集成测试中被测函数内部调用的其他函数必须是真实的，不允许使用桩代码代替，而单元测试中允许使用桩代码来模拟内部调用的其他函数。但是，代码级集成测试对测试框架的要求非常高，这个框架除了可以顺利装载自己的软件模块外，还必须能装载其他相互依赖的模块，做到被测软件模块可运行（Runnable）。由于代码级集成测试主要应用在早期非互联网的传统软件企业，那时候的软件以“单体”应用居多，一个软件内部包含大量的功能，每一个软件功能都是通过不同的内部模块来实现的，那么这些内部模块在做集成的时候，就需要做代码级集成测试。由于代码级集成测试主要应用在早期非互联网的传统软件企业，那时候的软件以“单体”应用居多，一个软件内部包含大量的功能，每一个软件功能都是通过不同的内部模块来实现的，那么这些内部模块在做集成的时候，就需要做代码级集成测试。

【3】**web service测试**：Web Service 测试，主要是指 SOAP API 和 REST API 这两类 API 测试，最典型的是采用 SoapUI 或 Postman 等类似的工具。但这类测试工具基本都是界面操作手动发起 Request 并验证 Response，所以难以和 CI/CD 集成，于是就出现了 API 自动化测试框架。基于代码的API测试通常包含以下三个步骤：1、准备 API 调用时需要的测试数据；2、准备 API 的调用参数并发起 API 的调用；3、验证 API 调用的返回结果。目前最流行的 API 自动测试框架是 REST Assured，它可以方便地发起 Restful API 调用并验证返回结果。同样地，Web Service 测试“自动化”的内涵不仅仅包括 API 测试用例执行的自动化，还包括以下四个方面：1、测试脚手架代码的自动化生成；2、部分测试输入数据的自动生成；3、Response 验证的自动化；4、基于 SoapUI 或者 Postman 的自动化脚本生成。

【4】**GUI测试：**GUI 自动化测试主要分为两大方向，传统 Web 浏览器和移动端原生应用（Native App）的 GUI 自动化。虽然二者采用的具体技术差别很大，但是用例设计的思路类似。1、对于传统 Web 浏览器的 GUI 自动化测试，业内主流的开源方案采用 Selenium，商业方案采用 Micro Focus 的 UFT（前身是 HP 的 QTP）；2、对于移动端原生应用，通常采用主流的 Appium，它对 iOS 环境集成了 XCUITest，对 Android 环境集成了 UIAutomator 和 Espresso。

十四、**测试覆盖率：**测试覆盖率主要分为两大类，一类是面向项目的需求覆盖率，另一类是更偏向技术的代码覆盖率。

1. 需求覆盖率： 我们通常采用 ALM，Doors 和 TestLink 等需求管理工具来建立需求和测试的对应关系，并以此计算测试覆盖率。比较适用于传统瀑布模型下的软件工程，互联网测试项目中很少直接基于需求来衡量测试覆盖率，而是将软件需求转换成测试需求，然后基于测试需求再来设计测试点。因此，现在人们口中的测试覆盖率，通常默认指代码覆盖率，而不是需求覆盖率。
2. 代码覆盖率：简单来说，是指，至少被执行了一次的条目数占整个条目数的百分比。

代码覆盖率工具以针对java的jacoco为例讲解：

<https://time.geekbang.org/column/article/10759>

很多敏捷团队可能没有完全文档化的测试用例，所以会比较依赖团队成熟度以及一些BDD和TDD的工具，关于什么是bdd和tdd，后面会专门来讲，这两者都属于迭代开发下的敏捷实践

关于安卓智能手机代码覆盖率; Android SDK内置了Emma test coverage，你可以直接用，不过还是比较推荐Jacoco

十五、**软件缺陷报告：**缺陷报告本身的质量将直接关系到缺陷被修复的速度以及开发工程师的效率，同时还会影响测试工程师的信用、测试与开发人员协作的有效性。1、缺陷标题；2、缺陷概述3、缺陷影响4、环境配置5、前置条件6、缺陷重现步骤7、期望结果和实际结果8、优先级和严重程度9、变通方案10、根原因分析11、附件

十五、**测试执行计划：**敏捷开发模式下，测试计划依旧存在，只是从原来的一次性集中制定测试计划，变成了以迭代的方式持续制定测试计划。测试执行计划需要明确：【1】测试范围：测什么和不测什么【2】

测试策略：先测什么后测什么以及怎样测试，采取什么测试类型和测试方法【3】测试资源：谁来测、在哪里测【4】测试进度：在敏捷模式下，测试活动贯穿于整个开发过程，很多测试工作会和开发工作同步进行，比如采用行为驱动开发（Behavior-Driven Development）模式，这样测试进度就不会完全依赖于开发递交可测试版本的时间。行为驱动开发，就是平时我们经常说的 BDD，指的是可以通过自然语言书写非程序员可读的测试用例，并通过 StepDef 来关联基于自然语言的步骤描述和具体的业务操作，最典型的框架就是知名“Cucumber”。、；【5】测试风险评估

十五：**软件测试工程师的核心竞争力：**

1. **传统测试工程师应该具备的核心竞争力：**
2. 测试策略设计能力：对于各种不同的被测软件，能够快速准确地理解需求，并在有限的时间和资源下，明确测试重点以及最适合的测试方法的能力。
3. 测试用例设计能力：无论对于什么类型的测试，都能设计出高效地发现缺陷，保证产品质量的优秀测试用例。
4. 快速学习能力：对不同业务需求和功能的快速学习和理解能力；对于测试新技术和新方法的学习和应用能力。
5. 探索性测试思维：测试工程师在执行测试的过程中不断学习被测系统，同时结合基于自己经验的错误猜测和逻辑推理，整理和分析出更多的有针对性的测试关注点。
6. 缺陷分析能力：1）对于已经发现的缺陷，结合发生错误的上下文以及后台日志，可以预测或者定位缺陷的发生原因，甚至可以明确指出具体出错的代码行，由此可以大幅缩短缺陷的修复周期，并提高开发工程师对于测试工程师的认可以及信任度；2）根据已经发现的缺陷，结合探索性测试思维，推断同类缺陷存在的可能性，并由此找出所有相关的潜在缺陷；3）可以对一段时间内所发生的缺陷类型和趋势进行合理分析，由点到面预估整体质量的健康状态，并能够对高频缺陷类型提供系统性的发现和预防措施，并以此来调整后续的测试策略。
7. 自动化测试技术：解放劳动力，但是切记，自动化测试的核心价值还是“测试”本身，“自动化”仅仅是手段，实际工作中千万不要本末倒置，把大量的精力放在“自动化”上，一味追求自动化而把本质的“测试”弱化了。
8. 良好的沟通能力：与产品经理与开发的沟通协调能力
9. **测试开发工程师的核心竞争力：**
10. 测试系统需求分析能力：除了代码开发能力，测试开发工程师更要具备测试系统需求分析的能力。你要能够站在测试架构师的高度，识别出测试基础架构的需求和提高效率的应用场景。从这个角度说，你更像个产品经理，只不过你这个产品是为了软件测试服务的。
11. 更宽广的知识体系：测试开发工程师需要具备非常宽广的知识体系，你不仅需要和传统的测试、开发工程师打交道，因为他们是你构建的测试工具或者平台的用户；而且还要和 CI/CD、和运维工程师们有紧密的联系，因为你构建的测试工具或者平台，需要接入到 CI/CD 的流水线以及运维的监控系统中去。

十六：**软件工程师需要掌握的非测试知识：**

**小到** Linux/Unix/Windows 操作系统的基础知识，Oracle/MySQL 等传统关系型数据库技术，NoSQL 非关系型数据库技术，中间件技术，Shell/Python 脚本开发，版本管理工具与策略，CI/CD 流水线设计，F5 负载均衡技术，Fiddler/Wireshark/Tcpdump 等抓包工具，浏览器 Developer Tool 等；

**大到**网站架构设计，容器技术，微服务架构，服务网格（Service Mesh），DevOps，云计算，大数据，人工智能和区块链技术等。

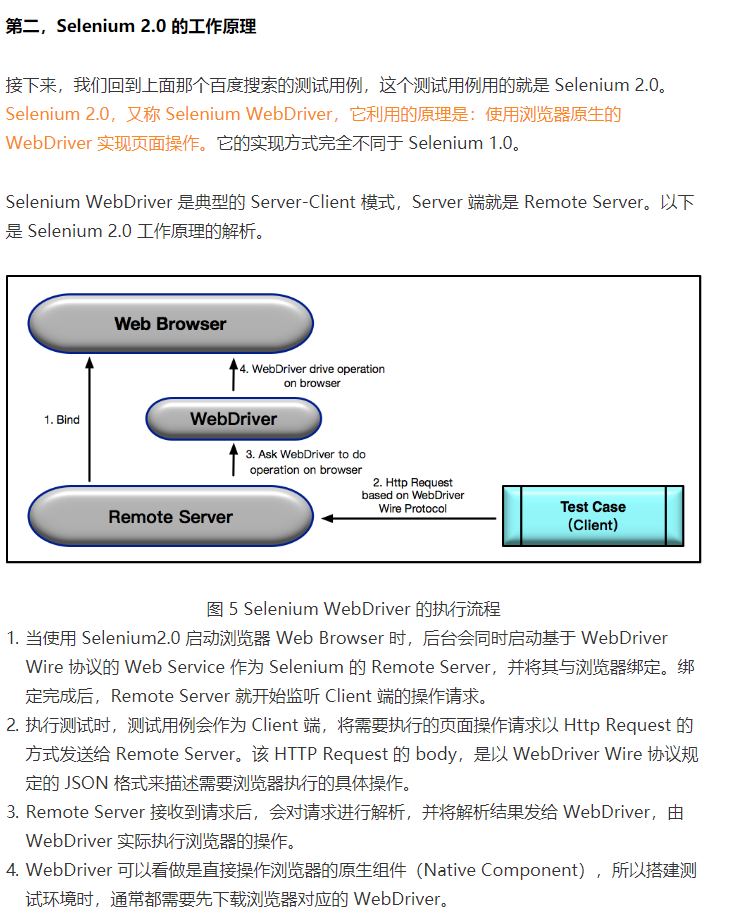
1. **网站架构的核心知识：**所以，我强烈建议你要掌握网站架构的核心知识，你不需要像系统架构师那样能够熟练驾驭各种架构，并根据业务选型，但你至少需要理解架构相关的基本知识以及核心原理。基于此，我在专栏的最后安排了一系列文章，包括了**网站高性能架构设计、网站高可用架构设计、网站伸缩性架构设计和网站可扩展性架构设计**，为你详细讲解互联网架构的核心知识，提升你的互联网产品测试能力。
2. **容器技术：**很多中大型互联网企业都在推行容器化开发与运维，开发人员递交给测试工程师的软件版本通常就是一个 Docker Image，直接在容器上进行测试。有些公司还会把测试用例和执行框架也打包成 Docker Image，配合版本管理机制，实现用容器测试容器。

[**https://docs.docker.com/get-started/**](https://docs.docker.com/get-started/)

1. **云计算技术：**一方面，很多企业，尤其是互联网企业都在尝试“上云”， 也就是逐渐把生产环境从原本的集中式数据中心模式转向私有云或者混合云模式。另一方面，测试基础服务作为提供测试服务的基础设施，比如测试执行环境服务（Test Execution Service）和测试数据准备服务（Test Data Service）等，也在逐渐走向云端。比如，国外非常流行的 Sauce Labs，就是一个著名的测试执行环境公有云服务。
2. **DevOps思维：**DevOps 强调的是，开发、测试和运维等组织团队之间，通过高效自动化工具的协作和沟通，来完成软件的全生命周期管理，从而实现更频繁地持续交付高质量的软件，其根本目的是要提升业务的交付能力。
3. **前端开发技术：**从测试工程师的角度来讲，如果你能够掌握前端开发技术，也就意味着你可以更高效地做前端的测试，更容易发现潜在缺陷。同时，你还可以自己构建测试页面，来完成各类前端组件的精细化测试，大大提高测试覆盖率和效率。
4. **人工智能、机器学习：**现在其实已经有些应用了，但是没有大规模落地的实际案例，比如测试失败用例的自动分类，MBT基于模型的遍历算法，基于图像特征的页面控件识别等等

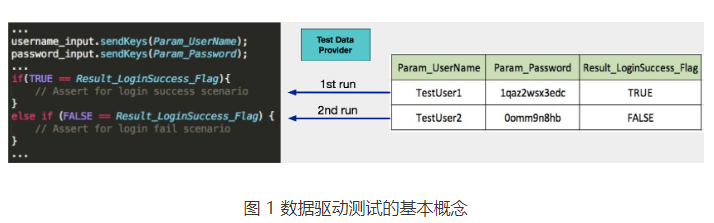
十七、**简单的GUI测试：**传统软件的GUI自动化测试推荐UFT，互联网的推荐SELENIUM，关于 Selenium 1.0 和 Selenium 2.0 的内部实现机制和原理：Selenium 1.0 的核心是，基于 JavaScript 代码注入；而 Selenium 2.0 的核心是，运用了浏览器原生支持的 WebDriver。

<https://time.geekbang.org/column/article/11913> java+selenium打开百度搜索示例



十八、**脚本与数据的解耦 + Page Object模型：**

**【1】测试脚本和数据解耦**：本质是实现了数据驱动的测试，让操作相同但是数据不同的测试可以通过同一套自动化测试脚本来实现，只是在每次测试执行时提供不同的测试输入数据。



1. **页面模型（Page Object）：**页面对象模型的核心理念是，以页面（Web Page 或者 Native App Page）为单位来封装页面上的控件以及控件的部分操作。而测试用例，更确切地说是操作函数，基于页面封装对象来完成具体的界面操作，最典型的模式是“XXXPage.YYYComponent.ZZZOperation”。 通过这样的代码结构，你可以清楚地看到是在什么页面执行什么操作，代码的可读性以及可维护性大幅度提高，也可以更容易地将具体的测试步骤转换成测试脚本。

**另：模块化设计：**利用模块化思想，把一些通用的操作集合打包成一个个名字有意义的函数，然后 GUI 自动化脚本直接去调用这些操作函数来构成整个测试用例，这样 GUI 自动化测试脚本就从原本的“流水账”过渡到了“可重用脚本片段”。

**十九、让自动化测试脚本更好的描述业务：**以“如何把控操作函数的粒度，以及如何衔接两个操作函数之间的页面”这两个问题作为引子，为你介绍 GUI 自动化测试中“业务流程（business flow）”的概念、核心思想以及应用场景。

1. **如何把控操作函数的粒度**：操作函数的粒度是指，一个操作函数到底应该包含多少操作步骤才是最合适的。

那么，操作函数的粒度到底应该如何控制呢？其实这个问题，在很大程度上取决于项目的实际情况，以及测试用例步骤的设计，并没有一个放之四海而皆准的绝对标准。但是，脚本粒度的控制还是有设计依据可以遵循的，往往以完成一个业务流程（business flow）为主线，抽象出其中的“高内聚低耦合”的操作步骤集合，操作函数就由这些操作步骤集合构成。也就是说，业务流程会依次调用各个操作函数，来完成具体的业务操作。

1. **如何衔接两个操作函数之间的页面：**完成一个业务流程操作，往往会需要依次调用多个操作函数，但是操作函数和操作函数之间会有页面衔接的问题，即前序操作函数完成后的最后一个页面，必须是后续操作函数的第一个页面。如果连续的两个操作函数之间无法用页面衔接，那就需要在两个操作函数之间加入额外的页面跳转代码，或者是在操作函数内部加入特定的页面跳转代码。

**引入概念：**

1. **业务流程抽象：**业务流程抽象是，基于操作函数的更接近于实际业务的更高层次的抽象方式。基于业务流程抽象实现的测试用例往往灵活性会非常好，你可以很方便地组装出各种测试用例。

然后，对于每一个业务流程类，都会有相应的业务流程输入参数类与之一一对应。具体的步骤通常有这么几步：1）初始化一个业务流程输入参数类的实例；2）给这个实例赋值；3）用这个输入参数实例来初始化业务流程类的实例；4）执行这个业务流程实例。

1. **业务流程的优点：**1）业务流程（Business Flow）的封装更接近实际业务；2）基于业务流程的测试用例非常标准化，遵循“参数准备”、“实例化 Flow”和“执行 Flow”这三个大步骤，非常适用于测试代码的自动生成；3）由于更接近实际业务，所以可以很方便地和 BDD 结合。BDD 就是 Behavior Driven Development，即行为驱动开发，我会在后续文章中详细讲解。

**二十、自动化测试过程中的测试数据准备：**

**GUI测试中常见的两种测试数据：**

1. 第一大类是，测试输入数据，也就是 GUI 测试过程中，通过界面输入的数据。比如“用户登录”测试中输入的用户名和密码就就属于这一类数据；再比如，数据驱动测试中的测试数据，也是指这一类。
2. 第二大类是，为了完成 GUI 测试而需要准备的测试数据。比如，“用户登录”测试中，我们需要事先准备好用户账户，以便进行用户的登录测试。今天我分享的测试数据创建的方法，也都是围着这一部分的数据展开的。

**创建测试数据的方法：**

**从创建的技术来讲：**

1. **API调用【2】数据库操作【3】综合运用API调用和数据库操作**

**从创建的时机来讲：**

1. 测试用例执行过程中，实时创建测试数据，我们通常称这种方式为 On-the-fly。
2. 测试用例执行前，事先创建好“开箱即用”的测试数据，我们通常称这种方式为 Out-of-box。

**注意，API调用、数据库操作的综合运用；操作涉及多个业务表，数据库操作的测试数据工具的版本管理问题；关于测试数据创建时机**On-the-fly 和 Out-of-box 的互补

**二十一、脑洞大开：GUI测试还能这么玩（Page Code Gen + Data Gen + Headless）：**今天这篇文章，我将从页面对象自动生成、GUI 测试数据自动生成、无头浏览器三个方面展开，这也是 GUI 测试中三个比较有意思的知识点。

【1】**页面对象自动生成：**页面对象自动生成技术，属于典型的“自动化你的自动化”的应用场景。它的基本思路是，你不用再手工维护 Page Class 了，只需要提供 Web 的 URL，它就会自动帮你生成这个页面上所有控件的定位信息，并自动生成 Page Class。

**【2】GUI测试数据自动生成：**GUI 测试数据自动生成，指的由机器自动生成测试用例的输入数据。

1. **根据 GUI 输入数据类型，以及对应的自定义规则库自动生成测试输入数据**：比如，GUI 界面上有一个“书名”输入框，它的数据类型是 string。基于数据类型就可以自动生成诸如 Null、SQL 注入、超长字符串、非英语字符等测试数据。同时，自定义规则库，还可以根据具体规则生成各种测试数据。规则，往往反映了具体的业务逻辑。比如，对于“书名”，就会有书名不能大于多少个字符、一些典型的书名（比如，英文书名、中文书名等）等等业务方面的要求，那么就可以根据这些业务要求来生成测试数据。根据自定义规则生成测试数据的核心思想，与安全扫描软件 AppScan 基于攻击规则库自动生成和执行安全测试的方式，有异曲同工之处。
2. **对于需要组合多个测试输入数据的场景，测试数据自动生成可以自动完成多个测试数据的笛卡尔积组合，然后再以人工的方式剔除掉非法的数据组合：**但是，这种方式并不一定是最高效的。对于输入参数比较多，且数据之间合法组合比较少或者难以明确的情况，先自动化生成笛卡尔积组合，再删除非法组合，效率往往还不如人为组合来得高。更常见的用法是，先手动选择部分输入数据进行笛卡尔积，并删除不合法的部分；然后，在此基础上，再人为添加更多业务上有意义的输入数据组合。
3. **无头浏览器：**无头浏览器，其实是一个特殊的浏览器，你可以把它简单地想象成是运行在内存中的浏览器。它拥有完整的浏览器内核，包括 JavaScript 解析引擎、渲染引擎等。与普通浏览器最大的不同是，无头浏览器执行过程中看不到运行的界面，但是你依然可以用 GUI 测试框架的截图功能截取它执行中的页面。

无头浏览器并不完美，它最大的缺点是，不能完全模拟真实的用户行为，而且由于没有实际完成页面的渲染，所以不太适用于需要对于页面布局进行验证的场景。同时，业界也一直缺乏理想的无头浏览器方案。

但好消息是，2017 年 Google 发布了 Headless Chrome，以及与之配套的 Puppeteer 框架，Puppeteer 不仅支持最新版本的 Chrome，而且得到 Google 官方的支持，这使得无头浏览器可以在实际项目中得到更好的应用。

1.如果使用 selenium + xpath，应该可以解决大部分的控件识别，chrome 支持右键一键拷贝 xpath，贼方便；

2、测试数据自动生成主要是思想方法，用excel➕vba就能轻易完成大部分的功能，当然用代码实现也是可以的

工具：katalon、robotframe

**二十二、提高GUI测试稳定性的关键技术：**GUI 自动化测试稳定性，最典型的表现形式就是，同样的测试用例在同样的环境上，时而测试通过，时而测试失败。以下介绍造成不稳定的**主要五个因素以及解决方案**：

1. **非预计的弹出对话框**：1）GUI 自动化测试用例执行过程中，操作系统弹出的非预计对话框；2）被测软件本身也有可能在非预期的时间弹出预期的对话框，

**解决方案：**当自动化脚本发现控件无法正常定位，或者无法操作时，GUI 自动化框架自动进入“异常场景恢复模式”。 在“异常场景恢复模式”下，GUI 自动化框架依次检查各种可能出现的对话框，一旦确认了对话框的类型，立即执行预定义的操作（比如，单击“确定”按钮，关闭这个对话框），接着重试刚才失败的步骤。

1. **页面控件的属性变化：**比如，当“登录”按钮的 ID 从“Button\_Login\_001”变成了 “Button\_Login\_888”

因此，采用**“组合属性”定位控件**会更精准，而且成功率会更高，如果能在此基础上加入**“模糊匹配”**技术，可以进一步提高控件的识别率。

1. **被测系统的 A/B 测试：**A/B 测试，是互联网产品常用的一种测试方法。它为 Web 或 App 的界面或流程提供两个不同的版本，然后让用户随机访问其中一个版本，并收集两个版本的用户体验数据和业务数据，最后分析评估出最好的版本用于正式发布。A/B 测试通常会发布到实际生产环境，所以就会造成生产环境中 GUI 自动化测试的不稳定。
2. **随机的页面延迟造成控件识别失败：**对于这种随机的页面延迟，办法就是，加入重试（retry）机制。重试机制是指，当某一步 GUI 操作失败时，框架会自动发起重试，重试可以是步骤级别的，也可以是页面级别的，甚至是业务流程级别的。

对于开源 GUI 测试框架，重试机制往往不是自带的功能，需要自己二次开发来实现。比如，eBay 的 GUI 自动化测试框架，分别实现了步骤级别、页面级别和业务流程级别的重试机制，默认情况下启用的是步骤级别的重试，页面级别和业务流程级别的重试可以通过测试发起时的命令行参数进行指定。

**【5】测试数据问题：**比如，测试用例所依赖的数据被其他用例修改了；再比如，测试过程中发生错误后自动进行了重试操作，但是数据状态已经在第一次执行中被修改了。 后续章节补充。。。。。。。。。。。。。。

**问答环节：**作者回复: 步骤和页面级别的retry是会在测试框架中实现的，往往是在try catch中实现重试，而用例级别的retry会在用例调用级别，也就是发起测试的ci流水线中实现。

1. 对于非预期的弹框也可以通过检查置顶窗口是否是预期软件窗口，从而确定是否被第三方弹框影响。
2. 页面控件变化：如果是 selenium 的话，建议优先使用 xpath，这样就算 id、clases、name 等控件属性改变，只有不是页面改版，应该不会影响自动化稳定性。
3. A/B页面：判断当前页面属于哪个分支，然后走兼任处理逻辑，同意这个方案。其实很多地方都可以通过类似的兼容方案进行处理，比如第一个非预期弹框，也可以算是异常场景的兼容处理。
4. **页面延迟：**重试机制确实是个好办法，但是如果用例都是因为重试才执行正确，有可能会漏出和缓存相关的问题，因为重试应该算一个独立测试场景了，现在是把它作为主要测试场景了。这地方我记得 selenium 有一个函数可以设定一个最大超时时间，在这个时间之前都会等待，一旦超时时间内满足了继续执行的条件，也可以立刻执行，这个方法还是比较不错的，既保证了用例操作的预期性，也解决了延迟的不可控的问题。
5. **测试数据问题：**构造自动化数据时要特别注意，构造一些带特殊字段的数据库信息，最好是超出常人操作的数据信息，这样可以有效避免数据被误修改的风险，当然，还有一个处理办法在 15 讲的时候提到过，就是先检查测试数据是否存在/异常，不存在或异常都**进行重建**即可，这部分也算是测试代码的兼容处理吧。

**二十三、GUI自动化测试的报告：**理想中的 GUI 测试报告应该是由一系列按时间顺序排列的屏幕截图组成，并且这些截图上可以高亮显示所操作的元素，同时按照执行顺序配有相关操作步骤的详细描述。

目前，商业的 GUI 自动化测试软件，比如使用最为广泛的 UFT（就是以前的 QTP），已经自带了截图以及高亮显示操作元素功能。也就是说，使用 UFT 执行一个 GUI 自动化测试用例，你无需做任何额外的工作，就能得到一份比较理想的 GUI 测试报告。

**开源 GUI 测试框架的测试报告实现思路：**

但是，如果你使用的是开源软件，比如 Selenium WebDriver，那就需要自己去实现截图以及高亮显示操作元素的功能。实现的思路通常是：利用 Selenium WebDriver 的 **screenshot 函数**在一些特定的时机（比如，页面发生跳转时，在页面上操作某个控件时，或者是测试失败时，等等）完成界面截图功能。

具体的代码实现，主要有两种方式：1、**扩展 Selenium 原本的操作函数；2、在相关的 Hook 操作中调用 screenshot 函数。**

**【1】扩展 Selenium 原本的操作函数实现截图以及高亮显示操作元素的功能：**实现一个自己的click函数。

1. 首先，用 Javascript 代码高亮显示被操作的元素，高亮的实现方式就是利用 JavaScript 在对象的边框上渲染一个 5-8 个像素的边缘；
2. 然后，调用 screenshot 函数完成点击前的截图；
3. 最后，调用 Selenium 原生的 click 函数完成真正的点击操作。

那么，以后凡是需要调用 click 函数时，都直接调用这个自己封装的 click 函数，直接得到高亮了被操作对象的界面截图。

1. **在相关的 Hook 操作中调用 screenshot 函数实现截图以及高亮显示操作元素的功能：**当执行某个函数 F 时，系统会在执行函数 F 前先隐式执行一个空实现的函数，那么当你需要做一些扩展或者拦截时，就可以在这个空实现的函数中加入自定义的操作了。那么这个空实现的函数就是所谓的 Hook 函数。

说到这里，你可能已经知道要怎么做了：我可以在这些 Hook 函数中添加截图、元素高亮，以及额外的任意操作，比如更多的详细日志输出等等。

上面所讲的 GUI 测试报告都是针对一个国家的，当面对多个国家站点的 GUI 测试时，事情就会变得更加复杂，你就必须去考虑全球化 GUI 测试报告应该如何设计。

**全球化 GUI 测试报告的创新设计：**所谓全球化测试是指，同一个业务在全球各个国家都有自己网站。比如，一些大型全球化电商企业在很多国家都有自己的站点，那么对这些站点的测试除了要关注基本的功能，以及各个国家特有的功能外，还要去验证界面布局以及翻译在上下文环境中是否合适。

早期的做法是，雇佣当地的测试工程师，由他们手工执行主要的业务场景测试，并验证相关的页面布局，以及翻译内容与上下文中的匹配度。在当地专门雇佣的这些测试工程师，被称为 LQA。但是比较低效。

为了解决这种低效的模式，最好的解决方法就是：利用 GUI 自动化测试工具生成完整的测试执行过程的截图。这样，LQA 就不再需要去手工执行测试用例了，而是直接分析测试报告中业务操作过程中 GUI 界面截图就可以了，然后发现页面布局问题或者是不恰当的翻译问题。

以下，是 eBay 在全球化站点测试中采用的方案，目前已经取得了很好地效果，降低了工作量的同时，还大幅度提高了全球化测试的质量：以下形式的测试报告。这里有一张图片展示了一份包含多国语言比较报告的示例



**二十四、**