代码随想录知识星球-测试常见面试题精选

前序

不少录友打算从开发岗转冲测开岗，但对测试相关的问题还一无所知。

为了让录友了解测试相关基础知识，在代码随想录知识星球里每天会给录友们安排每日一题，都是根据星球里面经高频题目 整理而来，如图：

结合代码随想录知识星球里录友们面试测开岗位的面经，提炼30道常问测试相关问题，整理成这份【代码随想录知识星球–测试常见面试题精选】。 这份文档，是结合了代码随想录知识星球里两年多来录友们的测开面经里高频问题以及对应的高频回答，**最后又精细整理，去冗精简而成**。

关于测开，全称是测试开发工程师，本PDF聚焦在测试部分，至于开发部分，其实和正常开发问的八股文是一样的，只不过 测开问的内容相对会简单一些。

本PDF主要帮助开发想转测开的录友 快速补充测试相关面试知识，帮助大家可以在面测开的时候尽快拿到offer。

本PDF是快速背诵版（问题+回答），**如果想检查自己的学习成果，可以按照 每日一题汇总链接 去回答问题**，链接里面只有问题，可以尝试锻炼自己独立回答，看看答的如何。

本文档只分享在 代码随想录知识星球 里，**作为星球录友转投测开岗位 突击的资料。**

1. 谈一谈你对测试的理解?

1. 我认为测试不仅仅是找出错误，而且还包括验证软件的功能、性能、可靠性以及用户体验等方面，发现其中的问题并及时解决。
2. 测试可以提前发现和修复缺陷，能减少未来的维护成本，对于保障软件的质量和提升用户满意度十分重要。
3. 我了解多种测试方法，比如功能测试、性能测试、安全测试等，知道多种测试工具的使用。

2. 测试开发需要哪些知识?

1. 编程技能：测试人员需要掌握至少一种编程语言，Java、Python，能够编写一些自动化测试脚本和工具。
2. 掌握测试基本知识，比如理解不同类型的测试，如单元测试、集成测试、系统测试和验收测试，掌握黑盒测试和白盒测试的原理和适用场景。
3. 知道如何进行测试需求分析，并能够设计有效的测试用例和测试计划。
4. 熟悉常用的的测试工具和框架，例如Selenium，JMeter、Jenkins 等。

3. 测试需要哪些能力?

1. 分析能力：测试开发首先要能够理解复杂的软件系统和业务需求，并设计有效的测试用例。
2. 技术能力：对于测试开发职位，编程能力也是必要的，测试开发要求测试人员了解不同的编程语言和测试框架，能够编写自动化测试脚本。
3. 细节和认真观察的能力：测试要对细节高度敏感，善于发现bug。
4. 沟通能力：要与开发团队、产品经理进行沟通，确保测试的结果被正确传达。
5. 学习能力：测试是一个不断发展的领域，需要学习相关的测试知识和业务知识。
6. 解决问题的能力：遇到问题时，能够有效地分析问题根源并提出解决方案。

4. 讲一下你们的测试流程

我参与项目和实习中的测试流程通常按照下面这些步骤进行：

1. 需求分析：测试首先要对软件需求有着深入的理解，所以我们会开需求评审会议，分析和讨论需求。
2. 指定测试计划：这个计划会明确测试的范围、方法、资源分配、时间表和目标。
3. 测试用例设计：基于测试计划，设计详细的测试用例。这些用例应该覆盖所有功能点，包括正常情况和边界情况的测试。
4. 测试用例评审
5. 搭建和配置适合的测试环境，执行测试用例，记录测试结果（找 bug), 将在测试过程中发现的缺陷报告给开发团队
6. 回归测试：每当代码发生更改后，执行回归测试以确保新的更改没有破坏现有的功能
7. 性能测试：评估系统的响应时间、稳定性和扩展性。
8. 部署项目到预生产环境，在预生产环境测试
9. 编写测试报告，总结测试活动的结果，包括测试覆盖率、发现的缺陷和未解决的问题。
10. 项目上线后，根据反馈进行复盘和总结。

实际的工作流程可能会根据公司的具体情况和项目的特点有所不同。

5. 针对测试流程，会输出哪些材料

1. 测试计划：描述整个测试过程的策略、范围、资源、时间表和目标。
2. 测试用例：详细的步骤、预期结果和测试数据，用于验证软件功能是否符合需求。
3. 测试脚本：自动化测试的脚本代码
4. 错误报告：在测试过程中发现的缺陷，包括缺陷描述、重现步骤、影响范围和严重程度。
5. 测试执行日志：记录测试执行的详细过程，包括测试用例的执行结果。
6. 测试环境配置文档
7. 测试报告：总结整个测试周期的活动、发现的缺陷、测试覆盖率和最终的测试评估。

6. 请你说一下测试的常用方法

1. 功能测试：检查软件的各项功能是否按照需求规格书执行，包括用户界面、数据库、安全性、功能等。
2. 单元测试：测试软件中最小的可测试部分，验证这些单元在各种条件下都按预期工作
3. 集成测试：测试多个单元、模块或组件协同工作时是否能正常运行。
4. 系统测试：测试完整的、集成的软件系统来评估系统的符合度。通常包括功能性和非功能性测试。
5. 回归测试：在发生修改之后重新测试先前的测试用例以保证修改的正确性。
6. 性能测试：检查软件的速度、响应时间、稳定性、资源消耗等性能指标。包括负载测试、压力测试和稳定性测试。
7. 验收测试：确定软件是否满足其业务需求。验收测试是软件交付之前的最后一阶段测试。验收测试包括 `Alpha` 测试和 `Beta` 测试。

- Alpha测试：是由用户在开发者的场所来进行的，在一个受控的环境中进行。
- Beta测试：由软件的最终用户在一个或多个用户场所来进行的，开发者通常不在现场，用户记录测试中遇到的问题并报告给开发者，开发者对系统进行最后的修改，并开始准备发布最终的软件。

7. 你对单元测试和 集成测试有哪些了解

1. 单元测试是针对软件的最小可测试部分（通常是一个函数、方法或类）进行的测试。通常在编写或修改代码后立即进行，以快速发现和修正代码中的错误，常用的工具包括JUnit（Java）、PyTest（Python）等。
2. 集成测试是在多个模块或组件被集成在一起后进行的测试，用来验证不同模块之间的接口和交互是否按预期工作，通常使用集成测试框架，比如Postman（API测试）、Selenium（Web应用集成测试）来进行。

- 增量集成：逐步添加新的模块并测试。
- 大爆炸集成：同时集成所有模块后一次性测试。

8. 系统测试和集成测试的区别和使用场景是什么

1. 系统测试是在整个软件系统完成集成后进行的测试。它的目的是验证整个系统是否符合指定的需求，关注整个系统的行为，测试涵盖所有集成的模块，以确保它们作为一个完整的系统正确地协同工作，包含功能性测试（如功能完整性、用户界面、用户流程）和非功能性测试（如性能、安全性、兼容性）。
2. 集成测试是在多个软件模块或组件被集成在一起时进行的测试。它的目的是验证这些模块或组件之间的交互，关注于模块之间的接口和交互。确保不同模块的数据交换和功能协作符合预期，主要用来检查数据传递、接口调用、异常处理等模块间交互的方面。

集成测试通常在单元测试之后、系统测试之前进行，当整个应用开发接近完成时，进行系统测试。

9. 什么是黑盒测试

黑盒测试，也被称为功能测试或行为测试，测试者只关注软件的输入和输出，不需要了解程序的内部实现，主要验证软件的功能是否符合用户需求和规格说明。常用的测试方法包括**等价类划分、边界值分析**、因果图法、状态转换测试、错误猜测等。

黑盒测试：想象你在玩一款新游戏，你只关心游戏的功能、操作和画面，而不需要知道游戏的源代码或内部实现。你测试游戏的可玩性、故事情节等，这就是黑盒测试。

10. 什么是白盒测试

白盒测试，也称为结构测试或透明盒测试，测试者需要了解程序的内部工作机制，包括代码、逻辑流程、内部结构，主要验证代码的逻辑路径、分支覆盖、循环、语句覆盖等，常用的测试方法包括路径覆盖、条件覆盖、循环覆盖、语句覆盖等，主要适用于单元测试和集成测试。

白盒测试：如果你是游戏开发者，你可能需要检查游戏的源代码，确保每个游戏功能都按照设计要求正确实现。这就是白盒测试。

11. 说一下等价类划分和边界值法

1. 等价类划分

等价类划分是将系统的输入域划分为若干部分，然后从每个部分选取少量代表性数据进行测试。等价类划分认为如果一个测试用例在某个等价类中的一个值上通过测试，那么它在这个类中的其他值上也会通过，适用于输入数据较多的情况，有助于减少测试用例的数量并保证覆盖率。

- **有效等价类**：符合规格说明的输入条件。
- **无效等价类**：不符合规格说明的输入条件。

通过测试有效等价类来验证系统的正确性，通过无效等价类来验证系统的健壮性。

2. 边界值法

软件错误往往发生在输入或输出范围的边缘，所以边界值分析专注于测试输入数据的边界条件，而不是中间值，包括正常边界值（最大、最小值）和异常边界值（最大值+1、最小值–1），适用于测试那些对输入数据有明确范围或限制的功能。

12. 等价类划分的难点是什么

1. 正确定义等价类：确定等价类的难点在于正确理解需求和规格说明。如果对需求的理解不准确或不全面，可能导致等价类的定义不准确，从而影响测试覆盖率和有效性。
2. 识别有效和无效的等价类：不仅要识别有效的等价类（符合规格说明的输入条件），还要能够识别出无效的等价类（不符合规格说明的输入条件）
3. 处理复杂的输入条件：当输入条件非常复杂或者相互依赖时，定义清晰且准确的等价类变得更加困难。

13. 接口测试用例的编写需要注意哪些要点

1. 明确接口规格：：理解接口的功能、输入输出参数、数据格式、请求方法（如GET、POST、PUT、DELETE等）和预期的行为。
2. 返回值：各种情况下（正确的输入值和异常的输入值）下的响应内容是否正常
3. 接口的业务逻辑和功能是否正常
4. 数据库校验
5. 性能测试（接口 tps, 响应时间等）
6. 安全性，敏感信息加密，权限控制等。

14. 接口测试有哪些工具

- `Postman`：API测试工具，用于发送各种HTTP请求，并检查响应，支持自动化测试脚本编写。
- `JMeter`:主要用于性能测试和负载测试，但也可以用于API测试。
- `Swagger UI` 用于设计、构建、文档化和测试REST API的工具

15. 你是怎么测试接口的

1. 理解接口文档，了解接口的业务功能，请求方法、请求参数、响应结构、错误码以及对应的数据库存储
2. 编写测试用例，涵盖正常的输入情况（验证接口的功能性）和异常的输入情况（验证接口的健壮性和错误处理
3. 使用测试工具，比如 `Postman` 执行测试用例，观察响应是否符合预期，验证响应的状态码、响应体内容、响应时间等。

16. 性能测试时，一般关注哪些指标

- **TPS:**每秒事务数，代表了性能的好坏，TPS越高，性能越好
- 平均响应时间：请求的平均消耗时间，时间越短，性能越好
- **并发数:**同时向服务端发起请求的虚拟用户数，在不同的工具里可以用多个进程/线程来实现
- **错误率:**失败的请求比例

17. 功能测试用例一般包含哪些内容

1. 测试用例ID：一个唯一标识符，用于区分和引用测试用例。
2. 测试用例标题：简短描述测试用例的目的或主要功能。
3. 功能模块：指明此测试用例所属的软件功能模块或部分。
4. 测试目的/描述：对测试用例的目标和测试内容的详细描述。
5. 前置条件：执行测试用例之前需要满足的条件，如特定的系统状态或配置。
6. **测试步骤：**详细描述如何执行测试，包括用户如何与系统交互，每一步应该输入什么数据，选择哪些选项等。
7. **测试数据**：在测试中使用的具体数据，包括输入值和需要验证的输出值。
8. **预期结果**：描述在成功执行测试步骤后预期的系统行为或输出。
9. **实际结果**：在执行测试后记录的实际结果，用于与预期结果进行比较。
10. 通过/失败标准：定义何种条件下测试用例被认为是通过或失败。
11. **测试环境**：描述执行测试用例所需的软件、硬件、网络配置等环境信息。
12. **备注信息**：任何额外的信息，比如相关的依赖、特殊注意事项等。
13. **缺陷/问题ID**：如果测试失败，关联的缺陷或问题的标识符。

18. 如何写测试用例

- 测试人员尽早介入，彻底理解清楚需求，这个是写好测试用例的基础
- 如果以前有类似的需求，可以参考类似需求的测试用例，然后还需要看类似需求的bug情况
- 清楚输入、输出的各种可能性，以及各种输入的之间的关联关系，理解清楚需求的执行逻辑，通过等价类、边界值、判定表等方法找出大部分用例
- 找到需求相关的一些特性，补充测试用例
- 根据自己的经验分析遗漏的测试场景
- 多总结类似功能点的测试点，才能够写出质量越来越高的测试用例
- 书写格式清晰

19. 请你说一下设计测试用例的方法

黑盒测试方法：

- 等价类划分法：将输入数据划分为不同的等价类，每个等价类都有相似的行为。然后从每个等价类中选择测试用例。
- 边界值分析法：关注输入值的边界情况，测试接近边界值和边界之间的情况。
- 因果图法：使用因果图来识别和描述系统中各种因果关系，辅助设计测试用例。
- 决策表测试：创建决策表，列出不同的输入组合和相应的输出，确保所有可能的组合都得到测试。
- 状态转换测试：适用于有状态的系统，测试系统在不同状态下的行为和状态之间的转换。

白盒测试方法：

- 语句覆盖：确保每个源代码语句都至少执行一次。测试用例的目标是覆盖代码的所有语句。
- 分支覆盖：确保每个分支语句都至少执行一次，以测试代码中的条件语句。
- 路径覆盖：通过执行代码的所有可能路径来测试系统，包括所有可能的条件分支和循环。
- 条件覆盖：测试代码中条件表达式的所有可能取值，以确保所有条件的不同情况都被覆盖。
- 循环覆盖：确保测试覆盖了循环的不同情况，包括循环的入口、中间和退出。

20. 如何提高用例的覆盖率，减少漏测

1. 根据需求文档编写用例，确保每条需求都能被对应的用例覆盖

2. 要充分理解业务，挖掘隐形需求，并编写对应的用例
3. 除了正常的业务场景，多考虑一些异常的场景和数据
4. 要从多个维度对软件进行测试，功能、性能、安全等各方面来考虑
5. 多站在用户的角度去思考问题，模拟用户的使用场景
6. 组织用例评审

21. bug 的生命周期

1. New:（新的）

当某个“bug”被第一次发现的时候，测试人员需要与项目负责人沟通以确认发现的的确是一个bug，如果被确认是一个bug，就将其记录下来，并将bug的状态设为New

2. Assigned（已指派的）

当一个bug被指认为New之后，将其反馈给开发人员，开发人员将确认这是否是一个bug，如果是，开发组的负责人就将这个bug指定给某位开发人员处理，并将bug的状态设定为“Assigned”

3. Open（打开的）

一旦开发人员开始处理bug的时候，他（她）就将这个bug的状态设置为“Open”，这表示开发人员正在处理这个“bug

4. Fixed（已修复的）

当开发人员进行处理（并认为已经解决）之后，他就可以将这个bug的状态设置为“Fixed”并将其提交给开发组的负责人，然后开发组的负责人将这个bug返还给测试组

5. Pending Reset（待在测试的）

当bug被返还到测试组后，我们将bug的状态设置为Pending Reset”

6. Reset(再测试)

测试组的负责人将bug指定给某位测试人员进行再测试，并将bug的状态设置为“Reset”

7. Closed（已关闭的）

如果测试人员经过再次测试之后确认bug 已经被解决之后，就将bug的状态设置为“Closed”

8. Reopen（再次打开的）

如果经过再次测试发现bug（指bug本身而不是包括因修复而引发的新bug）仍然存在的话，测试人员将bug再次传递给开发组，并将bug的状态设置为“Reopen”

9. Pending Reject（拒绝中）

如果测试人员传递到开发组的bug被开发人员认为是正常行为而不是bug时，这种情况下开发人员可以拒绝，并将bug的状态设置为“Pending Reject”

10. Rejected(被拒绝的)

测试组的负责人接到上述bug的时候，如果他（她）发现这是产品说明书中定义的正常行为或者经过与开发人员的讨论之后认为这并不能算作bug的时候，开发组负责人就将这个bug的状态设置为“Rejected”

11. Postponed（延期）

有些时候，对于一些特殊的bug的测试需要搁置一段时间，事实上有很多原因可能导致这种情况的发生，比如无效的测试数据，一些特殊的无效的功能等等，在这种情况下，bug的状态就被设置为“Postponed“

Bug类型

- 代码错误
- 界面优化
- 设计缺陷
- 配置相关
- 安装部署
- 安全相关
- 性能问题
- 标准规范
- 测试脚本
- 其他

22. 常见的测试工具有哪些?

1. 自动化测试工具

自动化测试工具用过selenium和appium

- Selenium: 用于自动化Web应用程序测试的工具，支持多种浏览器和多种编程语言。
- Appium：用于自动化移动应用程序测试的开源工具，支持iOS和Android平台。

2. 性能测试工具

- JMeter：用于测试Web应用程序的性能和负载。
- LoadRunner：支持模拟大量用户并测量系统的性能。

3. 接口测试

- Postman: 用于API开发和测试的流行工具
- Swagger UI：用于设计、构建、文档化和测试REST API的工具。

4. 单元测试

- Pytest：用于Python应用程序的测试框架，支持简单和复杂的测试场景。
- JUnit：用于Java应用程序的单元测试框架，支持自动化测试脚本的执行和报告生成。

其他：

- Fiddler是一个常用的抓包工具

23. 说一说你知道的自动化测试框架

1. pytest 是 Python 的一种单元测试框架，能够支持简单的单元测试和复杂的功能测试，还可以用来做 selenium / appnium 等自动化测试、接口自动化测试（pytest+requests）。
2. Junit 是一个 Java 语言的单元测试框架
3. Selenium 是一个用于 Web 应用程序测试的工具，测试与浏览器的兼容性
4. appium 是一个开源自动化测试工具，支持 iOS 和 Android 平台上的原生应用、Web 应用以及混合应用。
5. LoadRunner，通过模拟上千万用户实施并发负载来进行性能测试

24. App 测试和 Web 测试有什么区别

1. web和app的区别

- web 项目 ，一般都是b/s架构，基于浏览器的。
- App则是C/S的，必须要有 客户端 。那么在系统测试的时候就会产生区别了。

首先从系统架构来看的话，Web测试只要更新了服务器端， 客户端 就会同步会更新。而且 客户端 是可以保证每一个用户的 客户端 完全一致的。但是App端是不能够保证完全一致的，除非用户更新 客户端 。如果是App下修改了服务端，意味着 客户端 用户所使用的核心版本都需要进行回归测试一遍。

2. 性能方面

- web页面可能只会关注响应时间。
- App则还需要关心流量、电量、CPU、GPU、Memory这些了。

3. 兼容方面

- Web是基于浏览器的，所以更倾向于浏览器和电脑硬件，电脑系统的方向的兼容，不过一般还是以浏览器的为主。而浏览器的兼容则是一般是选择不同的浏览器内核进行测试（IE、chrome、Firefox）。
- App的测试则必须依赖phone或者是pad，不仅要看分辨率，屏幕尺寸，还要看设备系统。系统总的来说也就分为Android和iOS，不过国内的Android的定制系统太多，也是比较容易出现问题的。

4. 相比较web测试，app更是多了一些专项测试：

- 一些异常场景的考虑以及弱网络测试。这里的异常场景就是中断，来电，短信，关机，重启等。

而弱网测试是App测试中必须执行的一项测试。包含弱网和网络切换测试。需要测试弱网所造成的用户体验，重点要考虑回退和刷新是否会造成二次提交。需要测试丢包，延时的处理机制。避免用户的流失。

- 安装、卸载、更新：

web测试是基于浏览器的所以不必考虑这些。而app是 客户端 的，则必须测试安装、更新、卸载。除了常规的安装、更新、卸载还要考虑到异常场景。包括安装时的中断、弱网、安装后删除安装文件，更新的强制更新与非强制更新、增量包更新、断点续传、弱网，卸载后删除App相关的文件等等。

- 界面操作

现在app产品的用户都是使用的触摸屏手机，所以测试的时候还要注意手势，横竖屏切换，多点触控，事件触发区域等测试。

25. 软件质量的六个特征

任何事物的用例设计，需要按照软件质量六大特征来分类说明

1. **功能性**：表示软件能够提供满足明确和隐含需求的功能。它涉及适用性、准确性、互操作性、安全性和符合性。
2. **可靠性**：指软件在特定条件下持续正常运行的能力。包括成熟度、容错性、恢复能力等。
3. **易用性**：软件的界面和功能对用户是否友好，是否易于理解、学习、使用和吸引用户。包括可理解性、易学性、操作性、吸引力和用户界面的适用性。
4. **效率**：指软件在特定条件下对系统资源的有效使用。这涉及时间效率和资源效率。
5. **可维护性**：指在必要时能够有效地进行修正和改进软件的能力。这包括模块化、可重用性、分析性、修改性和测试性。
6. **可移植性**（Portability）：表示软件能够从一个环境迁移到另一个环境的能力。包括适应性、安装性、共存性和更换性。

26. 如果做一个杯子的检测，你怎么测试

从软件质量的六大特征（功能性、可靠性、易用性、效率、可维护性、可移植性）的角度来看，对一个实体产品如杯子进行类比测试，可以这样考虑：

1. 功能性
- 适用性：测试杯子是否适合其设计用途，如饮用水、茶、咖啡等。
- 准确性：确保杯子的容量标示准确。
- 符合性：检查杯子是否符合相关的健康和安全标准。

2. 可靠性
- 成熟度：确保杯子在正常使用条件下不会轻易破裂或损坏。
- 容错性：测试杯子在非常规使用（如高温、跌落）情况下的性能。

3. 易用性
- 理解性：检查杯子是否容易被用户正确使用（如是否明显标示不可微波）。
- 学习性：对于特殊设计的杯子（如带温度显示），测试用户学习如何使用的难易程度。
- 操作性：确保杯子设计方便持握和使用。

4. 效率
- 时间行为：对于保温杯，测试保温效果持续的时间。
- 资源利用：评估杯子设计在材料利用上的效率。

5. 可维护性
- 分析性：检查是否容易识别杯子的损坏或磨损。
- 修改性：评估修复损坏杯子的难易程度（尽管大多数杯子可能是一次性的）。

6. 可移植性
- 适应性：测试杯子是否适用于各种环境（如室内、户外）。
- 安装性：考虑杯子是否容易清洁、存储。

27. 给你一个页面，如何开展测试

- UI测试：页面布局、页面样式检查、控件长度是否够长；显示时，是否会被截断；支持的快捷键，Tab键切换焦点顺序正确性等。
- 功能测试：页面上各类控件的测试范围，测试点。结合控件的实际作用来补充检查点： 比如， 密码框是否*显示， 输入是否做 `trim` 处理等。
- 安全测试：输入特殊字符， `sql` 注入，脚本注入测试。后台验证测试，对于较重要的表单 ，绕过js检验后台是否验证；数据传输是否加密处理，比如， 直接请求转发，地址栏直接显示发送字符串?
- 兼容性测试
- 性能测试

28. 请你说一说用户界面登录过程都需要做哪些测试

1. 功能测试

- 输入正确的用户名和密码，点击提交按钮，验证是否能正确登录。
- 输入错误的用户名或者密码,验证登录会失败，并且提示相应的错误信息。
- 登录成功后能否能否跳转到正确的页面
- 用户名和密码，如果太短或者太长，应该怎么处理
- 用户名和密码，中有特殊字符（比如空格），和其他非英文的情况
- 记住用户名的功能
- 登陆失败后，不能记录密码的功能
- 用户名和密码前后有空格的处理
- 密码是否非明文显示显示，使用星号圆点等符号代替。
- 牵扯到验证码的，还要考虑文字是否扭曲过度导致辨认难度大，考虑颜色（色盲使 用者），刷新或换一个按钮是否好用
- 登录页面中的注册、忘记密码，登出用另一帐号登陆等链接是否正确
- 输入密码的时候，大写键盘开启的时候要有提示信息。
- 什么都不输入，点击提交按钮，检查提示信息。

2. 界面测试

- 布局是否合理
- 界面的设计风格是否与UI的设计风格统一。
- 界面中的文字简洁易懂，没有错别字。

3. 性能测试

- 打开登录页面，需要的时间是否在需求要求的时间内。
- 输入正确的用户名和密码后，检查登录成功跳转到新页面的时间是否在需求要求的时间内。
- 模拟大量用户同时登陆，检查一定压力下能否正常登陆跳转。

4. 安全性测试

- 登录成功后生成的Cookie，是否是httponly (否则容易被脚本盗取)。
- 用户名和密码是否通过加密的方式，发送给Web服务器。
- 用户名和密码的验证，应该是用服务器端验证， 而不能单单是在客户端用javascript 验证。
- 用户名和密码的输入框，应该屏蔽SQL注入攻击。

- 用户名和密码的的输入框，应该禁止输入脚本 （防止XSS攻击）。
- 防止暴力破解，检测是否有错误登陆的次数限制。
- 是否支持多用户在同一机器上登录。
- 同一用户能否在多台机器上登录。

5. 可用性测试

- 是否可以全用键盘操作，是否有快捷键。
- 输入用户名，密码后按回车，是否可以登陆。
- 输入框能否可以以Tab键切换。

6. 兼容性测试

- 不同浏览器下能否显示正常且功能正常（IE,6,7,8,9, Firefox, Chrome, Safari,等）。
- 同种浏览器不同版本下能否显示正常且功能正常。
- 不同的平台是否能正常工作，比如Windows, Mac。
- 移动设备上是否正常工作，比如Iphone, Andriod。
- 不同的分辨率下显示是否正常。

7. 本地化测试

- 不同语言环境下，页面的显示是否正确。

29. 测试提交一个 bug, 开发认为不是 bug，你会怎么办

1. 告知开发bug的判断依据，同时明确开发说不是bug的理由。
2. 对开发的理由进行校验，校验依据
    a. 参照需求文档
    b. 跟产品经理进行沟通确认
3. 校验结果不是bug，关闭bug，如果是bug提交给开发进行处理，确保产品质量

30. 发现一个 bug, 如何定位是客户端还是服务端的问题

1. 首先复现问题，确保能够可靠地重现这个bug。记录重现bug的具体步骤、输入、环境设置等。
2. 查看错误日志，通过查看客户端/服务端的日志，分析有没有异常的日志信息，从而确定具体原因
3. 分析客户端：使用开发者工具（如浏览器的开发者控制台）来检查网络请求、响应数据、控制台输出等，如果客户端收到的响应数据是正确的，但表现异常，可能是客户端问题
4. 分析服务端：检查服务端的处理逻辑。确保服务端正确处理了来自客户端的请求，并返回了正确的响应。如果服务端在处理请求时出现错误或返回了错误的数据，问题可能在服务端
5. 验证网络通信：还要确认客户端和服务端之间的网络通信是否正常。有的时候网络问题可能会导致错误。